JSTR
ASEAN Journal of Scientific and Technological Reports

*Research article*

# Improving Cluster-Based Index Structure for Approximate Nearest Neighbor Graph Search by Deep Learning-Based Hill-Climbing

**Munlika Rattaphun[1], Amorntip Prayoonwong[2*], Chih-Yi Chiu[3], and Kritaphat Songsri-in[4]**

[1] Faculty of Science and Technology, Nakhon Si Thammarat Rajabhat University, 80280, Thailand; munlika_rat@nstru.ac.th
[2] Faculty of Science and Technology, Suratthani Rajabhat University, 84100, Thailand; aprayoonwong@gmail.com
[3] Department of Computer Science and Information Engineering, National Chiayi University, Taiwan; cychiu@mail.ncyu.edu.tw
[4] Faculty of Science and Technology, Nakhon Si Thammarat Rajabhat University, 80280, Thailand; kritaphat_son@nstru.ac.th
* Correspondence: E-mail: aprayoonwong@gmail.com

**Abstract:** This study presents a novel approach to archive an excellent tradeoff between search accuracy and computation cost in approximate nearest neighbor search. Usually, the *k*-nearest neighbor (*k*NN) graph and hill-climbing algorithm are adopted to accelerate the search process. However, using random seeds in the original hill-climbing is inefficient as they initiate an unsuitable search with inappropriate sources. Instead, we propose a neural network model to generate high-quality seeds that can boost query assignment efficiency. We evaluated the experiment on the benchmarks of SIFT1M and GIST1M datasets and showed the proposed seed prediction model effectively improves the search performance.

**Keywords:** Inverted indexing; Nearest Neighbor Search; Nearest Neighbor Graph; Hill-climbing; Neural network;

**Citation:**
Rattaphun, M.; Prayoonwong, A.; Chiu, C.Y., Songsri-in, K. Title. *ASEAN J. Sci. Tech. Report.* **2022**, *25*(3), 25-33. https://doi.org/10.55164/ajstr.v25i3.247183.

**Article history:**
Received: August 3, 2022
Revised: September 20, 2022
Accepted: September 21, 2022
Available online: September 28, 2022



## 1. Introduction

The nearest neighbor (NN) search technique is widely applied in numerous fields, including computer vision, pattern recognition, signal processing, information retrieval, recommender systems, and so on [1-2]. Typically, the feature of each object of interest (such as an image) is represented in a high-dimensional space. A distance function is used to calculate distances between all data points and a given query. The NN is the data point with the smallest distance to the given query. When the number of data points and the number of data dimensions increase, an exhaustive search becomes impractical due to the expensive computation cost. To accomplish this task, numerous approximate nearest neighbor (ANN) [3-9] search methods are proposed to address the tradeoff between speed and accuracy.

The graph-based method is one of the most popular approaches to addressing the ANN search problem. The basic idea is that a neighbor of a neighbor is also likely to be a neighbor [5]. Most graph-based methods are based on constructing the *k*-nearest neighbor graph (*k*NN graph), built in the offline phase. A straightforward way to create the *k*NN graph is an exhaustive comparison

between each pair of vectors. Then, the top-$k$ nearest neighbors for each reference vector are selected to be the connected node in the $k$NN graph. When the query is given, the search process is started by traversing the chart to find the NN candidates.

Two main challenges are considered to improve the graph-based index structure performance: (1) how to effectively generate the $k$NN graph and (2) how to traverse the $k$NN graph to find the NN candidates efficiently. The first challenge focuses on reducing the computation cost of constructing the $k$NN graph. It can be addressed using an approximate $k$NN graph [5] [10-11]. In this paper, we focus on the second challenge. One popular method is the hill-climbing algorithm [12-13] which utilizes the $k$NN graph for ANN search. It generates multiple random seeds to traverse the $k$NN graph and takes several iterations to refine the traverse result. However, random seeds are easily trapped in local optima and frequently visit unlikely NN candidates. To address the problem and speed up the process, Zhao et al. [12] proposed using inverted indexing in the residual vector space and applying cascaded pruning to avoid redundant candidates. Still, it may take a long time to converge. We thus propose an index method that employs the $k$NN graph to accelerate the search process. The contributions of the proposed method are emphasized as follows:

- We propose to modify the hill-climbing algorithm with a novel seed generation method. Instead of using random seeds in the original hill-climbing algorithm, we generate high-quality seeds based on a neural network.
- The proposed model learns the relation between query features and clusters in the $k$NN graph, which can estimate the cluster probabilities for a given query and provides a better way to select the initial seed without constructing an extra index table.
- We evaluated the experiment on the benchmarks of SIFT1M and GIST1M datasets and showed the proposed methods effectively improve the search performance.

