Approximate *k*-NN Graph Construction: A Generic Online Approach

Wan-Lei Zhao, Hui Wang, and Chong-Wah Ngo

***Abstract*—Nearest neighbor search and *k*-nearest neighbor graph construction are two fundamental issues that arise from many disciplines such as multimedia information retrieval, data-mining, and machine learning. They become more and more imminent given the big data emerge in various fields in recent years. In this paper, a simple but effective solution both for approximate *k*-nearest neighbor search and approximate *k*-nearest neighbor graph construction is presented. These two issues are addressed jointly in our solution. On one hand, the approximate *k*-nearest neighbor graph construction is treated as a search task. Each sample along with its *k*-nearest neighbors is joined into the *k*-nearest neighbor graph by performing the nearest neighbor search sequentially on the graph under construction. On the other hand, the built *k*-nearest neighbor graph is used to support *k*-nearest neighbor search. Since the graph is built online, the dynamic update on the graph, which is not possible for most of the existing solutions, is supported. This solution is feasible for various distance measures. Its effectiveness both as *k*-nearest neighbor construction and *k*-nearest neighbor search approaches is verified across different types of data in different scales, various dimensions, and under different metrics.**

***Index Terms*—*k*-nearest neighbor graph, nearest neighbor search, high-dimensional, NN-descent.**

I. Introduction

GIVEN a dataset S, the k-NN graph refers to the structure that keeps top-k nearest neighbors for each sample in the dataset. It is the key data structure in the manifold learning [1]–[3], computer vision, machine learning, multimedia information retrieval [4], and video annotation [5]. Due to the fundamental role it plays, it has been studied for several decades. Basically, given a metric, the construction of k-NN graph is to find the top-k nearest neighbors for each data sample. When it is built in brute-force way, the time complexity is $O(d{\cdot}n^2)$, where d is the dimension and n is the size of dataset. Due to the prevalence of big data issues in various contexts, both d and n could be very large.

Despite numerous progress has been made in recent years, the major issues latent in the approximate k-NN graph construction still remain challenging. First of all, many existing approaches perform well only on low-dimensional data. The scale of data they are assumed to cope with is usually less than one million. Moreover, most of approaches are designed under specific metric i.e., l_2-norm. Only recent few works [4], [6], [7] aim to address this issue in the generic metric spaces. Thanks to the introduction of NN-Descent in [4], the construction time complexity has been reduced from $O(n^{1.94})$ [6] to $O(n^{1.14})$ for data with low dimensionality (e.g., 5) [4].

Besides the aforementioned major issues, many existing approaches still face another potential problem. Namely, most of the approaches are designed to build approximate k-NN graph for a fixed dataset. In practice, it is not unusual that the dataset changes from time to time. This is particularly the case for large-scale multimedia search tasks. For example, the photos and videos in Flickr grow daily. Given k-NN graph is adopted to support the content-based search and browse. The new items should be searchable in the next minute after being uploaded. Similar scenario is expected for e-shopping. The new products should be ready for retrieval and browsing by content right after being put on sale in the website.

In these scenarios, one would expect the k-NN graph that works behind should be updated dynamically. Unfortunately, for most of the existing approaches [4], [8]–[10], the dataset is assumed to be fixed. Any update on the dataset invokes a complete reconstruction on the k-NN graph. As a consequence, the aggregated costs are very high even the dataset is in small-scale. It is more convenient if it is allowed to simply insert/remove the samples into/from the existing k-NN graph. Nevertheless, it is complicated to update the k-NN graph with the support of conventional indexing structure such as locality-sensitive hashing (LSH) [11], R-Tree [12] or k-d tree [13].

Another problem that is closely related to approximate k-NN graph construction is the nearest neighbor search (NN search), which also arises from a wide range of applications. The nearest neighbor search problem is defined as follows. Given a query vector ($q \in R^d$), and n candidates in S that are under the same dimensionality. It is required to return the sample(s) that are the closest to the query according to a given metric $m(\cdot,\cdot)$.

In this paper, a generic approximate k-NN graph construction approach is presented. The issues of k-NN graph construction and NN search are addressed under a unified framework. The

Manuscript received September 17, 2020; revised February 15, 2021; accepted April 13, 2021. Date of publication April 19, 2021; date of current version April 6, 2022. This work was supported in part by National Natural Science Foundation of China under Grants 61572408 and 61972326, and in part by Xiamen University under Grant 20720180074. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Yongdong Zhang. *(Corresponding author: Wan-Lei Zhao).*

Wan-Lei Zhao and Hui Wang are with the Department of Computer Science and Technology, Xiamen University Xiamen 361005, China (e-mail: wlzhao@xmu.edu.cn; hwang2019@stu.xmu.edu.cn).

Chong-Wah Ngo is with the School of Computing and Information Systems, Singapore Management University, Singapore, Singapore (e-mail: cscwngo@cityu.edu.hk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TMM.2021.3073811.

Digital Object Identifier 10.1109/TMM.2021.3073811

graph construction problem is treated as a *k*-NN search task. The approximate *k*-NN graph is incrementally built by invoking each sample to query against the graph under construction. After one round of *k*-NN search, the query sample is joined into the graph along with the discovered top-*k* nearest neighbors. The *k*-NN lists of samples (already in the graph) that are visited during the search are accordingly updated. The NN search basically follows the hill-climbing strategy [14]. In order to achieve high performance in terms of both efficiency and quality, two major innovations are proposed.

- Restricted recursive neighborhood propagation (RRNP) is proposed to introduce the newly coming sample to its most likely neighbors, which enhances the quality of approximate *k*-NN graph considerably.
- In order to boost the search performance, a lazy graph diversification (LGD) scheme is proposed. It helps to avoid unnecessary distance computations during the hill-climbing while involving no additional computations.

The advantages of this approach are several folds. Firstly, the online construction avoids repetitive distance computations that most of the current approximate *k*-NN graph construction approaches suffer from. This makes it more efficient than the classic NN-Descent algorithm [4]. Secondly, the online graph construction is particularly suitable for the scenario that the dataset is dynamically changing. Moreover, with the support of *k*-NN graph, efficient NN search is achievable as demonstrated in [15], [16]. As a result, two related issues have been jointly addressed in our solution. Compared to the state-of-the-art NN search approaches [8], [15], [16], it shows similar or even slightly higher search efficiency while maintaining an online *k*-NN graph. The advantage is that it allows the user to browse over hyperlinks between closely related contents (i.e., *k* nearest neighbors). Furthermore, our approach has no specification on the distance measure, it is therefore a generic solution, which is confirmed in our experiments.

The remainder of this paper is organized as follows. In Section II, a brief review of the research works on the approximate *k*-NN graph construction and approximate *k*-NN search is presented. Section III presents an NN search algorithm upon which the approximate *k*-NN graph construction approach is built. Section IV presents an online approximate *k*-NN graph construction approach and the enhancement schemes. A more efficient NN search approach based on the online graph is presented in Section V. The experimental studies about the effectiveness of proposed *k*-NN graph construction and NN search are presented in Section VI. Section VII concludes the paper.

II. Related Works

A. k-*NN Search*

The early study about the *k*-NN search issue could be traced back to the *1970 s* when the need for NN search on the file system arises. In those days, the data to be processed are in very low dimensions, typically 1D. This problem is well-addressed by B-Tree [17] and its variant B^+-tree [17], based on which the NN search time complexity could be as low as $O(log(n))$. B-tree is not naturally extensible to more than 1D case. More sophisticated indexing structures were designed to handle NN search in multi-dimensional data. Representative structures are k-d-tree [13], R-tree [12] and R*-tree [18]. For k-d tree, pivot vector is selected each time to split the dataset evenly into two. By applying this bisecting repeatedly, the hyper-space is partitioned into embedded hierarchical subspaces. The NN search is performed by traversing over one or several branches to probe the nearest neighbors. Unlike B-tree in 1D case, the partition scheme does not exclude the possibility that nearest neighbor resides outside of these candidate subspaces. Therefore, extensive probing over a large number of branches in the tree becomes inevitable. For this reason, NN search with k-d tree and the like could be very slow. Recent indexing structure FLANN [19], [20] partitions space with hierarchical *k*-means and multiple k-d trees. Although efficient, sub-optimal results are observed.

For all the aforementioned tree partitioning methods, another major disadvantage lies in their heavy demand in memory. On the one hand, in order to support fast comparison, all the candidate vectors are loaded into the memory. On the other hand, the tree nodes that are used for indexing also take up considerable amount of extra memory. Overall, the memory consumption is usually several times bigger than the size of reference set.

