16th International Conference on Machine Vision Applications (MVA)
National Olympics Memorial Youth Center, Tokyo, Japan, May 27-31, 2019.

Indexing in k-Nearest Neighbor Graph by Hash-Based Hill-Climbing

Munlika Rattaphun
National Chiayi University
munlika.r@gmail.com

Amorntip Prayoonwong
National Chiayi University
aprayoonwong@gmail.com

Chih-Yi Chiu
National Chiayi University
cychiu@mail.ncyu.edu.tw

Abstract

A main issue in approximate nearest neighbor search is to achieve an excellent tradeoff between search accuracy and computation cost. In this paper, we address this issue by leveraging k-nearest neighbor graph and hill-climbing to accelerate vector quantization in the query assignment process. A modified hill-climbing algorithm is proposed to traverse k-nearest neighbor graph to find closest centroids for a query, rather than calculating the query distances to all centroids. Instead of using random seeds in the original hill-climbing algorithm, we generate high-quality seeds based on the hashing technique. It can boost the query assignment efficiency due to a better start-up in hill-climbing. We evaluate the experiment on the benchmarks of SIFT1M and GIST1M datasets, and show the proposed hashing-based seed generation effectively improves the search performance.

Index Terms - inverted index, nearest neighbor graph, hill-climbing, hashing.

1 Introduction

Nowadays, nearest neighbor (NN) search in a large-scale and high-dimensional dataset is a challenging task in many research communities. NN search is motivated by the need of diverse applications in computer vision, information retrieval, and machine learning. NN search can be considered as an optimization problem of finding a collection of closest data to a given query. To avoid exhaustive search, numerous approximate nearest neighbor (ANN) search methods are proposed under the consideration of the balance between speed and accuracy.

Vector quantization (VQ) is one of the most popular approaches to address ANN search problems. VQ divides the data space into clusters, each of which can be approximately represented by its centroid. To quantize a data point, we have to compute distances/similarities for the data point with all centroids to find the nearest one. The computational complexity is thus about $O(nD)$, where n is the number of centroids and D is the data dimensions. When a query is given to be soft assigned with the k nearest centroids, it has to compute with all centroids and then sort them, taking $O(nD+n\lg k)$ time. The computation overhead is not negligible for large n and D.

Another effective approach for ANN search is kNN graph [1]. The kNN graph is constructed offline for each data point, which keeps the closest neighbors according to a given distance/similarity metric. With affordable space overhead, kNN graph characterizes useful neighborhood relationships embedded in the data space to facilitate ANN search.

In this paper, we propose an index method that employs kNN graph to accelerate the query assignment process. We adopt the hill-climbing algorithm that traverses kNN graph to find closest centroids for a query, rather than calculating the query distances to all centroids. Moreover, a modified hill-climbing algorithm is presented with a novel seed generation method. That is, instead of using random seeds in the original hill-climbing algorithm, we generate high-quality seeds by leveraging the hashing technique [2][3]. It boosts the query assignment efficiency due to a better start-up of the hill-climbing algorithm. Experiments are demonstrated on two benchmark datasets of SIFT1M and GIST1M. Results show the proposed hashing-based seed generation can effectively improve the search performance.

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

2 Related Work

The kNN graph is an index structure proposed to avoid exhaustive search in ANN search tasks. A straightforward way to construct kNN graph is an exhaustively comparison between each pair of vectors in the offline phase. Then, top k nearest neighbors for each reference vector are selected to generate kNN graph. The computation complexity is about $O(DN^2)$, where N is the number of data points. In order to decrease the computation cost, some approximate kNN graph are proposed by using the graph structure [4], tree structure [5][6] and hashing technique [7]. Although these approaches perform well, they have the problem of large memory consumption.

The hill-climbing algorithm [1][8] is a popular way to utilize kNN graph for ANN search. It generates multiple random seeds to traverse kNN graph, and takes several iterations to refine the traversal result. However, random seeds are easily trapped in local optima and often visits unlikely NN candidates. To address the problem and speedup the process, Zhao et al. [8] proposed using inverted indexing in residual vector space and applying cascaded pruning to avoid redundant candidates. Still, it may take much time to converge to a satisfactory accuracy.

3 Hash-Based Hill-Climbing

The proposed method consists of two parts. We first construct kNN graph integrated with inverted indexing. Then we apply a modified hill-climbing algorithm

to traverse kNN graph for ANN search. Details are elaborated in the following.