The remainder of this paper is organized as follows. Section 2 presents a brief review of the NN search related to the $k$NN graph and hill-climbing algorithm. Section 3 offers the detail of the proposed method. We demonstrate some experimental results in section 4 and give the conclusion in Section 5.

## 2. Related Works

To improve index structure for better performance of ANN search, we are interested in the collaboration of the following three techniques: $k$-nearest neighbor graph ($k$NN graph), hill-climbing algorithm, and inverted file-based (or inverted index) method. The detail of each process are summarized as follows:

The $k$NN graph is a graph-based index structure widely used for ANN search. A graph structure is added to the index structure to make the search more efficient. To avoid exhaustive search and decrease the computation cost in ANN search tasks, the $k$NN graph construction process can be performed in the offline phase. The $k$NN graph is a directed graph defined for a set of $N$ points in a metric space. The chart has a vertex for each data point, in which two vertices are connected by an edge whenever. The distance between those two vertices is among the $k^{th}$ smallest distances. An exhaustive comparison between each pair of vectors is performed until the top $k$ nearest neighbors are selected for each reference vector to generate a $k$NN graph. The computation complexity is about $O(DN)^2$, where $D$ is the number of data dimensions and $N$ is the number of data points [14]. Over the years, many researchers have applied and developed graph structures to increase the performance of ANN search in many forms. Zhang et al., 2013 [10] and Wang et al., 2012 [15] both constructed the $k$NN graph by randomly partitioning their samples into a small number of subsets with different technical details. [15] used the hierarchical random projections technique while [10] exploited the locality-sensitive hash functions. Combining the neighborhood graph with a bridge graph yields superior performance over large-scale datasets [16]. Two more popular graph-based index structure techniques are deployed to speed up searches: HNSW [8] and NSG [17]. HNSW is a hierarchical multi-level proximity graph that enables hopping with multi-scales on different graph layers. NSG intends to reduce the density of the graph edges while the search performance remains accurate.

The hill-climbing algorithm [12-13] is a mathematical optimization algorithm that often utilizes the $k$NN graphs for ANN search to find the best solution together with various possible solutions. The hill-climbing algorithm is a local search algorithm with the following working principles. First, it tries to find the best solution to the given problem by starting with a random seed (node or solution). It then evaluates the neighbor nodes. If the best of those neighbor nodes is better than the current node, it replaces the current

solution with this better solution. It subsequently continues to move in the direction of elevation or increasing values to find the peak of the local mountain or the best solution. The algorithm terminates when it reaches a peak value where no neighbor has a higher value. By starting to traverse the *k*NN graphs from a random seed and applying the abovementioned working principles, there is no guarantee that the best solution will always be found. It is easy for the algorithm to trap in a local maximum. Many researchers have proposed several approaches to address that problem in terms of speed and optimized values. For example, [12] used the combination of the inverted indexing in residual vector space and cascaded pruning to speed up the search by avoiding redundant candidates. [14] avoided using the random seed by incorporating new hash-based methods for seed generation: LSH [18] and ITQ [19]. These principles have been applied to generate higher quality seeds, which lead to increased accuracy and a lower computation cost.

The inverted file-based (or inverted index) method is another popular framework in multimedia retrieval. In text retrieval, an inverted index is used as a vocabulary of words, where each word has a list of the associated documents that contain this word. For image retrieval, the inverted index is used to store the codewords from the quantization process, and each codeword has a list of all vectors belonging to these codewords. The non-exhaustive search performs by checking only items in the codeword list. The inverted file-based method can be operated on the billion-scale dataset to avoid the exhaustive search and achieve good recall and speed in many works [20-22]. For instance, the inverted multi-index (IMI) [21] built a multi-dimensional index table, which is the Cartesian product of codebooks. IMI showed better accuracy without increasing the query time. The jointly inverted indexing [22] is a method that generates multiple quantizers and jointly optimizes all the codewords instead of computing the multiple quantizers independently. IVFADC [3] showed that the combination of inverted indexing with Product Quantization (PQ) [3-4] and Asymmetric Distance Computation (ADC) could handle billion-scale datasets efficiently.

In the past years, the learned index structure employed by machine learning techniques has been used in place of the existing index structure in many studies. For example, [23] proposed a combination of supervised classification and graph partitioning to build better-balanced graph partitioning. [9] presented the learned index structure by using the nearest neighbor probabilities model that employed neural networks to characterize the neighborhood relationships, which can rank and find candidate clusters of the given query effectively. A previous work [14] used the hashing approaches to generate a compact binary code to index the associated clusters in the inverted index table. When the query is given, the same hashing function is used to create binary code for each query. The query hash codes and hash code index in the inverted index are then mapped to retrieve the initial seeds for the hill-climbing algorithm. In this paper, the proposed method also brings up a higher quality seed generation method by utilizing the learned index structure. We present a novel method based on a neural network that predicts cluster probabilities. Then, top-*R* clusters are then selected as the initial seeds for hill-climbing.