In order to reduce the memory complexity, quantization based approaches [21]–[25] are proposed to compress the reference vectors [21], [26]. In recent works [27], [28], the performance of quantization based approaches is boosted due to the better indexing structure and the efficient computation on GPU. For all the quantization based methods, they share two things in common. Firstly, the candidate vectors are all compressed via vector (or sub-vector) quantization. Secondly, NN search is conducted between the query and the compressed candidate vectors. The distance between query and candidates is approximated by the distance between query and vocabulary words that are used for quantization. Due to the heavy compression on the reference vectors, high search quality is hardly achievable. Furthermore, these types of approaches are only suitable for metric spaces of l_p-norm.

Apart from the above approaches, several attempts have been made to apply LSH [11], [29] on NN search. In general, there are two steps involved in the search stage. Namely, *step 1* collects the candidates that share the same or similar hash keys as the query. *Step 2* performs an exhaustive comparison between the query and all these selected candidates to find out the nearest neighbor. In recent work, hashing by scalar quantization is also proposed for large-scale image search [30]. Whereas computational cost remains high if one expects high search quality. Additionally, the design of hash functions that are feasible for various metrics is non-trivial.

Recently, the graph-based approaches have been extensively explored [7], [8], [14], [31]–[33]. Although they are different from each other in details, all of them are built upon a hill-climbing procedure. The search [14] starts from a group of random seeds (random locations in the vector space). It traverses iteratively over an approximate *k*-NN graph or a relative *k*-NN graph (built-in advance) by the best-first search. Guided by the NN graph, the search procedure ascends closer to the true nearest neighbor in each round until no better candidates

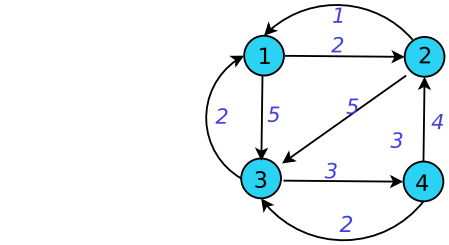
could be found. Approaches in [8], [14], [16], [32], [33] follow similar search procedure. The major difference between them lies in the graph used to support the search. According to recent study [15], these graph-based approaches demonstrate superior performance over other types of approaches across a variety of types of data.

B. Approximate k-NN Graph Construction

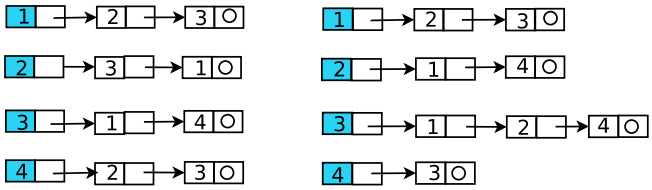
The approaches for approximate k-NN graph construction can be roughly grouped into two categories. Approaches such as [10], [34] basically follow the divide-and-conquer strategy. On the first step, samples are partitioned into a number of small subsets. Exhaustive comparisons are carried out within each subset. The closeness relations (*viz.*, edges in the k-NN graph) between any two samples in one subset are established. In the second step, these closeness relations are collected to build the k-NN graph. To enhance the performance, the first step is repeated several times with different partitions. The produced closeness relations are used to update the k-NN graph. Since it is hard to design a partition scheme that is feasible for various generic spaces, they are generally only effective in l_2-space. Another category of approximate k-NN graph construction, typically NN-Descent [4] avoids such disadvantages. The graph construction in NN-Descent starts from a random k-NN graph. Based on the principle "neighbor's neighbor is likely to be the neighbor," the k-NN graph evolves by invoking comparison between samples in each sample's neighborhood. Better closeness relations that are produced in the comparison are used to update the neighborhood of one sample. This approach turns out to be generic and efficient. Essentially, it can be viewed as performing hill-climbing batchfully [4]. Recently, the mixture scheme derived from the above approaches is also seen in the literature [8].

It is worth noting that approaches proposed in [7], [15], [16] are not approximate k-NN graph construction algorithms. The graphs are built primarily for k-nearest neighbor search. In these approaches, the samples which should be in the k-NN list of one sample are deliberately omitted for comparison efficiency. While the links to the remote neighbors are maintained [16]. As a consequence, graphs constructed by these approaches are not k-NN graph in the real sense. Such kind of graphs are hardly supportive for tasks beyond k-NN search. Navigable small-world graph (NSW) [32] is primarily designed to support fast NN search, it could be viewed as an online graph construction approach. However, it is not suitable for approximate k-NN graph construction for its poor search strategy, which leads to low graph quality.

In most of the aforementioned approaches, one potential issue is that the construction procedure has to keep records on the comparisons that have been made between sample pairs to avoid repetitive comparisons. However, its space complexity could be as high as $O(n^2)$. Otherwise the repetitive comparisons are inevitable even by adopting specific sampling schemes [4]. In this paper, the approximate k-NN graph construction and k-NN search are addressed jointly. The approximate k-NN graph construction is undertaken by invoking each sample as a query to query against the approximate k-NN graph that is under construction. Since the query is new to the graph under construction each time, two samples in the dataset are compared at most once. The repetitive comparisons are avoided.



(a) a graph with *4* vertices



(b) *2*-NN graph G for (a) (c) reverse *2*-NN graph $\overline{G}$ for (a)

Fig. 1. An illustration of k-NN graph G and its reverse k-NN graph $\overline{G}$. In the illustration, k is set to *2*.

III. NN Search on the K-NN Graph

Before we introduce our graph construction approach, the NN search, on which the construction approach is based, is presented. Our search procedure is largely similar as the procedure proposed in HNSW [16]. Whereas unlike HNSW, our search procedure is undertaken on a flat k-NN graph instead of a hierarchical relative neighborhood graph. Compared to the single-layer search in HNSW, a few modifications are further introduced. In the flat k-NN graph, both the k nearest neighbors of one sample and its reverse neighbors are kept. In the following, the structure of this graph is discussed before we present the search procedure.

Given k-NN graph G, $G[i]$ returns a k-NN list of sample i. Accordingly, $\overline{G}$ is the reverse k-NN graph of G, which is nothing more than a re-organization of graph G. $\overline{G}[i]$ keeps ID of samples that sample i appears in their k-NN lists. Noticed that the size of $\overline{G}[i]$ is not necessarily k and there would be overlappings between $\overline{G}[i]$ and $G[i]$. An illustration of graphs G and $\overline{G}$ are seen in Fig. 1. In our implementation, the union of $G[i]$ and $\overline{G}[i]$ are kept in a dynamic array (instead of linked list). The first k elements are the k nearest neighbors (namely $G[i]$). The remaining elements are the reverse neighbors of sample i that are outside the coverage of k nearest neighbors. For presentation clarity, these samples in the neighborhood are still referred to as $G[i]$ and $\overline{G}[i]$. With the support of k-NN graph G and its reverse graph $\overline{G}$, the NN search algorithm is presented in Alg. 1.

Alg. 1 in general is a hill-climbing procedure [14] with multiple starting seeds. The query q is firstly compared to p seed samples. The compared samples are kept in two priority queues Q and R. Q basically maintains the top-k nearest neighbors of

Algorithm 1: NN Search on Graph (NNSearch($\cdot$))

Data: q: query; G: k-NN Graph; $\overline{G}$: reverse k-NN Graph; $S_{n\times d}$: reference set; k: num. of NN; p: num. of seeds
Result: Q: k-NN list of q; V: Samples visited during the search; D: Distances between sample q and the samples visited during the search

1 $Q \leftarrow \emptyset$; D[1$\cdots$n] $\leftarrow \infty$;
2 Flag[1$\cdots$n]$\leftarrow$ 0; R[1$\cdots$p] $\leftarrow$ p random seeds;
3 **for** *each* $r \in R$ **do**
4 Flag[r] $\leftarrow$ 1; D[r] $\leftarrow m(r,q)$;
5 $f \leftarrow$ get furthest sample to q from Q ;
6 **if** $m(r,q) < m(f,q)$ *or* $|Q| < k$ *or f is NULL* **then**
7 $Q \leftarrow Q \cup r$;
8 $R \leftarrow R \cup r$;
9 **end**
10 **end**
11 **while** $|R| > 0$ **do**
12 $r \leftarrow$ pop nearest sample to q from R;
13 $f \leftarrow$ get furthest sample to q from Q ;
14 **if** $m(r,q) > m(f,q)$ **then**
15 break;
16 **end**
17 **for** *each* $e \in G[r] \cup \overline{G}[r]$ **do**
18 **if** *Flag[e] == 0* **then**
19 Flag[e] $\leftarrow$ 1; D[e] $\leftarrow m(e,q)$;
20 $f \leftarrow$ get furthest sample to q from Q;
21 **if** $m(e,q) < m(f,q)$ *or* $|Q| < k$ **then**
22 $Q \leftarrow Q \cup e$;
23 $R \leftarrow R \cup e$;
24 **end**
25 **end**
26 **end**
27 **end**
28 Collect the visited samples marked in *Flag* to V;

the query q. The retrieved neighbors are sorted in ascending order according to their distance to the query.[1] As a closer sample is joined into Q, the rear sample in Q will be swapped out. Unlike Q, the size of R is not fixed. In each iteration (*Line 11 – 27*), the algorithm visits the neighborhood of the closest sample $r \in R$ to the query q. Q and R are updated accordingly with new close samples to the query. The iteration continues until R is empty or Q cannot be updated. In the comparison, the distance function $m(\cdot,\cdot)$ could be any metric defined on the input dataset. It is clear to see this is a generic search algorithm. The major differences between Alg. 1 and the single-layer NN search algorithm from HNSW [16] are in two aspects. Firstly, it starts from multiple random seeds, which leads to more stable performance over HNSW. Moreover, the visited samples and the corresponding distances to the query are kept. These distances will be used to assist the online *k*-NN graph diversification later.