3.1 kNN graph construction

Given a reference set of data points $\{x_i \in \mathbb{R}^D | i = 1, 2, ..., N\}$, where N is the number of reference data points and D is the number of data dimensions. We apply principle component analysis (PCA) to reduce the dimensionality of x_i to d dimensions. Then k-means clustering is used to divide the compressed data space into M clusters $\{c_j | j = 1, 2, ..., M\}$, where c_j represents the centroid of the jth cluster. Finally, we construct kNN graph, as summarized in Algorithm 1. Each cluster is associated with a list of k nearest centroids that are closest to the cluster, where in step 4, function $\arg\min_j(A, k)$ returns k indexes with the smallest values in set A.

Algorithm 1: kNN graph construction

Input : clusters $\{c_j | j = 1, 2, ..., M\}$; the number of nearest centroids for each cluster k;
Output: kNN graph for the clusters;

1 **for** $i = 1, ..., M$ **do**
2 **for** $j = 1, ..., M$ **do**
3 $distance[j] \leftarrow \|c_i - c_j\|^2$;
4 $\phi \leftarrow \arg\min_j(distance[j], k)$;
5 **return** ϕ for c_i;

3.2 Hill-climbing seed generation

To improve the performance of the hill-climbing algorithm, we present a novel seed generation method based on the hashing technique. The cluster centroids are hashed into binary codes, which are the index keys of an inverted table. When a query is given, we match its hash code in the inverted table and take the associated centroids from kNN graph as the initial seeds of hill-climbing. Unlike random seeds, the hash-based seeds are generally distributed around the query, providing a better initialization for hill-climbing.

We employ two hashing methods, namely, locality-sensitive hashing (LSH) [2] and iterative quantization (ITQ) [3] to produce the hash function f. LSH employs random projection to embed a data point into a binary code. It states the probability that two points are hashed to the same bit is proportional to their similarity. Further, ITQ tries to find the optimized rotation of PCA projection data. Therefore, ITQ is expected to preserve the data locality structure in a shorter binary code than LSH.

Through f, we embed each data point x_i to a c-bit binary code: $h_{x_i} = f(x_i) \in \{0, 1\}^c$. We then construct an inverted table by matching the cluster id of x_i and hash value of y_i. Fig. 1 gives an example to illustrate the inverted table construction process. A pair of data id x_i and cluster id c_i represents that the data point is assigned to the cluster. The hash code h_{x_i} of the data point is generated by f. The inverted table is constructed by using the hash codes as the index keys to associate with the cluster ids.

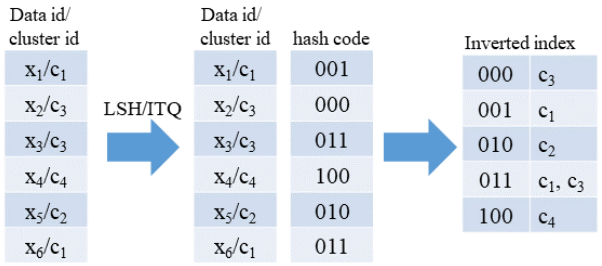

Figure 1. Inverted index table construction.

Given a query q, it is hashed by $h_q = f(q)$ to retrieve the cluster ids as the initial seeds. For example, in Fig. 1, suppose that hash value of $h_q = 011$, then the initial seeds for q are c_1 and c_3. Then we adopt the enhanced hill-climbing algorithm [8] that traverses kNN graph to refine the closest clusters iteratively. Finally, the data points belonging to the closest clusters are regarded as candidates, which are further examined their distances with q to output the final NNs.

4 Experiment

In this section, we conduct experiments to evaluate the proposed method in terms of recall and computation cost.

4.1 Dataset

We experimented on SIFT1M and GIST1M datasets of BIGANN [9] that contain one million SIFT and GIST vectors together with 10000 and 1000 query vectors, respectively. Each query provides 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. Afterwards, we used k-means clustering to generate clusters. In this study, we evaluate the performance under different numbers of clusters $M \in \{256, 1024, 4096, 16384\}$.

Table 1. Summary of SIFT1M and GIST1M datasets

Datasets	SIFT1M	GIST1M
# data dimensions	128	960
# data	1,000,000	1,000,000
# queries	10,000	1,000
# ground truth per query	100	100

4.2 Implementation

We implemented several configurations for the hill-climbing algorithm:

- **Exhaustive search**. Euclidean distances between a given query and cluster centroids are calculated to select clusters in an exhaustive way. The hill-climbing algorithm is not applied here.
- **Random seed hill-climbing**. Initial seeds for hill-climbing are generated randomly.