## 3. Methods

The proposed method consists of two parts. Figure. 1 shows an overview of the proposed method. First, we construct a *k*NN graph using a clustering approach integrated with inverted indexing. Second, we offer a novel seed selection method based on the neural network approach. The model learns the relationship from the training data to produce the cluster probabilities. Then, the clusters are ranked based on their possibilities, and the top-*R* clusters are selected as the initial seeds in the hill-climbing algorithm to find a set of NN candidates. The details for each part are elaborated in the following subsections.

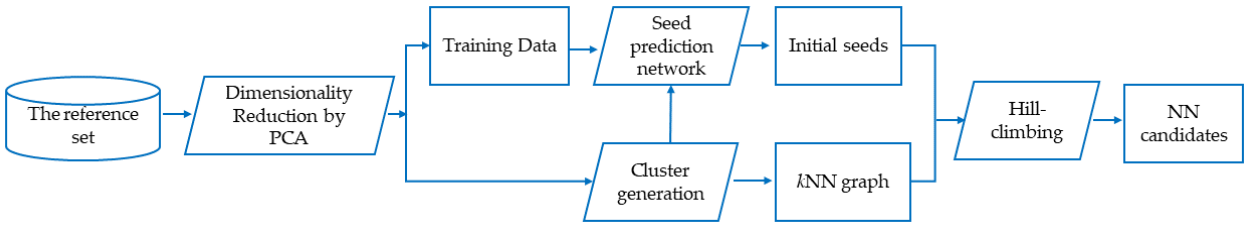


**Figure 1.** Overview of the proposed method

### *3.1. Cluster and kNN graph construction*

To further reduce the memory space and time complexity of building a *k*NN graph, we adopt Principal Component Analysis (PCA) to perform dimensionality reduction.

Given a reference set of data points $\{x_i \in \mathbb{R}^D | i = 1,2,\dots,N\}$, where $N$ is the number of data points and $D$ is the number of data dimensions. First, we applied PCA to reduce the dimensionality of $x_i$. To $d$ dimensions, where $d < D$. The PCA process is done by a linear transformation of $x_i$ from the original space $\mathbb{R}^D$ into a lower space $\mathbb{R}^d$. We first standardize the data by subtracting each $x_i$ with the mean and dividing it by the standard deviation of the reference set. Next, a covariance matrix, which is a $D \times D$ symmetric matrix, is computed. Each element in the covariance matrix reflects the covariance of the corresponding variables. After that, we can identify the principal components by computing the eigenvectors and eigenvalues of the covariance matrix. We selected the highest $d$ principal components to construct a projection matrix. Then, each data point $x_i$ is transformed into a new space with the projection matrix.

After we performed dimensionality reduction, the *k*-means clustering is adopted to divide the compressed data space into *M* clusters, $\{c_j | j = 1,2,\dots,M\}$ where $c_j$ represents the centroid of the *j*th cluster. Finally, we construct the *k*NN graph by calculating the distance between cluster centroids to find *k* nearest clusters. So, each cluster is associated with a list of *k* nearest centroids. After that, the inverted index is used to store *k* indexes.

### *3.2. Hill-climbing seed generation using neural network*

The main idea of our approach is to build a model that can estimate cluster probabilities for a given query. Then, $n$ clusters with the highest probabilities are selected as a set of the initial seed in the hill-climbing algorithm. Function $Z$ implicitly models the neighborhood relationships expressed as $Z(x_i) = \{p_1, p_2, \dots, p_m\}$, which maps a data point $x_i$ to the NN probabilities $= \{p_1, p_2, \dots, p_M\}$, where $p_M$ represents the NN probability of the $m$th cluster. The following training process constructs it. Let $Q = \{q^{(1)}, q^{(2)}, \dots, q^{(T)}\}$ be the training dataset of $T$ queries. The $t$th query $q^{(t)} \in \{\mathbb{R}^d\}$ is associated with the weighted ground truth of *k*NNs, denoted as $G^{(t)} = \{g^{(t,1)}, g^{(t,2)}, \dots, g^{(t,K)}\}$ and $W^{(t)} = \{w^{(t,1)}, w^{(t,2)}, \dots, w^{(t,K)}\}$ where $g^{(t,k)}$ is the $k$th NN of $q^{(t)}$and the corresponding weight is $w^{(t,k)}$. In $Z(x_i)$, the input $x_i$ is the query feature representation $q^{(t)}$. The output, also known as the target, is a vector of NN probabilities of $M$ clusters, denoted as, $Y^{(t)} = \{y_1^{(t)}, y_2^{(t)}, \dots, y_M^{(t)}\}$, where $y_M^{(t)}$ is defined by:

$$y_m^t = \frac{\sum_k \{w^{(t,k)} | g^{(t,k)} \in c_m\}}{|G^t|} \quad (1)$$

where $|\cdot|$ denote the set cardinality.

