Optimizing Graph-based Approximate Nearest Neighbor Search: Stronger and Smarter

Jun Liu[1], Zhenhua Zhu[1], Jingbo Hu[1], Hanbo Sun[1], Li Liu[2], Lingzhi Liu[2], Guohao Dai[1], Huazhong Yang[1], Yu Wang[1]

[1] Department of Electronic Engineering, BNRist, Tsinghua University, Beijing, China

[2] Heterogeneous Computing Group, Kuaishou Technology, Palo Alto, CA, USA

{liu-j20, zhuzhenh18, hjb19, sun-hb17}@mails.tsinghua.edu.cn,
liliu@kwai.com, liulingzhi@kuaishou.com, {daiguohao, yanghz, yu-wang}@mail.tsinghua.edu.cn

***Abstract*—Approximate Nearest Neighbor Search (ANNS) is widely used in many fields (e.g., recommender systems). In recent years, the graph-based ANNS method has attracted the attention of many researchers due to its superiority compared to non-graph-based algorithms. Compared with traditional recommender systems, mobile recommender systems have higher latency requirements [1]. The graph-based ANNS method faces the following challenges that make it difficult to meet the requirements. (1) Poor connectivity. Due to the limitation of the construction algorithm, the connectivity of the graph index is poor, which in turn affects the search performance. (2) Redundant search. Existing search algorithms use sufficiently long search steps for all queries to achieve high search accuracy. However, the query search steps follow the long-tailed distribution that brings the redundant search, e.g., for more than 40% of the queries, 87% of the search overhead is redundant.**

We propose two optimization strategies to tackle the above challenges. (1) Reverse connection enhancement strategy. In the graph construction process, we increase the in-degree of the point to be inserted to enhance the graph connectivity, while keeping the out-degree low to maintain the high search efficiency. (2) Query aware early termination strategy. We identify regional features to predict the number of remaining search steps to achieve dynamic search termination and reduce the redundant search overhead. Finally, we verify the proposed solutions on multiple representative datasets. Compared with the existing graph-based ANNS method, our solutions can improve the search speed by 1.06x - 1.29x.

***Index Terms*—Approximate Nearest Neighbor Search, Recommendation System, Data Retrieval, Big Data**

I. Introduction

The Nearest Neighbor Search (NNS) problem is widely used in many fields, such as recommendation systems [2], data retrieval [3], information matching [4], and machine learning [5]. A primitive and intuitive way to solve this problem is the brute force method. But as the number of mobile data increases, the calculation time of the brute force method becomes unacceptable. To tackle this problem, recent work uses Approximate Nearest Neighbor Search (ANNS) methods to reduce the search time by a few orders of magnitude with a slight accuracy loss [6]–[9]. ANNS meets the requirements of different scenarios by exploring the trade-off between accuracy and latency.

At present, ANNS methods can be divided into four categories: tree-based methods [6], [10]–[12], locality-sensitive hashing (LSH) based methods [7], [13]–[15], product quantization (PQ) based methods [8], [9], [16], [17], and graph-based methods [18]–[22]. Among these four categories, the graph-based methods show great potential in recent years and achieve a high recall rate in a short search time [23]. The graph-based methods build an approximate nearest neighbor graph, and then use the graph as an index (i.e., graph index) for greedy search. Most non-graph ANNS methods search for queries through spatial division, so these methods often require access to more data points [24]. The graph-based ANNS methods establish the connection relationship between data points, and the query can be quickly approached by only visiting fewer points. Therefore, the graph-based methods are more efficient than the non-graph methods.

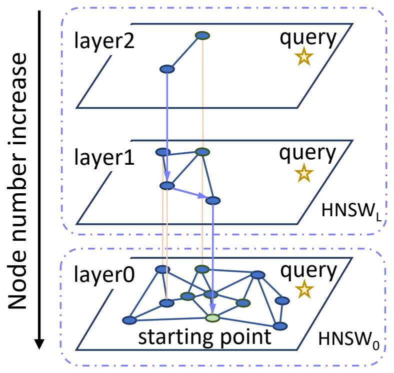

Fig. 1. HNSW [25] is composed of multiple layers, and the closer to the bottom layer, the more nodes in the layer. The starting search point of $HNSW_0$ is determined by $HNSW_L$.