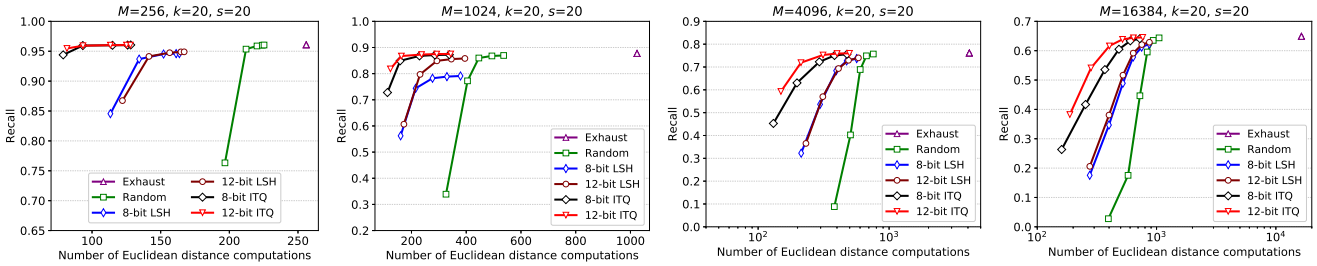

Figure 2. Recall in SIFT1M, where k and s are fixed at 20.

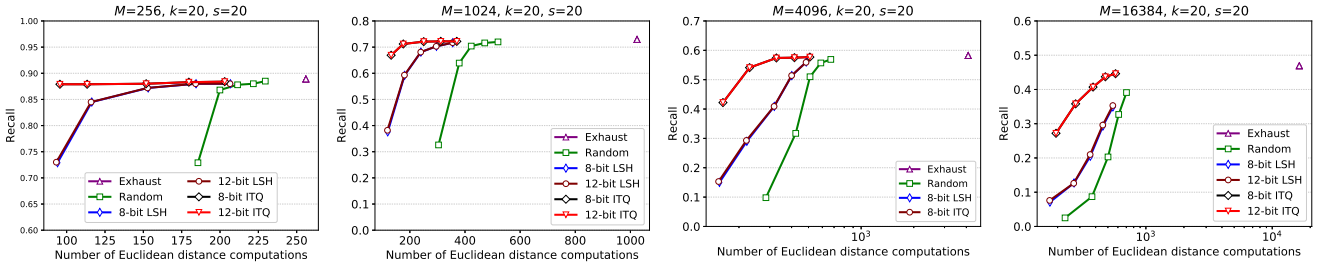

Figure 3. Recall in GIST1M, where k and s are fixed at 20.

- **8-bit LSH** and **12-bit LSH seed hill-climbing**. LSH [2] is used to transform data points into 8 and 12-bit hash codes.

- **8-bit ITQ** and **12-bit ITQ seed hill-climbing**. ITQ [3] is used to transform data points into 8 and 12-bit hash codes.

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

4.3 Result

We use the recall rate to measure the correctness in NN search. Let $Q = \{q_1, q_2, ..., q_T\}$ be a set of T queries, and $G = \{g_1, g_2, ..., g_T\}$ be the ground truth, where g_t is the first NN of ground truth for q_t. Recall is defined as:

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

where R_t is the retrieved set in response to q_t. In addition, we count the number of Euclidean distances calculated between query and cluster centroids to reflect the computation cost for each configuration.

Figs. 2 and 3 show the results under different M in SIFT1M and GIST1M, respectively, where k and s are both 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, say, random, LSH, and ITQ, all 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 less computations. Clearly in a larger number of clusters, more iterations are required to get converged. Both LSH and ITQ methods outperform the random method. Particularly, ITQ leverages the data distribution to generate seeds and thus yields the closest recall to the upper bound and spends the least computation cost. In addition, using more bits in LSH or ITQ generally increases the accuracy and computation cost. However, the influence is insignificant. Thus, the 8-bit hash code is considered sufficient to generate high quality seeds for hill-climbing.

Figs. 4 and 5 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, with the price of more space required to store larger kNN graph.

5 Conclusion

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

Acknowledgment

This work was supported by the Ministry of Science and Technology, Taiwan, under grants MOST 106-2221-E-415-019-MY3.