We characterized $Z$ by a fully-connected neural network to learn the neighborhood relationships from the training data $\{x^t, y^t | t = 1,2,\dots,T\}$. The input layer receives $x^{(t)}$ and the output layer predicts NN probabilities for *M* clusters, denoted as $p^{(t)} = \{p_1^{(t)}, p_2^{(t)}, \dots, p_M^{(t)}\}$. Based on the cross-entropy loss between the predictions $p^{(t)}$ and the target $y^{(t)}$, we computed the error derivative concerning the output of each unit, which is propagated backward to each layer to tune the weights of the neural network. The proposed seed prediction model is shown in Figure 2.

Given a query $q$, by using the learned mapping function $Z$, the model can predict the NN probability among clusters. We then rank these clusters based on their likelihood in descending order to return the top-$R$ cluster used as the initial seed in the hill-climbing algorithm.

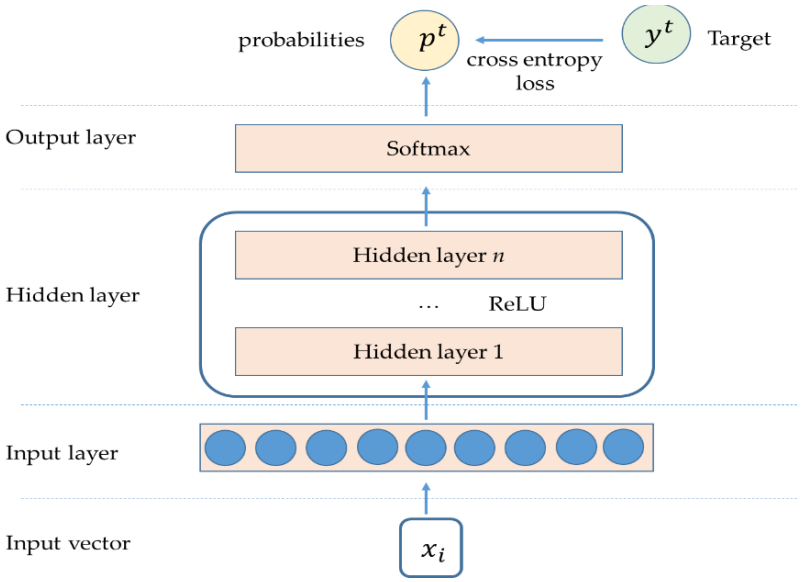


**Figure 2.** The seed prediction network

## 4. Experimental results

We performed experiments to evaluate the proposed method on the SIFT1M and GIST1M datasets of BIGANN. Each contains one million SIFT, GIST, and 10,000 and 1,000 query vectors, respectively. Each query is provided with the first 100 nearest neighbors of ground truth with the smallest Euclidean distances. The properties of the two datasets are summarized in Table 1. We applied PCA to reduce the dimensionality of SIFT1M from 128 to 32 dimensions and that of GIST1M from 960 to 120 dimensions. Afterward, we used *k*-means clustering to generate *M* clusters where $M \in \{256, 1024, 4096\}$.

**Table 1.** Summary of SIFT1M and GIST1M datasets

| Datasets | SIFT1M | GISR1M |
|---|---|---|
| #data dimensions | 128 | 960 |
| #Reference set | 1,000,000 | 1,000,000 |
| #Training set | 20,000 | 20,000 |
| #Queries set (Test set) | 10,000 | 1,000 |
| #Ground truth set (per query) | 100 | 100 |

The neural network model is characterized by a three-layer fully connected neural network, including an input layer, two hidden layers, and an output layer. The input layer receives the query feature as input, where the number of units equals the dimension of the input vector. The two hidden layers use ReLU as an activation function, each having an *M* unit. The output layer also has *M* units that predict the probability for *M* clusters using the softmax activation function. The detail of the configuration is elaborated in Table 2.

**Table 2.** The configurations of the neural network model with various sizes of the cluster.

| | The number of units | | | | | | Activation function |
|---|---|---|---|---|---|---|---|
| | SIFT1M | | | GIST1M | | | |
| #clusters | 256 | 1024 | 4096 | 256 | 1024 | 4096 | - |
| Input | 32 | 32 | 32 | 128 | 128 | 128 | - |
| Hidden | 256 | 1024 | 4096 | 256 | 1024 | 4096 | ReLU |
| Hidden | 256 | 1024 | 4096 | 256 | 1024 | 4096 | ReLU |
| Output | 256 | 1024 | 4096 | 256 | 1024 | 4096 | Softmax |

A training set is generated by randomly selecting 20,000 data points from the reference set to train the neural network model. We provide 100 NNs of ground truth calculated from Euclidean distance for each training data point. We set the batch size to 128 and ran 100 epochs for training. The test set consists of 10,000 and 1,000 queries, respectively, provided by the SIFT1M and GIST1M, as shown in Table 1.