[1]Without the loss of generality, it is assumed that the smaller of the distance the closer of two samples across the paper.

Algorithm 2: Online Approximate k-NN Graph Constr. (OLGraph)

Data: $S_{n\times d}$: dataset; k: num. of NN; p: num. of seeds
Result: G: k-NN Graph

1 *Extract* a small subset I from S;
2 *Initialize* G and $\overline{G}$ with I;
3 **for** *each* $q \in S - I$ **do**
4 $Q, V, D \leftarrow$NNSearch$(q, G, \overline{G}, S_{n\times d}, k, p)$;
5 /*V is the set of samples visited during the search*/
6 /*D keeps distances btw. q and the visited samples*/
7 **for** *each* $r \in V$ **do**
8 InsertG$(r, q, D[r], G)$;
9 InsertG$(q, r, D[r], \overline{G})$;
10 **end**
11 **end**

IV. Online Approximate *k*-NN Graph Construction

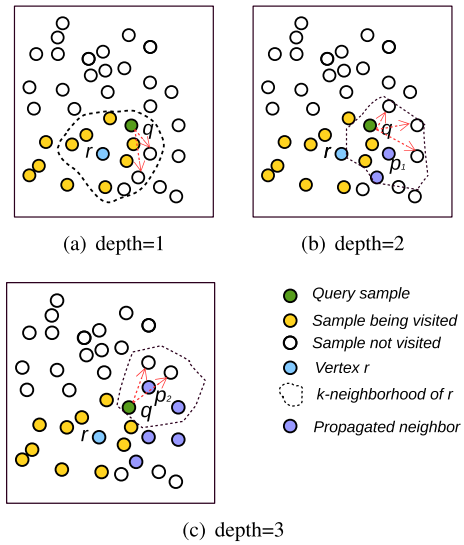
The prerequisites for the NN search algorithm in Alg. 1 are the *k*-NN graph G and its reverse *k*-NN graph $\overline{G}$. In this section, we are going to show how an approximate *k*-NN graph and its reverse graph are built based on NN search algorithm itself. Additionally, a strategy called *restricted recursive neighborhood propagation* is proposed to enhance the quality of approximate *k*-NN graph. Moreover, an online graph diversification scheme is proposed. Compared to the approaches in [7], [15], it involves no additional distance computations while leading to more efficient NN search than Alg. 1.

A. Approximate k-NN Graph Construction by Search

In Alg. 1, the search starts from several random locations of the reference set and moves along several trails. The search moves towards the closer neighborhood of a query in each iteration. In the ideal case, top-*k* nearest neighbors will be discovered. On the one hand, the top-*k* nearest neighbors of this query are known after the search. On the other hand, some samples in the reference set are introduced with a new neighbor (namely the query). As a result, the *k*-NN graph could be augmented to include this query sample.

Motivated by this observation, the online *k*-NN graph construction algorithm is conceived. First, an initial graph is built exhaustively from a small subset of S. The size of S is fixed to *64* in this paper. After that, each of the remaining samples is treated as a query to query against the *k*-NN graph following the flow of Alg. 1. The *k*-NN list of a query sample is joined into the graph after each search. The *k*-NN lists of samples which have been visited during the search are accordingly updated. The general procedure of the construction algorithm is summarized in Alg. 2.

In Alg. 2, the procedure of approximate *k*-NN graph construction is basically a repetitive calling of the NN search algorithm and graph update function. Function *InsertG*($\cdot$) is responsible for inserting an edge into *k*-NN list of *r* in graphs G and $\overline{G}$. The major operations inside *InsertG*($\cdot$) involve the update of *k*-NN list and the reverse *k*-NN list of *r*. A sample in the rear of a *k*-NN



(a) depth=1 (b) depth=2

(c) depth=3

Fig. 2. The illustration of *restricted recursive neighborhood propagation*. The propagation starts from unvisited samples in r's neighborhood and expands deeper to the neighbors of r's neighbor. In the figure, p_1 is r's neighbor, p_2 is p_1's neighbor. Neither of them are encountered during the search.

list is deleted if a closer sample is joined in. Distances of k-NNs are kept to allow the list to be sorted all the time.

Although the size of input dataset is fixed in Alg. 2, it is apparently feasible for an open set, for which new samples are allowed to join in from time to time. As will be revealed in the experiments (Section VI), Alg. 2 already performs very well. In the following, two novel schemes are presented to further boost the performance in the graph construction and NN search respectively.

B. Restricted Recursive Neighborhood Propagation

In the last steps of Alg. 2, the query sample q is inserted to the neighborhood of each r as long as q is in their k-NN range. Noticed that only a few rs that are sufficiently close to q will be considered. In the neighborhood of r, it is possible that there are some samples that were not compared to q during the hill-climbing search (shown as empty circles inside the dashed polygon of Fig. 2(a)). Based on the principle that "neighbor's neighbor is likely to be the neighbor," these unvisited samples are likely to be close neighbors of sample q. Therefore, it is reasonable to insert q and these unvisited samples into the neighborhoods of each other. After this insertion, it is possible that q is introduced to meet with more unvisited neighbors. As a result, such kind of insertion could be undertaken recursively until no new unvisited neighbors are encountered. In addition, such kind of propagation is restricted to the close neighborhood of a sample. For example, as shown in Fig. 2(b), p_1's neighborhood is propagated only if $m(q, p_1)$ is smaller than the distance from p_1 to its k-th neighbor. This operation is called *restricted recursive neighborhood propagation* (RRNP). We find this operation further improves the quality of k-NN graph while only inducing minor computational overhead.

Algorithm 3: Online Approximate k-NN Graph Constr. with RRNP and LGD (OLGraph$^+$)

Data: $S_{n\times d}$: dataset; k: num. of NN; p: num. of seeds; dp: max. depth of propagation
Result: G: k-NN Graph with LGD information

```
1  Extract a small subset I from S;
2  Initialize G and Ḡ with I;
3  for each q ∈ S − I do
4      Q, V, D ←NNSearch(q, G, Ḡ, S_{n×d}, k, p);
5      Flag[1···n]← 0;
6      for each r ∈ V do
7          InsertG(r, q, G);
8          InsertG(q, r, Ḡ);
9          W ← ∅;
10         Push(r, depth(r), W); //depth of r is 0
11         while |W| > 0 do
12             s ← Pop(W);
13             f ← the k-th neighbor of s;
14             if depth(s) < dp && m(q, s) < m(s, f) then
15                 for each e ∈ G[s] ∪ Ḡ[s] do
16                     if e ∉ V && Flag[e] == 0 then
17                         Flag[e] ← 1;
18                         InsertG(e, q, G);
19                         InsertG(q, e, Ḡ);
20                         Push(e, depth(s) + 1, W);
21                     end
22                 end
23             end
24         end
25         ApplyLGD(G[r], D); //Apply LGD rules
26     end
27 end
```

The procedure of RRNP is illustrated in Fig. 2 and shown in Alg. 3 (*Line 9 – 24*). In Alg. 3, W is a working queue for the samples to be propagated. Function *Push*$(\cdot)$ inserts a new element at the end of the queue and function *Pop*$(\cdot)$ removes and returns the next element in the queue. Firstly, a sample r is pushed into W. After that, all neighbors of the sample(s) in W that meet the conditions will be pushed into the queue, and the k-NN graph is updated accordingly. This operation will be repeated until the queue is empty, which means that there is no sample available for propagation.

Although RRNP is conceptually similar to neighborhood propagation proposed in [35], they are essentially different in two aspects. Firstly, there is no priority in terms of propagation in RRNP. All the unvisited samples in one neighborhood are compared with the query in random order. Furthermore, such kind of propagation is restricted to the close neighborhood (within the radius of a sample's k-NN list). The neighborhood propagation proposed in [35] is more like a mini version of NN-Descent.