Hierarchical Navigable Small World graphs (HNSW) [18] is one representative graph-based ANNS method, which is widely used in commercial application scenarios due to its excellent search performance and the support for incremental graph construction [23]. As Fig. 1 shows, HNSW constructs a multi-layer graph structure. The hierarchical graph structure ($HNSW_L$) randomly provides long inter-layer links for some points, and the starting search point can be quickly found through these points. Then HNSW uses a greedy search algorithm to search for the k nearest points to the query in the bottom layer ($HNSW_0$).

TABLE I
SYMBOLS AND THEIR INTERPRETATION.

Symbols	Interpretation
X	The base vectors of dataset
N	The number of base vectors X
p_{in}	Insertion point: The point that is being added to the graph index during the construction process
efc	The parameter of the construction process, weigh the construction time and graph index performance.
efs	The parameter of the search process, weigh the search speed and search accuracy.
$maxM0$	The maximum number of neighbors per point on the graph index (the bottom layer for HNSW)

Algorithm 1 Construction algorithm

```
Input: Base vectors $X$, the number of maximum neighbors $maxM0$, starting point $p_s$
Output: Complete graph index $G_N$
1: for each point $p_{in} \in X$ do
2:     $Q^{in}_{cand} \leftarrow$ greedy search$(q, G_s, p_s, efc, efc)$
3:     $Q^{in}_{sel} \leftarrow$ SelectNeighborByRNG$(p_{in}, Q^{in}_{cand}, maxM0)$
4:     $setNeighbor(p_{in}) \leftarrow Q^{in}_{sel}$
5:     for each point $p_r \in Neighbor(p_{in})$ do
6:         $addNeighbor(pr) \leftarrow p_{in}$
7:     end for
8: end for
9: return $G_N$
```

Compared with traditional recommender systems, mobile recommender systems have higher latency requirements [1]. HNSW still faces the following challenges, which make it difficult to meet the requirements. The first challenge is that **the connectivity of the graph index is poor**. 84.6% of the recall rate loss is caused by poor connectivity of the graph index. Poor connectivity is mainly reflected in the existence of some points with small in-degree on the graph index. These small in-degree points are hard to find, and if they belong to the ground truth, they will cause recall loss. The second challenge is **the heavy redundant search overhead in the search process**. In order to achieve the high search accuracy of all various queries (e.g., recall rate $>$ 0.95), the existing search strategy uses sufficiently long search steps as the search termination condition. However, the number of minimum search steps for different queries vary greatly, and the number of search steps follow the long-tailed distribution. When applying the sufficiently long search steps, more than 40% of the queries only need 12.6% of the total search step, revealing heavy redundant search overhead.

To tackle these challenges, we propose two optimization solutions to **make the graph index stronger** and **make the search smarter**. The main contributions are shown as follows:

1) For the challenge of unsatisfactory search results caused by poor graph connectivity, we conducted a thorough analysis of the existing construction strategy. Then we indicate that the existing construction strategy restricts the in-degree of each point, which leads to the poor connectivity of the graph index. We propose the reverse connection enhancement strategy to increase the graph connectivity while maintaining search efficiency.
2) For the challenge of the heavy overhead of redundant search, we propose the query-aware early termination strategy. We indicate that the redundant search problem is caused by the region connection relationship near the query, which can be represented by the distance change during the search process. Then we predict the number of remaining search steps by identifying the region connection relationship.
3) We validate our method on multiple datasets. Under the condition of maintaining the same search performance, the search speed is increased by 1.06x - 1.29x.

II. Preliminaries

In this section, we introduce the ANNS problem and the state-of-the-art ANNS algorithms. Table. I shows the symbols that need to be used in the full text.

A. Approximate Nearest Neighbor Search

1) Problem Definition: Nearest Neighbor Search [26] (NNS) is to find a result that is the nearest or most similar to the given **query vector** in a d-dimensional finite set $X \in \mathbb{R}^d$. The NNS problem is defined as:

$$res = \underset{x \in X}{\arg\min}\, dist\langle q, x\rangle \tag{1}$$

Here X is the **base vectors**, query $q \in \mathbb{R}^d$ is a certain vector in the query vectors, $dist\langle q, x\rangle$ is the distance metric to indicate the similarity between q and x. $res \subseteq X$ is the nearest result to q in the X, similarly, we denote the nearest k results as res_k.