We implemented several configurations for the hill-climbing algorithm as follows:

- **Exhaustive search**. This method uses the Euclidean distances to calculate the distance between a given query and cluster centroids and select clusters in a particular way so that the hill-climbing algorithm is not applied here.
- **Random seed hill-climbing** [12-13]. Initial seeds for hill climbing are generated randomly.
- **8-bit, 10-bit, and 12–bit LSH seed hill-climbing** [14]. LSH [18] is used to transform data points into 8, 10, and 12-bit hash codes.
- **8-bit, 10-bit, and 12-bit ITQ seed hill-climbing** [14]. ITQ [19] is used to transform data points into 8, 10, and 12-bit hash codes.
- **DNN** seed hill-climbing. This approach uses the proposed neural network model to rank the cluster according to their probabilities.

Other parameters were set as follows: the number of the neighboring clusters kept in the *k*NN graph of a cluster $k \in \{20,30\}$, and the number of seeds for hill-climbing search $s = \{10, 20, 30\}$. Experiments were run on a PC using Windows 10, with an Intel Core i7 3.4 GHz CPU and 32 GB of Ram. The program was implemented in Python.

We use the recall rate to measure the correctness of the NN search. Let $Q = \{q^1, q^2, \dots, q^T\}$ be a set of $T$ queries and $G = \{g^1, g^2, \dots, g^T\}$ be the ground truth, where $g^t$ is the first NN of ground truth for $q^t$. A recall is defined as:

$$recall = \frac{1}{T}\sum_{t=1}^{T} f(R^t), \qquad f(R^t) = \begin{cases} 1 & if\ g^t \in (R^t) \\ 0 & otherwise. \end{cases} \tag{2}$$

where $R^t$ is the retrieved set corresponding to the $q^t$. In addition, we counted the number of Euclidean distances calculated between query and cluster centroids to reflect the computation cost for each configuration.

Figures. 3 and 4 show the results under different *M* in SIFT1M and GIST1M, respectively, where *k* and *s* are fixed to 20. The X-axis denotes the number of Euclidean distance computations during the search process, and the Y-axis denotes the recall rate. The exhaustive method yields the best recall, which is served as the accuracy upper bound. However, the computation cost is the highest due to the exhaustive comparisons between the given query and all clusters. The other methods, such as random, LSH, ITQ, and DNN, run five iterations in the hill-climbing algorithm. It shows the recall rates get close to the upper bound with a few iterations and spend much fewer computations. More iterations are required to converge in a more significant number of clusters. The proposed deep learning-based methods outperform the random and hashing methods. It can get the highest recall rate at the first iteration. Using our seed prediction model can provide a better initial seed in a hill-climbing algorithm since the model can learn the relationship between the query feature and cluster to predict the probability for each cluster. Moreover, compared to the previous version of hash-based seeds [14], the proposed DNN method yields better results but requires no additional memory space to store the inverted index lookup table.

Figures. 5 and 6 show the results against different $k \in \{20, 30\}$ and number of seeds $s \in \{10, 20, 30\}$, where *M* is fixed at 4096. We observe that increasing *k* makes the convergence of recall faster than increasing *s*. However, it needs more space to store a larger *k*NN graph.

We also analyzed the memory usage of the proposed DNN model compared with the hash-based seed. The memory space consumed by the hash-based seed method is the inverted lookup table loaded into memory for real-time searching. The number of clusters determines the size of the inverted lookup table. Assume that *M* is the number of clusters, and it takes 2 bytes to store cluster-ID, $2M$ bytes are required for hash-based seed index. The neural network used for seed prediction occupies $8(\tau\lambda\text{^}2 + M\lambda)$, where the model has $\tau$ hidden layers, each of which contains $\lambda$ units, and each nan 8-bytes floating-point number represents each network coefficient.

The time complexity mainly involves two parts: index time and candidate calculation. The time spent on the index depended on the machine and the environment. Some programs merely cannot run very well in the same environment. Therefore, it would be better to analyze the complexity by calculating the number of candidates, making the result more comparative. The results in terms of the number of candidates are shown in figures 3-6.