C. Lazy Graph Diversification

In Alg. 1, when expanding sample r, all the samples in the neighborhood of r will be compared to the query. According to recent studies [7], [15], [16], when samples in the neighborhood

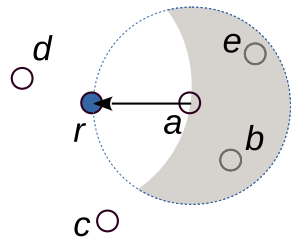


Fig. 3. An illustration of occlusion in 2D l_2-space happens in the neighborhood of sample r. Samples a, b, c, d and e are in the k-NN list of r. $m(r, b)$ is greater than $m(r, a)$, while $m(a, b)$ is smaller than $m(r, b)$, we say that b is occluded by a in r's neighborhood, while sample c and d are not occluded by a. Actually all the samples located in the moon shape shadow are occluded by a. Notice that the region that is occluded by a could be beyond this moon shape region.

of r are very close to each other, it is no need to compare to all of them during the expansion. The expansion on these close samples most likely guides the climbing process to the same local region. The phenomenon that samples in the k-NN list are closer to each other than they are to r is called as "occlusion" [7]. An illustration of occlusion is shown in Fig. 3. In the illustration, samples b and e are occluded by sample a. It is easy to see one sample can only be occluded by samples which are closer to r than that of it. According to [7], [15], [16], the hill-climbing will be more efficient when samples like b and e are not considered when expanding r.

In order to know whether samples in a k-NN list are occluded by each other, the pair-wise comparisons between samples in the k-NN list are required [7], [15], [16]. This is unfeasible for an online construction procedure (i.e., Alg. 2). First of all, k-NN lists are dynamically changing, pair-wise distances cannot be simply computed and kept for use all the way. Secondly, it is too costly to update the pair-wise occlusion relations as long as a new sample is joined in. It would induce a complete comparison between this new sample and the rest. Moreover, unlike HNSW the occluded samples cannot be simply removed from a k-NN list since our primary goal is to build an approximate k-NN graph as precise as possible.

In this paper, a novel scheme called *lazy graph diversification* (LGD) is proposed to identify the occlusions between samples during the online graph construction. Firstly, an occlusion factor λ is introduced as the attribute attached to each sample in a k-NN list. λs of all the samples in the list are initialized to *0* when the k-NN list of a new query is joined into the graph. Factor λ will be updated when another new sample is joined into this k-NN list at the later stages.

Let's consider a new sample q to be inserted into sample r's k-NN list. In order to update λs of r's neighbors, we should know the distances of all the neighbors to r and the distances between q and other neighbors in the list. The distances to r are already known. While the distances between the query and the rest neighbors are unknown. Instead of performing a costly comparison between q and the rest neighbors, we make use of distances that are recorded in variable D. Namely, the distances between q and all the visited samples during the NN search (Alg. 1). With the support of D, occlusion factor λ of all the samples in the k-NN list is updated with following three rules.

- **Rule 1**: λ is kept unchanged for samples ranked before q;
- **Rule 2**: λ of sample q is incremented by *1* if a sample ranked before q is closer to q than q to r;
- **Rule 3**: λ of a sample ranked after q is incremented by *1* if its distance to q is smaller than q to r.

The default distance of each sample to q is set to ∞. The λs of not-being-visited neighbors are not updated according to *Rule 1* and *Rule 3*. This is reasonable because the not-being-visited neighbors should be sufficiently far away from q, otherwise they are already being visited according to the principle "a neighbor of a neighbor is also likely to be a neighbor". Since we have all the possible distances (between q and samples in the graph) only after the hill-climbing converges, the operations of inserting q into k-NN list of r and updating factor λs in the list are postponed to the end of NN search. Fig. 4 illustrates a trail that is formed by the NN search. In the k-nearest neighborhood of r, the LGD operations are applied.

Our approach is different from [15], the graph diversification is undertaken in a lazy way in the sense no exhaustive comparison is involved within the k-NN list. This scheme is therefore called *lazy graph diversification* (LGD). The three rules used to calculate the occlusion factor are called as *LGD rules*. Our approach is also different from the way proposed in [7], [16], in which the occluded samples ($\lambda > 0$) are simply omitted. This is infeasible in our case as it deviates from the goal of k-NN graph construction. Alternatively, the occlusion factor λ works as an indicator of the degree of occlusion. If one sample's λ is above the average level $\overline{\lambda}$, it is viewed as being occluded. Such kinds of samples will not be visited during the fast NN search, which will be elaborated in Section V. Function *ApplyLGD(·)* in Alg. 3, *Line 25* is responsible to fulfill *LGD rules* as one sample is inserted.

D. Sample Removal From k-NN Graph

In practice, we should allow samples to be dropped out from the k-NN graph. A good use case is to maintain a k-NN graph for product photos for an e-shopping website, where old-fashioned products should be withdrawn from sale. The removal of samples dynamically from the k-NN graph is supported in our approach. If the graph is built by Alg. 2, the removal operation is as easy as deleting the sample from the k-NN lists of its reverse neighbors and releasing its own k-NN list. If the graph is built by Alg. 3, before the sample is deleted, the occlusion factors of the samples living in the same k-NN list have to be updated. Fortunately, not all the samples in the list should be considered. According to *LGD Rule 3*, only samples ranked after this sample should be considered. The update operations involve $k^2/2$ times distance computations on average. Given k is a small constant, the time cost is much lower than fulfilling a query on the graph.

Dynamic sample removal is not conveniently supported by other k-NN graph construction approaches [32], [32], [36] or other graph-based NN search indexing structures [15], [16]. In

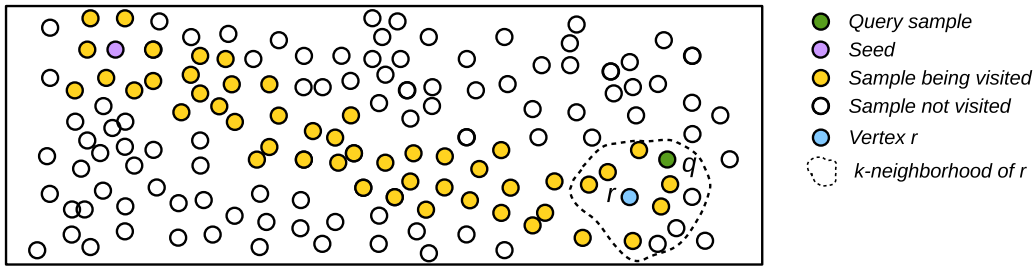


Fig. 4. A trail of hill-climbing procedure in 2D l_2-space. The hill-climbing starts from a single seed and converges when it reaches the neighborhood of the query. Query sample q is to be inserted into the k-NN list of r. The occlusion relations between q and the rest samples in r's neighborhood have to be updated. The distances from samples in the list to r are known. The distances from q to visited samples in r's neighborhood are also known. The distances from q to not-being-visited are ∞. Based on the *LGD rules*, the occlusion factor λs of samples in r's neighborhood could be updated.

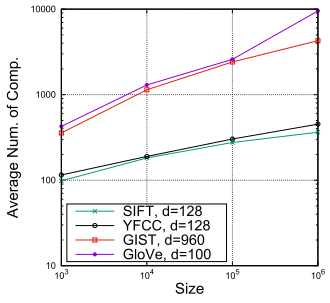


Fig. 5. The variations in the number of average comparisons required for insertion of one sample. The experiments are conducted with OLGraph$^+$ on four million-level datasets, The intrinsic data dimensions for SIFT, YFCC, GIST and GloVe are *16.3*, *23.4*, *38.1* and *39.5* respectively. The Recall@1 is maintained above *0.95* level.

HNSW [16], the sample removal operation may lead to the collapse of the indexing structure. While this issue is not even considered in [15].

E. Complexity and Optimality Analysis

For both OLGraph (Alg. 2) and OLGraph$^+$ (Alg. 3), the adjacency list structure is required to support the search and dynamic update. Memory for IDs, occlusion factors, and distances of k neighbors and reverse neighbors must be allocated. As a result, the upper bound of memory consumption of index is around $20{\cdot}k{\cdot}n$ bytes,[2] where n is the scale of the dataset. Since the structure is the union of neighbors and reverse neighbors, the real memory consumption will be much lower than this bound.

Essentially NN-Descent, OLGraph, and OLGraph$^+$ avoid exhaustive comparison via the routings supplied by the k-NN graph. The routing is effective when it guides the query quickly to its true neighbors. This is largely determined by the intrinsic data dimension [4], which varies from one dataset to another.