Unfortunately, the exhaustive search used by NNS is not feasible on the large-scale datasets. Therefore, researchers propose the Approximate Nearest Neighbor Search (ANNS) to speed up the search process. Although the goal of ANNS is the same as that of NNS, the result res'_k found by ANNS may not be the k nearest results in X. We use search accuracy (or recall rate) to represent the similarity between res'_k and res_k. Compared with NNS, ANNS can achieve a better trade-off between search accuracy and search speed.

2) Distance Measurement: Euclidean distance (e.g., DEEP [27] and Turing [28] dataset) is the most commonly used measurement in ANNS to describe the similarity between two vectors. Here we take the D-dimensional vector $a = (a_1, a_2, \cdots, a_D)$ and vector $b = (b_1, b_2, \cdots, b_D)$ as examples

Algorithm 2 SelectNeighborByRNG($p_c, S_{cand}, maxM0$)

```
Input: point $p_c$, candidate set $S_{cand}$, the number of maximum
    neighbors $maxM0$.
Output: Selected neighbor set $S_{sel}$.
1: $S_{sel} \leftarrow \emptyset$
2: for $x_i \in S_{cand}$ do
3:     if $S_{sel}.size = maxM0$ then
4:         break
5:     end if
6:     $dist \leftarrow getDistance(p_c, x_i)$
7:     for $s_i \in S_{sel}$ do
8:         $d \leftarrow getDistance(x_i, s_i)$
9:         if all $d > dist$ then
10:            $S_{sel} \leftarrow S_{sel} \cup x_i$
11:        else
12:            break
13:        end if
14:    end for
15: end for
16: return $S_{sel}$
```

to introduce the above distance measurement. The distance between vectors a and b is denoted as $dist\langle a, b\rangle$. Its calculation formula is as follows:

$$dist\langle a, b\rangle = \sqrt{\sum_{i=1}^{D} (a_i - b_i)^2} \quad (2)$$

3) Evaluation Metric: Three metrics are usually used to evaluate the performance of ANNS algorithms.

- Recall rate [29]: In ANNS, the recall rate is usually used to indicate the search accuracy. Assume that the number of query vectors is N_q. For each query vector q_i, its ground truth set of the top k nearest neighbors is GT_{ik}. The result set returned by ANNS is RES_{ik}, then the recall rate $Recall@k$ is defined as follows:

$$Recall@k = \sum_{i=1}^{N_q} \frac{|RES_{ik} \cap GT_{ik}|}{|GT_{ik}|} \quad (3)$$

- Queries Per Second (QPS): QPS is usually used to measure the search speed of the ANNS method. Its meaning is the number of queries that can be searched per second.
- The Number of Distance Calculations (NDC): it is positively correlated with the actual search overhead. Therefore, NDC is often used to evaluate the search speed of different ANNS algorithms and system implementations [20], [30]. For different hardware or code implementations, comparing NDC is a fair evaluation method.

B. Graph-based ANNS Methods

In the graph-based ANNS methods, we treat the base vectors X as points in d-dimensional space and then connect them through edges. These points and the edges between them construct the graph index. If there is exists an edge from x_i to x_j in the graph index, we call x_j is a neighbor of x_i. The search process means that we start from a certain point (starting search point) on the graph index and iteratively search for points closer to the query along these edges. The construction process is to connect edges between these points. The main difference between different ANNS methods lies in the algorithms used in the construction process.

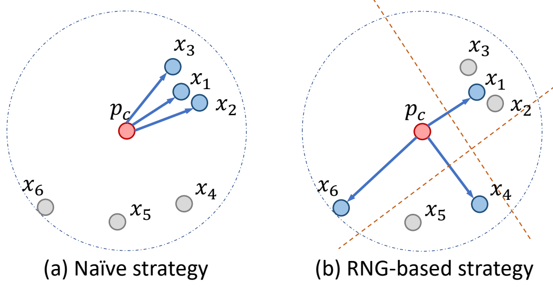

Fig. 2. The intuitive comparison of the three neighbor selection strategies is in 2-dimensional space. (a) The naïve strategy only selects points that are closer to each other as neighbors. (b) The RNG-based strategy makes the neighbors of p_c have better diversity and improves the efficiency of graph index.