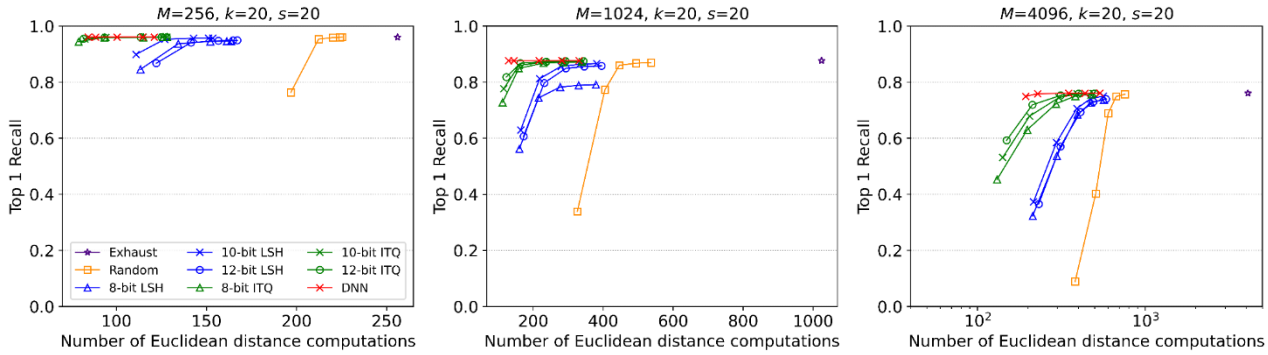


**Figure 3.** Recall in SIFT1M with different sizes of cluster, where *k* and *s* are fixed at 20

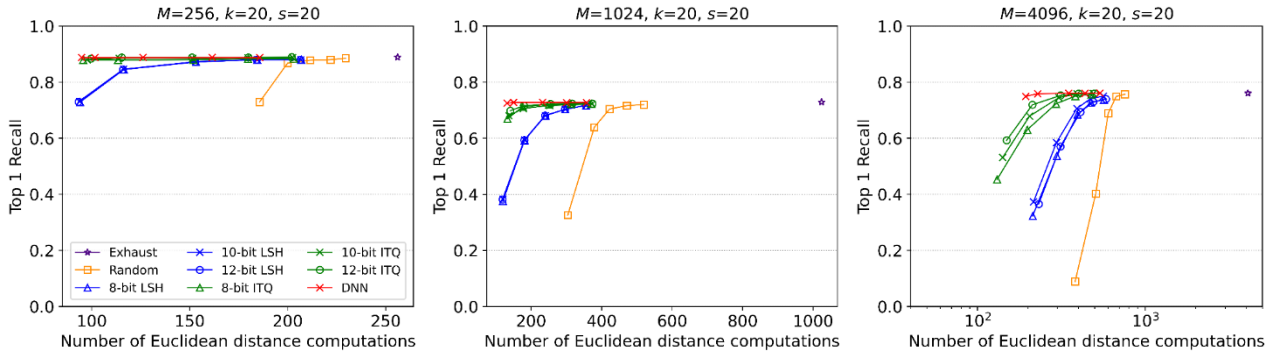


**Figure 4.** Recall in GIST1M with different sizes of cluster, where *k* and *s* are fixed at 20

## 4. Conclusions

In summary, we proposed a novel index method for ANN search that employs a *k*NN graph to accelerate the query assignment process. A modified hill-climbing algorithm is presented with the DNN-based seed generation method, which initializes high-quality seeds and thus improves the hill-climbing algorithm. Experimental results on the SIFT1M and GIST1M datasets demonstrate the superiority of the proposed method.

## 5. Acknowledgements

This study was supported by the Department of Computer Science and Information Engineering, National Chiayi University, Taiwan.