In order to investigate the time complexity of OLGraph$^+$,[3] an empirical study is conducted on four real-world datasets with varying data dimensions and scales. On each dataset, we investigate the average number of comparisons required to construct k-NN graphs in different data scales. As shown in Fig. 5, the number of average comparisons required for each dataset is at least one order of magnitude lower than the size of the dataset. This basically indicates that the time complexity of OLGraph$^+$ is in the range of $[n^{1.5}, n^{1.9}]$ when the data scale is on the million level. The time complexity of OLGraph$^+$ is lower on data with lower intrinsic dimensions, e.g. SIFT and YFCC. For the dataset GloVe, its time complexity is close to $O(n^{1.66})$ as its intrinsic dimension is as high as *39.5* [37]. However, compared to the exhaustive approach, OLGraph$^+$ is still very efficient in the sense it saves up *98%* comparisons in terms of its scanning rate, which will be illustrated in the later experiments. Compared to NN-Descent, OLGraph$^+$ avoids potential repetitive comparisons as each sample is new to the samples already in the graph. In addition, OLGraph$^+$ skips the comparison to the occluded samples in the neighborhood due to LGD. In general, it is more efficient.

In both OLGraph (Alg. 2) and OLGraph$^+$ (Alg. 3), the construction starts from a small-scale k-NN graph of *100%* quality. The search process appends a k-NN list of a new sample to the graph each time. Meanwhile, the k-NN lists of the already inserted samples will be possibly updated when the new sample happens to be in their neighborhoods. It is, therefore, a win-win situation for both graph construction and NN search. Effective search procedure returns high-quality k-NN list. While high-quality k-NN graph gives a good guidance for the hill-climbing process.

Besides the size of NN list k, there is another parameter involved in OLGraph and OLGraph$^+$. Namely, the number of seeds p. Usually, the size of NN list k should be no less than the intrinsic data dimension d^* [37], which is less than or equal to the data dimension d. The number of seeds is usually set to be no bigger than k. When d is very high (i.e., several hundred to thousand) and d^* is close to d, the construction process could be slow when k is set to be close to d. In such a situation, a trade-off has to be made between the quality of k-NN graph and the efficiency of the construction.

V. Fast NN-Search on the Diversified Graph

Under *LGD rules*, the samples in one k-NN list and the corresponding reverse k-NN list are considered as being occluded

[2]This is estimated with 64bits machine.
[3]OLGraph demonstrates similar trend.

when their λ is higher than $\overline{\lambda}$ of the *k*-NN list, where $\overline{\lambda}$ is the average occlusion factor of the list. This basically indicates these samples in the list are too close to other samples that they are no need to be considered during the hill-climbing. The factors of all the samples in one list are updated dynamically as long as samples are inserted/removed from the list. As the new samples are joined in the list, members that are previously occluded may become "visible". This is the essential difference between our approach and HNSW [16], in which occluded samples are removed permanently once it is identified.

Once the occlusion factor is available, the search algorithm (Alg. 1) is accordingly modified. When the query is compared to the neighbors in *r*'s *k*-NN list, we only consider the samples whose λ is no greater than the average occlusion factor $\overline{\lambda}$ of this list, where $\overline{\lambda}$ is the average occlusion factor of the *k*-NN list. The NN search on the diversified *k*-NN graph is largely similar as Alg. 1. In the modified NN search, a conditional judgment statement (like *Line 18* in Alg. 1) is added to check whether the occlusion factor λ of a sample is above $\overline{\lambda}$. Only the "visible" samples are joined in the comparison. Speed-up is expected as nearly *50%* of samples in one *k*-NN list are skipped during the search.

Although NN search on LGD graph is significantly more efficient than Alg. 1, it is not suitable to be adopted in the *k*-NN graph construction in Alg. 3. The close neighbors that are considered in the comparison in Alg. 1 will be ignored in this modified NN search. As a consequence, the samples should be joined in *k*-NN lists of each other are simply unvisited. For this reason, *NNSearch*$(\cdot)$ (Alg. 1) in Alg. 3 is recommended when one wants to add a sample to the graph. Alternatively, NN search with LGD check is recommended as one performs pure NN search. In the experiment section, we will show that the LGD strategy leads to considerable speed-up in the NN search. It becomes competitive in comparison to the state-of-the-art approaches.

To this end, one could imagine that there are two modes in our online *k*-NN graph framework, namely the construction mode and pure NN search mode. Under the pure NN search mode, *ApplyLGD(·)* is not called since no *k*-NN list update is invoked during the procedure. Under the construction mode, Alg. 3 is invoked.

The fast online *k*-NN graph construction and NN search have been integrated into one framework. Compared to the existing *k*-NN graph construction approaches [4], [32], [36], fast dynamic insertion, removal as well as NN search on a *k*-NN graph are all well supported. Compared to other graph-based NN indexing structures such as HNSW [16] and DPG [15], there is no offline construction stage. As a result, the cost of dynamically maintaining the indexing structure is much lower than either HNSW [16] or DPG [15]. Furthermore, compared to HNSW and DPG, a diversified *k*-NN graph and a *k*-NN graph are maintained simultaneously[4] in one structure. On the one hand, it guarantees fast NN search. On the other hand, it allows the user to browse over similar contents via the links between *k*-nearest neighbors.

[4]The diversified *k*-NN graph is actually a subset of *k*-NN graph.

TABLE I
SUMMARY ON DATASETS USED FOR EVALUATION

Name	n	d	# Qry	m$(\cdot,\cdot)$	Type
SIFT1M [22]	1×10^6	128	1×10^4	l_2	SIFT [40]
SIFT10M [22]	1×10^7	128	1×10^4	l_2	SIFT
GIST1M [41]	1×10^6	960	1×10^3	l_2	GIST [41]
GloVe1M [42]	1×10^6	100	1×10^3	*Cosine*	Text [42]
NUSW [43]	$22,660$	500	1×10^3	l_2	BoVW [44]
NUSW [43]	$22,660$	500	1×10^3	κ^2	BoVW
YFCC1M [45]	1×10^6	128	1×10^4	l_2	ResNet-50 [46]
Kosarak [47]	7.4×10^4	27983	1×10^3	*Jaccard*	Itemset

VI. EXPERIMENTS

In this section, the performance of the proposed algorithms is studied both as an approximate *k*-NN graph construction and a nearest neighbor search approach. In the evaluation, the performance is reported on popular evaluation datasets. The brief information about the datasets is summarized in Table I. In particular, the deep features for YFCC1M are extracted from the *2nd* last layer of ResNet-50. The features are further reduced to *128* dimension via PCA.

On the approximate *k*-NN graph construction task, existing approaches NN-Descent [4] and NSW [32] are considered in the performance comparison, both of which are feasible for various distance metrics. On the nearest neighbor search task, the performance of the proposed search approach is studied in comparison to the representative approaches of different categories. Namely, they are graph-based approaches such as NN-Descent [4], DPG [15] and HNSW [16]. The representative locality-sensitive hash approach SRS [38] is considered. For quantization based approach, product quantizer (PQ) [22] and double-bit quantization (DBQ) [30] are incorporated in the comparison. FLANN [19] and Annoy [39] are selected as the representative tree partitioning approaches. Both of them are popular NN search libraries in the literature.

A. Evaluation Protocol

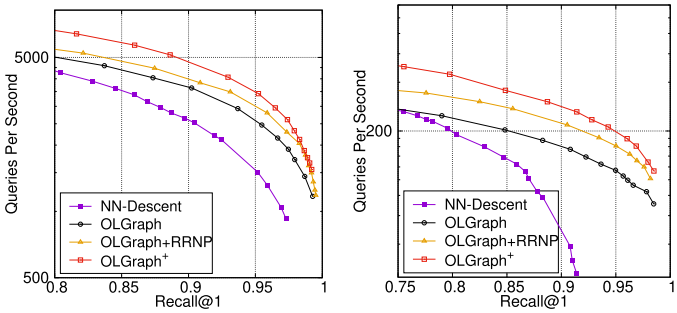
Eight real-world datasets are adopted to evaluate the performance of both *k*-NN graph construction and nearest neighbor search. These datasets are derived from real-world images, text data, or itemset. The top-*1* (*recall@1*) and top-*10* (*recall@10*) recalls on each dataset are studied under different metrics such as l_2, *Cosine*, *Jaccard* and κ^2. Given function $R(i,k)$ returns the number of truth-positive neighbors at top-*k* NN list of sample i, the recall at top-k on the whole set is given as

$$recall@k = \frac{\sum_{i=1}^{n} R(i,k)}{n\times k}. \tag{1}$$

Besides *k*-NN graph quality, the construction cost is also studied by measuring the scanning rate [4] and time cost of each approach. Given C is the total number of distance computations in the construction, the scanning rate is defined as

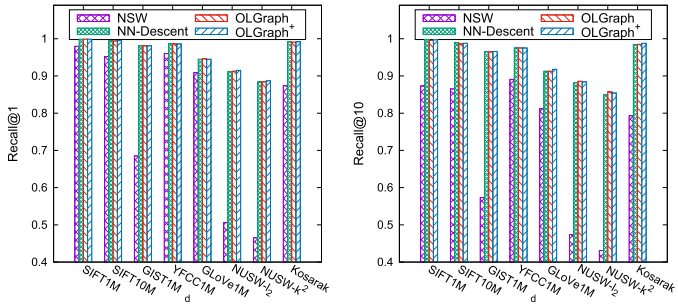
$$c = \frac{C}{n\times(n-1)/2}. \tag{2}$$

For each dataset, another *1000* or *10 000* queries of the same data type are prepared. The NN search quality is measured by the top-*1* recall for the first nearest neighbor. The search quality



(a) SIFT1M (b) GIST1M

Fig. 6. The NN search performance on SIFT and GIST based on the graphs built by NN-Descent, OLGraph, OLGraph+RRNP and OLGraph$^+$ respectively. k is fixed to *40* for all the graphs.