1) NSG: Fu et al. [20] proposed Navigating Spreading-out Graph (NSG) to re-select neighbors for each point on the pre-constructed k-Nearest Neighbor (kNN) Graph. The neighbor selection strategy based on the Monotonic Relative Neighborhood Graph (MRNG) ensures that each step is closer to the query than the previous step, but it would lead to excessive construction complexity. Therefore, NSG adopts an approximate MRNG strategy to fix the center point as the starting search point. NSG is difficult to implement construction incrementally because it requires a pre-constructed kNN graph.

2) HNSW: Hierarchical Navigable Small World graphs (HNSW) [18] is one representative graph-based ANNS method. As shown in the Fig. 1, the HNSW is composed of multiple graph layers, and there are connections between points in the same graph layer (abbreviated it layer). During the construction process, each point in the base vectors X is inserted into the graph index one by one. The highest layer of each point is determined by a random function with exponential decay of probability, and then this point (*insertion point*) needs to be inserted in all layers below the highest layer. Then, the bottom layer contains all the points in the base vectors.

As shown in Alg. 1, for each insertion point p_{in}, the construction process can be divided into three stages: the get candidate stage (line 2), the forward connection stage (line 3 to line 4), and the reverse connection stage(line 5 to line 7). The get candidate stage is to find the nearest efc points for p_{in} in the current graph index G_c as neighbor candidates Q_{cand}^{in}. The forward connection stage select Q_{sel}^{in} as p_{in} neighbors through the neighbor selection strategy based on Relative Neighborhood Graph (RNG) [31]. In the reverse connection stage, for all points p_r in p_{in} neighbors, use a similar method to reconnect their neighbors.

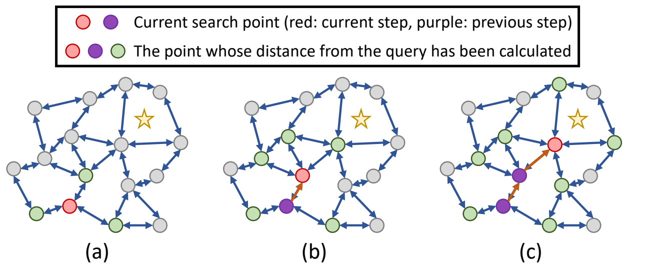
Current search point (red: current step, purple: previous step)
The point whose distance from the query has been calculated
(a)
(b)
(c)

Fig. 3. Schematic diagram of searching query nearest neighbors on a graph index. (a)-(c) The first three steps of the search process.

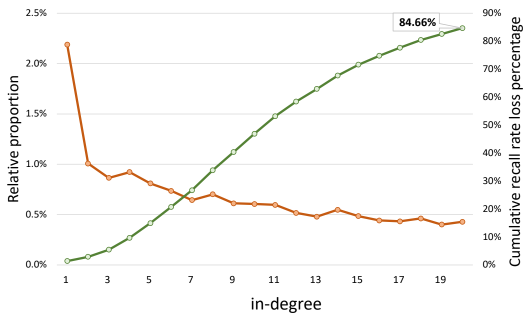

Fig. 4. (a) Relative proportion of points that cannot be found (left axis) versus in-degree, which cause the recall rate loss (on the Turing1M dataset). (b) Cumulative percentage of recall rate loss (right axis).

The RNG-based neighbor selection strategy (and its variants) is widely used in graph-based ANNS construction algorithms due to its high efficiency. The naïve strategy connects the nearest points to the point p_c as shown in the Fig. 2(a), which cause the search process to easily fall into a local optimum (because the neighbors of point p_c are very concentrated). The RNG-based strategy (shown in Fig. 2(b)) selects a part of points from the candidate set, and these points are as scattered as possible with respect to p_c.

During the search process, for all layers except the bottom layer, HNSW starts from the top layer and uses the greedy search algorithm to find the nearest neighbor for the query. From then on, the nearest neighbor enters the next layer and continues to search for the nearest neighbors of the graph until the nearest neighbor is found at the bottom layer. Then HNSW uses a greedy search algorithm to search for the k nearest points to the query in the bottom layer. We sequentially show the search process of the first three steps in the bottom layer through (a) to (c) in Fig. 3. At the beginning of the search process, there is only the starting search point p_s in the search queue Q_{sn}. For each step of the search process, we pop the point nearest to the query (pentagram) from the search queue as the current search point p_f (red). Then we calculate the distances between all neighbors of the current search point and the query and add these neighbors to the search queue and result queue Q_r. The search queue consists of all green points and always keeps the efs points nearest to the query in non-gray points (green, red, and purple). The purple point represents the current search point from the previous step. Then we perform the next step of search until the termination condition (line 7 to line 8) is met. Finally, the k points nearest to the query in the result queue Q_r are used as the search result.