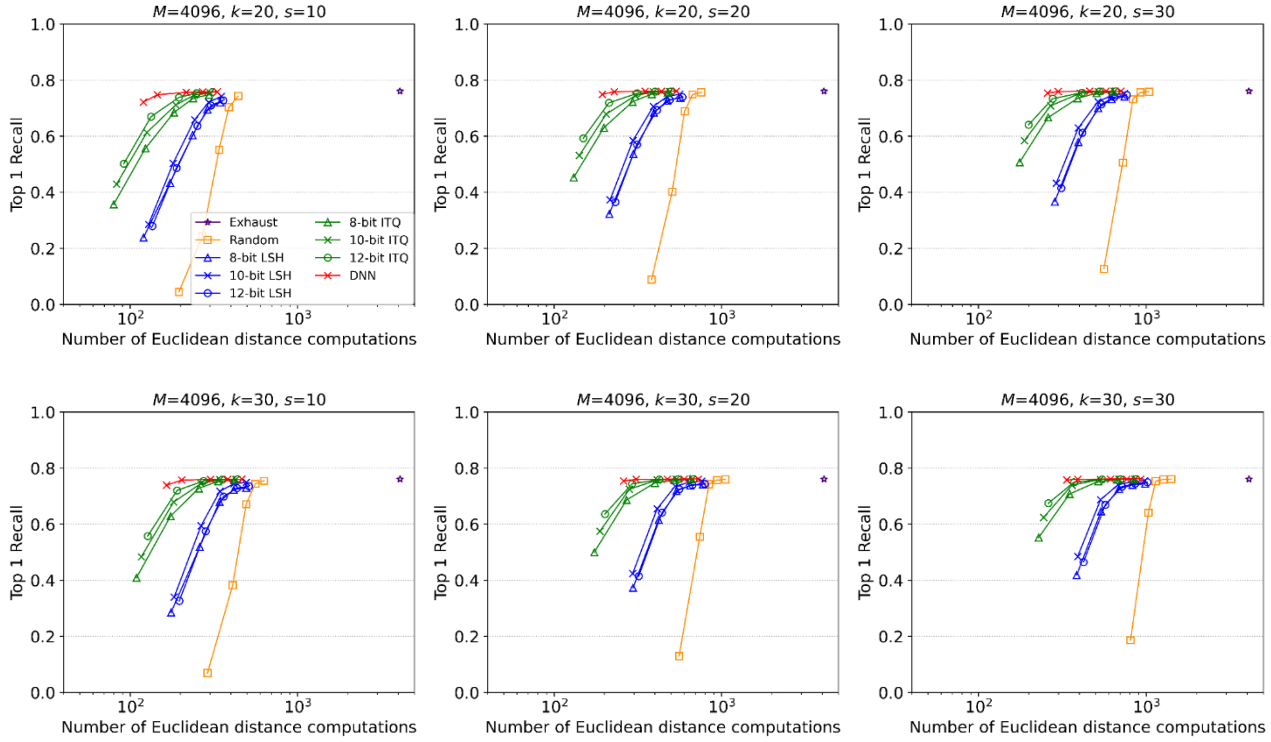


**Figure 5.** Recall in SIFT1M, where $M$ is fixed at 4096.

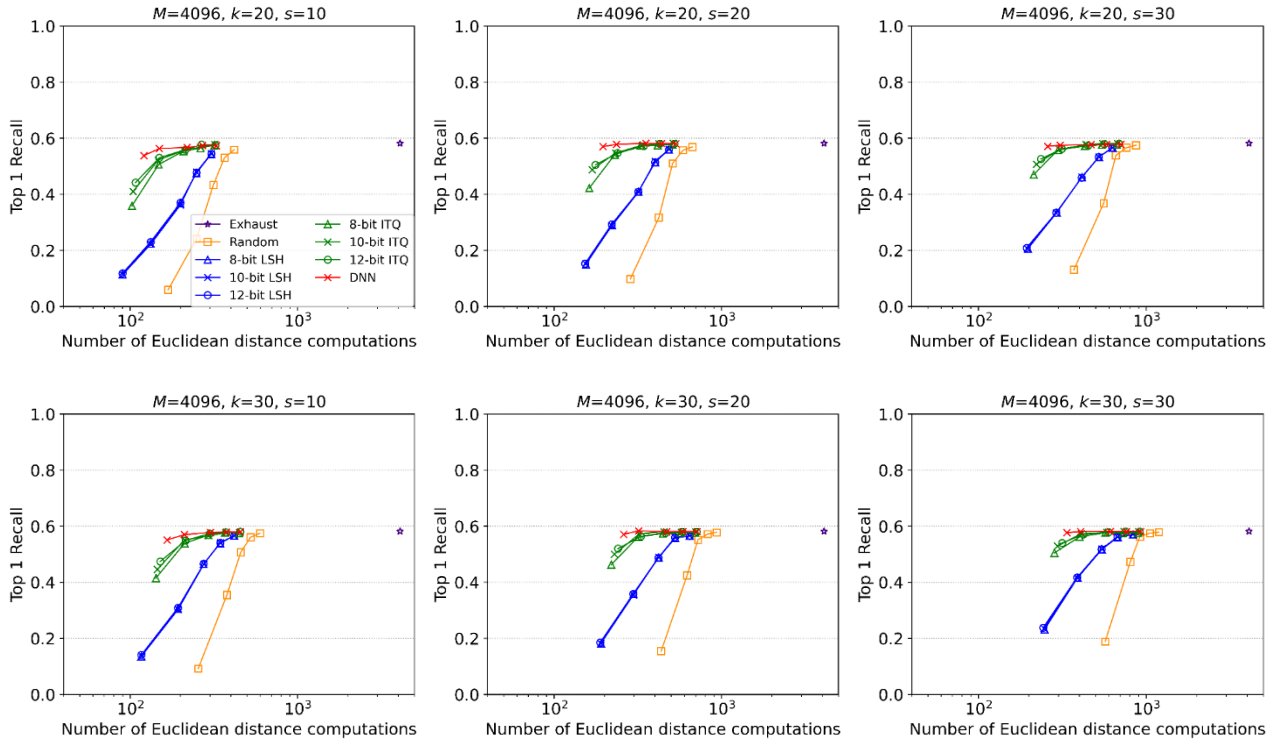


**Figure 6.** Recall in GIST1M, where $M$ is fixed at 4096.

## References


[1] Hu, P.; Peng, X.; Zhu, H.; Zhen, L.; Lin, J. Learning cross-modal retrieval with noisy labels. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, 5403-5413.

[2] Prétet, L.; Richard, G.; Peeters, G. Learning to rank music tracks using triplet loss. *In ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, 511-515.

[3] Jegou, H.; Douze, M.; Schmid, C. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*. 2011, *33*(1), 117-128.

[4] Ge, T.; He, K.; Ke, Q.; Sun, J. Optimized product quantization for approximate nearest neighbor search. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, 2946-2953.

[5] Fu, C.; Cai, D. Efanna: An extremely fast approximate nearest neighbor search algorithm based on *k*nn graph. 2016, *arXiv preprint arXiv:1609.07228*.

[6] Muja, M.; Lowe, D. G. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*. 2014, *36*(11), 2227-2240.

[7] Heo, J. P.; Lee, Y.; He, J.; Chang, S. F.; Yoon, S. E. Spherical hashing: Binary code embedding with hyperspheres. *IEEE transactions on pattern analysis and machine intelligence*. 2015, *37*(11), 2304-2316.

[8] Malkov, Y. A.; Yashunin, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*. 2018, *42*(4), 824-836.

[9] Chiu, C. Y.; Prayoonwong, A.; Liao, Y. C. Learning to index for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*. 2019, *42*(8), 1942-1956.

[10] Zhang, Y. M.; Huang, K.; Geng, G.; Liu, C. L. Fast kNN graph construction with locality sensitive hashing. *In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 2013, 660-674.

[11] Dong, W.; Moses, C.; Li, K. Efficient k-nearest neighbor graph construction for generic similarity measures. *In Proceedings of the 20th International Conference on World Wide Web*. 2011, 577-586.