(a) Recall@1 (b) Recall@10

Fig. 7. Top-*1* and Top-*10* recall of k-NN graphs produced by NSW, NN-Descent, OLGraph and OLGraph$^+$ on eight datasets.

is reported along with the number of queries per second. All the codes of different approaches considered in this study are compiled by g++ *5.4*. In order to make our study to be fair, we disable all the multi-threads, SIMD, and pre-fetching instructions in the codes since not all the original codes are optimized with these techniques. All the experiments are executed on a PC with *3.6*GHz CPU and *32*G memory setup.

B. Ablation Study

In this section, the effectiveness of the proposed online graph diversification scheme (LGD) and graph quality enhancement scheme (RRNP) is studied. As graph construction and NN search are closely related in our approach, we study the NN search performance improvement when each scheme is added to OLGraph. Namely, the NN search performance is studied when the search is conducted on the graphs built by OLGraph (Alg. 2), OLGraph+RRNP, OLGraph$^+$ (Alg. 3) respectively. In OLGraph$^+$, both LGD and RRNP are integrated. NN search based on k-NN graph from NN-Descent is treated as the comparison baseline. The results are shown in Fig. 6.

As seen from Fig. 6, due to the incorporation of the reverse k-NN list, OLGraph outperforms NN-Descent considerably on both datasets. The performance of OLGraph is further boosted as the neighborhood propagation scheme RRNP enhances the graph quality. The best performance is achieved when the graph diversification scheme LGD is integrated. According to our observation, LGD does not enhance the graph quality. Instead, it helps to skip unnecessary comparisons, which leads to the higher efficiency for search, and in turn for graph construction. As both LGD and RRNP are effective, they are integrated into OLGraph$^+$ in the following experiment.

C. Approximate k-NN Graph Construction

The performance of approximate k-NN graph construction is studied when the k-NN search algorithm is employed as a graph construction approach. Eight real-world datasets are adopted in the evaluation. Among them, NUSW is tested under both l_2 and κ^2 distance measures, and Kosarak uses $Jaccard$ distance measure. The performance of OLGraph (Alg. 2) and OLGraph$^+$ (Alg. 3) is compared to NN-Descent [4] and NSW [32]. NN-Descent is recognized as the most effective approximate k-NN graph construction approach that works in the generic metric spaces, and NSW is an online approach to construct a navigable small-world graph for k-nearest neighbor search. The k-NN graph can be derived from the NSW graph by only keeping the first k nearest neighbors. In the test, the parameter k in NN-Descent is fixed to *40* for all the datasets, and the parameters of OLGraph and OLGraph$^+$ are tuned to reach similar recalls as NN-Descent. Since it is difficult for NSW to reach similar k-NN graph quality as the other three approaches, its performance is reported on the level that its time cost is similar to other approaches. The scanning rates and time consumption of all four approaches are reported in Table. II. While the top-*1* and top-*10* recalls of all the approaches are shown in Fig. 7.

As seen from the table, in most of the cases, the scanning rates from OLGraph and OLGraph$^+$ are much lower than that of NN-Descent when their graph quality is maintained on the similar level. This basically indicates much less distance computations are involved with OLGraph and OLGraph$^+$. Compared to NN-Descent, OLGraph and OLGraph$^+$ avoid repetitive distance computations between any sample pairs. Since the distance computation is the most computationally intensive operation in all approaches, OLGraph and OLGraph$^+$ are expected to be much faster. However, the CPU cache structure is more friendly to NN-Descent since its distance computations are taken place in a local at each moment. For this reason, the computation costs from OLGraph are not considerably lower than NN-Descent as is shown in Table. II. In contrast, OLGraph$^+$ shows considerably lower time costs than NN-Descent due to its very low scanning rate. In particular, it has a greater advantage on high-dimensional datasets like GIST1M and NUSW.

As shown in Fig. 7, k-NN graphs produced by NSW show considerably poorer quality than the other approaches when they take similar time costs. Although NSW and OLGraph are conceptually similar to each other, they are essentially different from each other. In both OLGraph and OLGraph$^+$, the new query is attempted to insert into the neighborhoods of all the visited samples. In NSW, only the neighborhoods of returned top-k samples to the query are updated when a new query is joined in. As a result, a sample outside of the top-k ranking has no chance to

TABLE II
SCANNING RATES AND TIME CONSUMPTION OF NN-DESCENT, NSW, OLGRAPH AND OLGRAPH$^+$ ON EIGHT DATASETS

Dataset	NN-Descent		NSW		OLGraph		OLGraph$^+$	
	Scanning rate	Time(s)	Scanning rate	Time(s)	Scanning rate	Time(s)	Scanning rate	Time(s)
SIFT1M	0.01856	1278	0.00899	999	0.00699	1015	**0.00606**	**997**
SIFT10M	0.00220	15672	0.00130	16639	0.00090	**15286**	**0.00086**	15667
GIST1M	0.02406	11497	0.02029	10140	0.02140	12186	**0.01421**	**8412**
YFCC1M	0.01674	1206	0.01143	1288	**0.00732**	**1077**	0.00765	1247
GLoVe1M	0.01881	**1463**	0.01453	2010	0.01124	1920	**0.01101**	2011
NUSW-l_2	0.07655	1377	0.07624	1517	0.08749	2010	**0.05149**	**1207**
NUSW-k^2	0.07934	7546	0.08490	6592	0.08970	9441	**0.05948**	**6193**
Kosarak	0.21321	358	0.12524	317	0.12202	351	**0.11200**	**302**