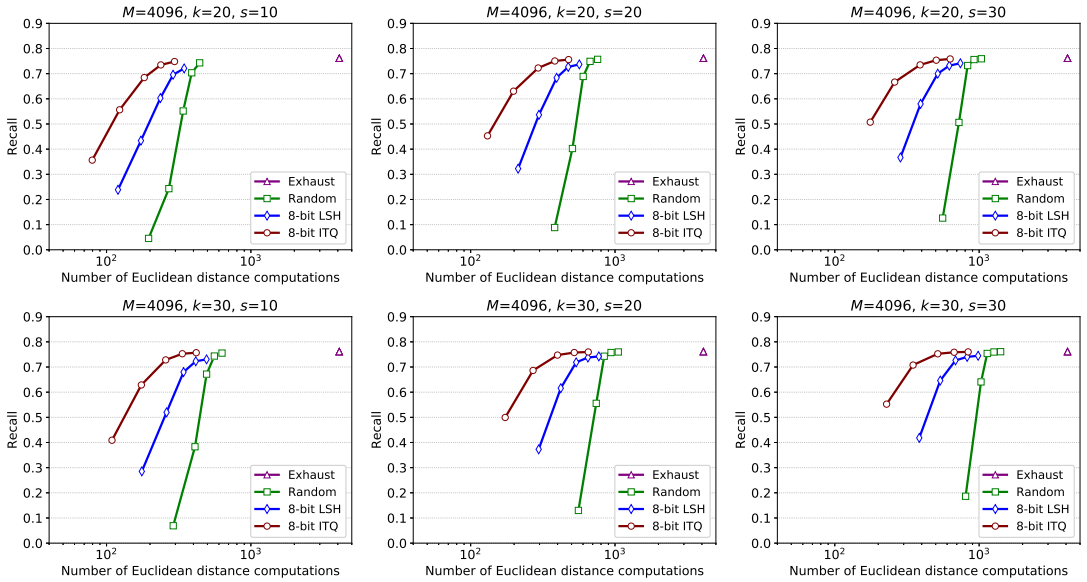

Figure 4. Recall in SIFT1M, where M is fixed at 4096.

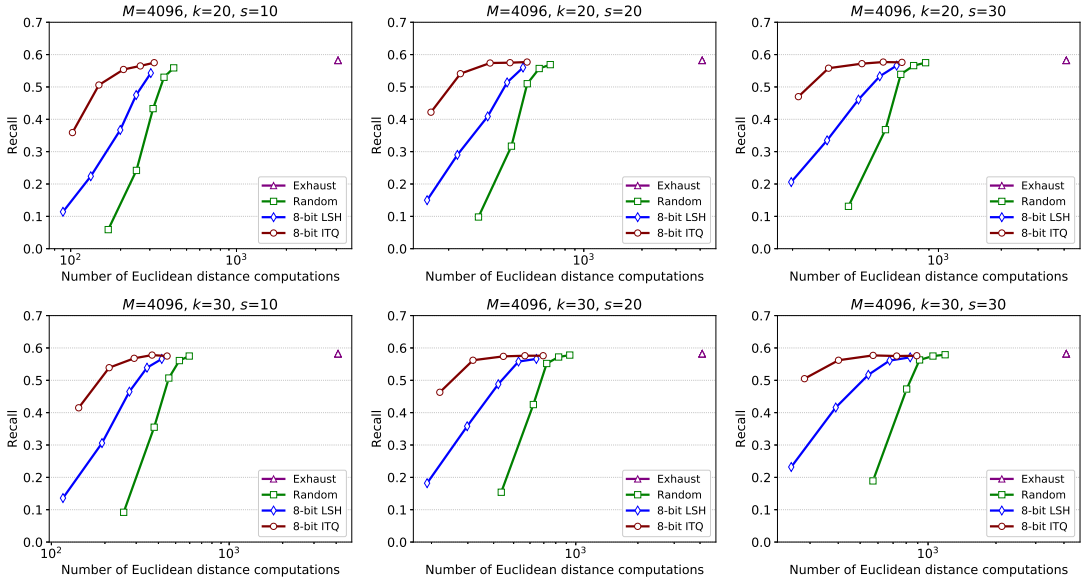

Figure 5. Recall in GIST1M, where M is fixed at 4096.

References

[1] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph," in *Proceedings of International Joint Conference on Artificial Intelligence*, vol. 22, no. 1, 2011, p. 1312.

[2] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the annual symposium on Computational geometry*. ACM, 2004, pp. 253–262.

[3] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2916–2929, 2013.

[4] J. Wang, J. Wang, G. Zeng, R. Gan, S. Li, and B. Guo, "Fast neighborhood graph search using cartesian concatenation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2128–2135.

[5] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 11, pp. 2227–2240, 2014.

[6] C. Fu and D. Cai, "Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph," *arXiv preprint arXiv:1609.07228*, 2016.

[7] Y.-M. Zhang, K. Huang, G. Geng, and C.-L. Liu, "Fast knn graph construction with locality sensitive hashing," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2013, pp. 660–674.

[8] W.-L. Zhao, J. Yang, and C.-H. Deng, "Scalable nearest neighbor search based on knn graph," *arXiv preprint arXiv:1701.08475*, 2017.

[9] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: re-rank with source coding," in *Acoustics, Speech and Signal Processing, IEEE International Conference on*, 2011, pp. 861–864.