[12] Zhao, W. L.; Yang, J.; Deng, C. H. Scalable nearest neighbor search based on KNN graph. *arXiv preprint arXiv:1701.08475* 2017.

[13] Hajebi, K.; Abbasi-Yadkori, Y.; Shahbazi, H.; Zhang, H. Fast approximate nearest-neighbor search with k-nearest neighbor graph. *In Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. 2011, 1312-1317.

[14] Rattaphun, M.; Prayoonwong, A.; Chiu, C. Y. Indexing in k-Nearest Neighbor Graph by Hash-Based Hill-Climbing. *In Proceedinngs of the 16th International Conference on Machine Visios Application (MVA)*. 2019, 1-4.

[15] Wang, J.; Wang, J.; Zeng, G.; Tu, Z.; Gan, R.; Li, S. Scalable k-nn graph construction for visual descriptors. In *Proceedinngs of IEEE Conference on Computer Vision and Pattern Recognition*. 2012, 1106-1113.

[16] Wang, J.; Wang, J.; Zeng, G.; Gan, R.; Li, S.; Guo, B. Fast Neighborhood Graph Search Using Cartesian Concatenation. *In Proceedings of the IEEE International Conference on Computer Vision*. 2013, 2128-2135.

[17] Fu, C.; Xiang, C.; Wang, C; Cai, D. Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph. *VLDB Endowment*. 2019, 461-474.

[18] Datar, M.; Immorlica, N.; Indyk, P.; Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. *In Proceedings of the 20th annual Symposium on Computational Geometry*. 2004, 253-262.

[19] Gong, Y.; Lazebnik, S.; Gordo, A.; Perronnin, F. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2013, *35*(12), 2916-2929.

[20] Ercoli, S.; Bertini, M.; Del Bimbo, A. Compact hash codes for efficient visual descriptors retrieval in large scale databases. *IEEE Transactions on Multimedia*, 2017, *19*(11), 2521-2532.

[21] Babenko, A.; Lempitsky, V. The inverted multi-index. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2014, *37*(6), 1247-1260.

[22] Xia, Y.; He, K.; Wen, F.; Sun, J. Joint inverted indexing. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2013, 3416-3423.

[23] Dong, Y.; Indyk, P.; Razenshteyn, I.; Wagner, T. Learning space partitions for nearest neighbor search. *arXiv preprint arXiv. 1901. 08544*, 2019.