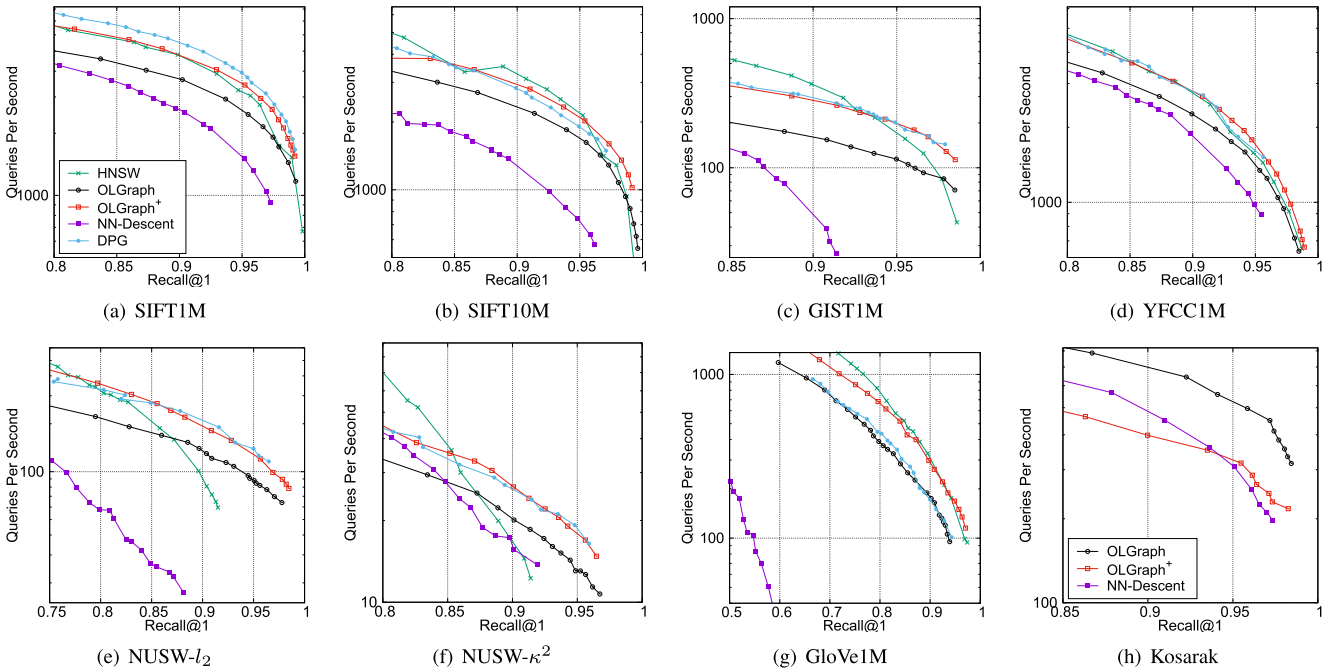


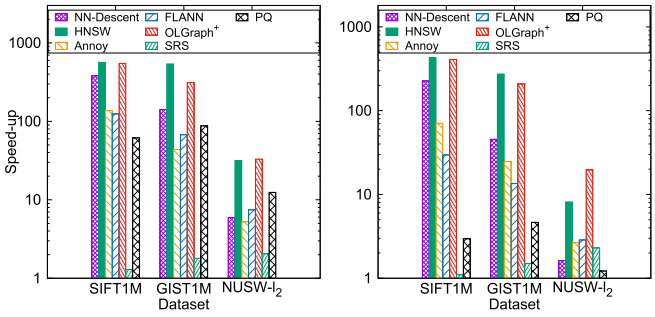
Fig. 8. The NN search performance on eight datasets. Five graph-based approaches are considered in this study. OLGraph and OLGraph$^+$ are the approaches proposed in this paper. Because the source codes of HNSW and DPG do not support sparse matrices, they do not participate in the comparison on Kosarak dataset.

add the new query into its neighborhood. For this reason, the graph quality is low. This in turn degrades its performance as an NN search indexing structure.

D. Nearest Neighbor Search

In our second experiment, the NN search performance is compared to three representative graph-based approaches NN-Descent [4], DPG [15], and HNSW [16], which work in generic metric spaces. All the approaches are based on the similar hill-climbing search procedure but different in details. They are different from each other mainly in the graphs upon which the search procedure is undertaken. For convenience, the NN search approaches based on the graph constructed by OLGraph and OLGraph$^+$ are given as OLGraph and OLGraph$^+$ respectively. The NN search on graph from OLGraph is based on Alg. 1, while NN search on OLGraph$^+$ graph is the approach presented in Section V, in which the LGD check is adopted. In the experiment, parameter k in NN-Descent is fixed to *40* for all the datasets to be in line with the experiments in [15]. DPG graph is built upon approximate k-NN graph produced by NN-Descent and undergone an offline diversification [15]. OLGraph searches over graph which is a merge of k-NN graph and its reverse k-NN graph that are produced by Alg. 2. NN search in OLGraph$^+$ is on a k-NN graph merged with its reverse k-NN graph that has been diversified online by *LGD rules* (Alg. 3). The parameter k in OLGraph and OLGraph$^+$ is also fixed to *40* for all the datasets. While for HNSW, the search is undertaken on a hierarchical small-world graph. The graph maintains the links between the close neighbors as well as the long-range links to the remote neighbors that are kept in a hierarchy. The parameter M in HNSW is fixed to *20*. The edges kept for each sample in the bottom layer is *40*. Its size of NN list is therefore on the same level as NN-Descent, OLGraph and OLGraph$^+$.

The search performance on eight datasets is shown in Fig. 8. Among all the graph-based approaches, the relative better performance is observed from DPG, HNSW, OLGraph, and OLGraph$^+$ over NN-Descent. The performance boost mainly owes to the use of reverse k-NN list and the introduction of graph diversification. The trend of OLGraph performance curve



(a) Recall@1=0.8 (b) Recall@1=0.9

Fig. 9. The NN search performance on three datasets ranging from "easy" to "hard" (best viewed in color). The performance from seven representative approaches are plotted in the figure. Figures (a) and (b) report the speed-up that one approach could achieve when the top-*1* recall is on *0.8* and *0.9* levels respectively.

is similar to that of OLGraph$^+$, whereas OLGraph$^+$ outperforms OLGraph on most of the datasets. The performance superiority of OLGraph$^+$ largely owes to the LGD strategy. Compared to HNSW, OLGraph$^+$ shows increasingly better performance as the recall is set to a high level. It is mainly because OLGraph$^+$ performs graph diversification on a high-quality approximate k-NN graph. In contrast, limited by the search framework, HNSW performs the diversification on a dynamically diversified graph. DPG shows a very similar performance as OLGraph$^+$ on most of the datasets. Both of the search approaches are based on the diversified k-NN graph. The proposed lazy graph diversification is more suitable for the online k-NN graph as the diversification is performed efficiently whenever a new sample joins in. In contrast, cross-matching required by DPG is too costly to be undertaken online.

Compared the result presented in Fig. 8(a) to the one presented in Fig. 8(b), the high scalability is observed on SIFT data by the proposed approach. As seen in the figure, the size of the reference set has been increased by one magnitude, while the time cost only increases from *0.21*ms (per query) to *0.32*ms (per query), when the search quality is maintained on *0.9* level. Similar high scalability is also observed on deep features i.e., YFCC1M (Fig. 8(d)). This is good news given the deep features have been widely adopted in various applications nowadays. In contrast, such kind of high speed-up is not achievable on NUSW, GloVe1M, and GIST1M. It is clear to see that the efficiency of graph-based approaches is partly related to the intrinsic data dimension [4], [7]. When the intrinsic data dimension is low, with the guidance of a k-NN graph or a relative k-NN graph, the hill-climbing search is actually undertaken on the subspaces where most of the data samples are embedded. Due to the low dimensionality of these subspaces, the search complexity is lower than it seemingly is. This is one of the major reasons that the graph-based approaches exhibit superior performance over other types of approaches.

E. Comparison to State-of-The-Art k-NN Search

Fig. 9 further compares our approach with the most representative approaches of different categories in the literature. Besides aforementioned HNSW and NN-Descent, approaches considered in the comparison include tree partitioning approaches Annoy [39] and FLANN [19], locality-sensitive hashing approach SRS [38], vector quantization approaches double-bit quantization (DBQ) [30], and product quantizer (PQ) [22]. In the figures, the speed-up relative to the exhaustive search that each approach achieves is reported when *recall@1* is set to *0.8* and *0.9* levels. For PQ, it is impossible to achieve top-*1* recall above *0.5* due to its heavy quantization loss. As an exception, its recall is measured at top-*16* for SIFT1M and NUSW, and measured at top-*128* for GIST1M. For DBQ, the vectors are projected by PCA first and binarized into *256* bits for both datasets SIFT1M and GIST1M.

As shown in the figure, the best results come from graph-based approaches. This observation is consistent across different datasets. The speed-up of all the approaches drops as the *recall@1* rises from *0.8* to *0.9*. The speed-up degradation is more significant for approaches such as PQ and FLANN. No considerable speed-up is observed for SRS on any of the datasets. This basically indicates SRS is not suitable for the tasks which require high NN search quality. For DBQ, the highest Recall@1 are *0.282* and *0.143* with *30.1* and *224.3* times speed-up on SIFT1M and GIST1M respectively. The representation lacks of discriminativeness due to the small distance space spanned by the binary code. Due to its low recall, the results from DBQ are not plotted in Fig. 9. An interesting observation is that the performance gap between graph-based approaches and the rest is wider on the "easy" dataset than that of "hard". Compared to the approaches of other categories, the NN search based on the graph takes advantage of the latent subspace structures in a dataset. Since the intrinsic dimension of "easy" dataset is low [7], the hill-climbing is actually undertaken on the low-dimensional subspace. The higher is the ratio between data dimension and intrinsic dimension, the higher is the speed-up of that graph-based approaches achieve. In contrast, there is such strategy in other types of approaches capitalizes on these latent structures in the data.

On the one hand, the high search speed-up is observed from OLGraph$^+$ on data types such SIFT, GIST, and deep features. With such efficiency, it is possible to realize a search system with the instant response on *100* million level dataset by a single PC. On the other hand, it is still too early to say the problem of NN search on high-dimensional data has been solved. As shown on NUSW dataset, where both the data dimension and intrinsic data dimension are high, the efficiency achieved from all the approaches is still limited. As pointed out in another work from us [48], the difficulty faced in this case is directly linked to the "curse of dimensionality," which will remain as an open issue.

VII. Conclusion

We have presented our solution for both approximate k-NN graph construction and nearest neighbor search. These two issues have been addressed under a unified framework. Namely, the NN search and NN graph construction are designed as an interdependent procedure that one is built upon another. The advantages of this design are several folds. First of all, the approximate k-NN graph construction is treated as an online procedure.

It allows the samples to be inserted in or dropped out from the graph dynamically, which is not possible from most of the existing solutions. Moreover, no sophisticated indexing structure is required to support this online approach. Furthermore, the solution has no specification on the distance measure, which makes it a generic approach both for *k*-NN graph construction and NN search. Superior performance is observed on both *k*-NN graph construction and nearest neighbor search tasks under various test configurations.

REFERENCES

[1] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, Dec. 2000.

[2] S. T. Roweis and L. K. Saul, "Report nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, Dec. 2000.

[3] C. Luo, B. Ni, S. Yan, and M. Wang, "Image classification by selective regularized subspace learning," *IEEE Trans. Multimedia*, vol. 18, no. 1, pp. 40–50, 2016.

[4] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *Proc. 20th Int. Conf. World Wide Web, WWW'11*, 2011, pp. 577–586.

[5] M. Wang, X.-S. Hua, J. Tang, and R. Hong, "Beyond distance measurement: Constructing neighborhood similarity for video annotation," *IEEE Trans. Multimedia*, vol. 11, no. 3, pp. 465–476, 2009.

[6] R. Paredes, E. Chávez, K. Figueroa, and G. Navarro, "Practical construction of k-nearest neighbor graphs in metric spaces," in *Proc. 5th Int. Conf. Exp. Algorithms, WEA'06*, 2006, pp. 85–97.

[7] B. Harwood and T. Drummond, "FANNG: Fast approximate nearest neighbour graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 5713–5722.

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

[9] J. Chen, H. ren Fang, and Yousef, "Fast approximate KNN graph construction for high dimensional data via recursive lanczos bisection," *J. Mach. Learn. Res.*, vol. 10, pp. 1989-2012, Dec. 2009.

[10] J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan, and S. Li, "Scalable k-NN graph construction for visual descriptors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 1106–1113.

[11] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. 20 Annu. Symp. Comput. Geometry*, 2004, pp. 253–262.

[12] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," in *Proc. 1984 ACM SIGMOD Int. Conf. Manag. Data*, vol. 14, (New York, NY, USA), pp. 47–57, ACM, Jun. 1984.

[13] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[14] K. Hajebi, Y. Abbasi-Yadkor, H. Shahbazi, and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph," in *Proc. Int. Joint Conf. Artif. Intell.*, 2011, pp. 1312–1317.

[15] W. Li *et al.*, "Approximate nearest neighbor search on high dimensional data-experiments, analysis and improvement," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 8, pp. 1475–1488, Apr. 2019.

[16] Y. A. Malkov and D. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 824–836, 2020.

[17] D. Comer, "Ubiquitous b-tree," *ACM Comput. Surveys*, vol. 11, no. 2, pp. 121–137, Jun. 1979.

[18] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," in *Int. Conf. Manage. Data*, pp. 322–331, 1990.

[19] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *Trans. Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2227–2240, 2014.

[20] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2008, pp. 1–8.

[21] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11259–11273, 2010.

[22] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *Trans. Trans. Pattern Anal. Mach. Intell.*, vol. 33, pp. 117–128, Jan. 2011.

[23] A. Babenko and V. Lempitsky, "Efficient indexing of billion-scale datasets of deep descriptors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2055–2063.

[24] T. Zhang, C. Du, and J. Wang, "Composite quantization for approximate nearest neighbor search," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 838–846.

[25] J. Martinez, H. H. Hoos, and J. J. Little, "Stacked quantizers for compositional vector compression," 20142014, *arXiv:1411.2173*.

[26] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Trans. Inf. Theory*, vol. 44, pp. 2325–2383, Sep. 2006.

[27] P. Wieschollek, O. Wang, A. Sorkine-Hornung, and H. Lensch, "Efficient large-scale approximate nearest neighbor search on the GPU," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2027–2035.

[28] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Trans. Big Data*, 2019.

[29] Q. Lv, W. Josephson, Z. Wang, and M. C. amd K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," in *Proc. Very Large Data bases*, Sep. 2007, pp. 950–961.

[30] H. Xie, Z. Mao, Y. Zhang, H. Deng, C. Yan, and Z. Chen, "Double-bit quantization and index hashing for nearest neighbor search," *IEEE Trans. Multimedia*, vol. 21, no. 5, pp. 1248–1260, 2019.

[31] J. Wang and S. Li, "Query-driven iterated neighborhood graph search for large scale indexing," in *Proc. 20th ACM Int. Conf. Multimedia*, 2012, pp. 179–188.

[32] Y. A. Malkov, A. Ponomarenko, A. Lovinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Inf. Syst.*, Elsevier, vol. 45, pp. 61–68, 2014.

[33] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," in *Proc. VLDB Endowment*, vol. 12, Jan. 2019, pp. 461–474.

[34] Y.-M. Zhang, K. Huang, G. Geng, and C.-L. Liu, "Fast knn graph construction with locality sensitive hashing," in *Proc. Mach. Learn. Knowl. Discov. Databases: Eur. Conf.*, Sep., 2013. pp. 660–674.

[35] J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan, and S. Li, "Scalable K-NN graph construction for visual descriptors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 1106–1113.

[36] T. Debatty, P. Michiardi, and W. Mees, "Fast Online K-NN Graph Building," *CoRR*, 2016, *arXiv:1602.06819*.

[37] E. Levina and P. J. Bickel, "Maximum likelihood estimation of intrinsic dimension," *Adv. Neural Inf. Process. Syst.*, pp. 777–784, 2005.

[38] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin, "SRS: Solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny," in *Proc. VLDB Endowment*, Sep. 2014. pp. 1–12.

[39] E. Bernhardsson, "Annoy: Approximate nearest neighbors in C++/python optimized for memory usage and loading/saving to disk," GitHub https://github.com/spotify/annoy, 2017.

[40] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[41] M. Douze, H. Jégou, H. Singh, L. Amsaleg, and C. Schmid, "Evaluation of GIST descriptors for web-scale image search," in *Int. Conf. Image and Video Retrieval*, pp. 19: 1–19:8, Jul. 2009.

[42] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods Natural Lang. Process.*, pp. 1532–1543, 2014.

[43] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng, "Nus-wide: A real-world web image database from national university of singapore," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2009, pp. 1–9.

[44] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Proc. Int. Conf. Comput. Vis. Workshop*, 2003, pp. 1470–1477.

[45] G. Amato, F. Falchi, C. Gennaro, and F. Rabitti, "YFCC100 M hybridnet fc6 deep features for content-based image retrieval," in *Proc. ACM Workshop Multimedia COMMONS*, 2016, pp. 11–18.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[47] M. Aumüller, E. Bernhardsson, and A. Faithfull, "Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," in *Int. Conf. Similarity Search Appl.*, 2017, pp. 34–49.

[48] P.-C. Lin and W.-L. Zhao, "A Comparative Study on Hierarchical Navigable Small World Graphs," *CoRR*, vol. abs/1904.02077, 2019.

Wan-Lei Zhao received the B.Eng. and M.Eng. degrees from the Department of Computer Science and Engineering from Yunnan University, Kunming, China, in 2006 and 2002, respectively, and the Ph.D. degree from the City University of Hong Kong, Hong Kong, in 2010. He is currently with Xiamen University, Xiamen, China, as an Associate Professor. Before joining Xiamen University, he was a Postdoctoral Scholar with INRIA, Le Chesnay, France. His research interests include multimedia information retrieval and video processing.

Hui Wang received the bachelor's degree in engineering from Zhejiang Sci-Tech University, Hangzhou, China, in 2019. He is currently working toward the master's degree with the School of Informatics, Xiamen University, Xiamen, China. His research focuses on large-scale nearest neighbor search.

Chong-Wah Ngo received the B.Sc. and M.Sc. degrees in computer engineering from the Nanyang Technological University of Singapore, Singapore, and the Ph.D. degree in computer science from the Hong Kong University of Science and Technology, Hong Kong. He is currently a Professor with the School of Computing and Information Systems, Singapore Management University, Singapore. Before joining Singapore Management University, he was with the Department of Computer Science, City University of Hong Kong, Hong Kong. He was a Postdoctoral Scholar with the Beckman Institute, University of Illinois, Urbana-Champaign, Champaign, IL, USA. His main research interests include large-scale multimedia information retrieval, video computing, multimedia mining, and visualization. He was an Associate Editor for the IEEE TRANSANCTION ON MULTIMEDIA and is currently a Steering Committee Member of the TRECVid, *International Conference on Multimedia Retrieval* and ACM Multimedia Asia. He is the Program Co-Chair of the ACM Multimedia 2019, and the General Co-Chair of ICIMCS 2018 and PCM 2018. In 2016, he was named ACM Distinguished Scientist for contributions to video search and semantic understanding.