III. Reverse Connection Enhancement

A. Motivation

In the graph-based ANNS methods, we approach the query through continuous iterative search based on the connections between points in the graph index. Therefore, the connectivity of the graph index is crucial to the final search results. Specifically, the greater the in-degree of a certain point, the more points can find it (search accuracy higher). The larger the out-degree of a certain point, the larger the search cost at each step (search speed lower). The existence of some points with small in-degree means that the graph index has poor connectivity. If these points belong to the ground truth but are not found during the search, this will lead to a lower recall rate. We illustrate this through experiments (shown in Fig. 4), we plot the distribution using these points as a percentage of the total points corresponding to the in-degree. Obviously, if the in-degree of a point is smaller, then it has a greater negative impact on the recall rate. And the recall rate loss due to this situation accounts for 84.66% of the total recall rate loss.

The poor connectivity of graph index is due to the restriction of the existing construction algorithm, which is the in-degree of the certain point is restricted by its out-degree. In order to keep the graph index efficient, the out-degree of some points will be small (by neighbor selection). For points with small out-degrees, their in-degrees are also small due to the above restriction. Next, we will conduct a detailed analysis of the out-degree and in-degree of the graph index and propose our solution for the challenge.

B. Degree Analysis

The construction process is to add the point (*insertion point*) in the base vectors X to the graph index one by one. As shown in Fig. 5, at time t_c, the corresponding insertion point is x_c, the corresponding graph index is G_c. The whole construction process can be divided into two parts: before t_c and after t_c. Then, the in-degree $IDG(x_c)$ and out-degree $ODG(x_c)$ of x_c are correspondingly expressed in two parts, as shown below:

$$IDG(x_c) = IF1_{x_c} + IF2_{x_c} \quad (4)$$

$$ODG(x_c) = OF1_{x_c} + OF2_{x_c} \quad (5)$$

- $OF1_{x_c}$: It is the out-degree of x_c before time t_c. To be specific, when insertion point is x_c in the construction process (before t_c in Fig. 5), the efc nearest points of x_c found in the graph index G_c (before t_c) are called candidates. We select some of these candidates as neighbors of x_c ($Neighbor(x_c)$) by the RNG-based neighbor selection strategy. The number of these neighbors is $OF1_{x_c}$.
- $IF1_{x_c}$: It is the in-degree of x_c before time t_c. To be specific, when insertion point is x_c in the construction

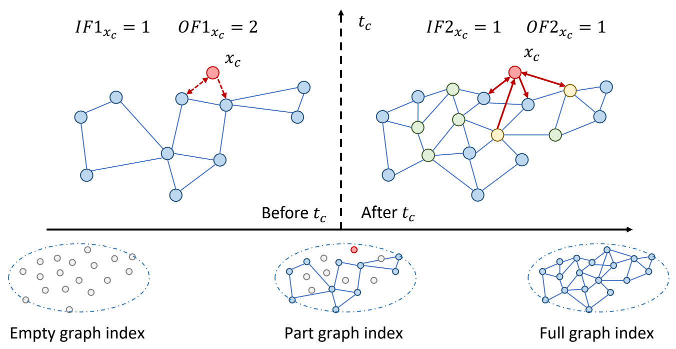

Fig. 5. The in-degree (out-degree) of point x_c is composed of two parts $IF1(OF1)$ and $IF2(OF2)$. $IF1(OF1)$ comes from the point (blue) in the graph index before t_c, and $IF2(OF2)$ comes from the point (green) added to the graph index after t_c.

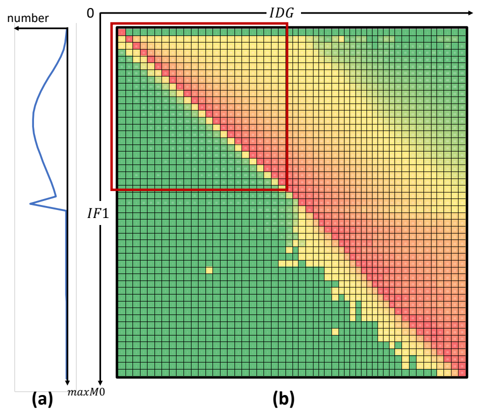

Fig. 6. (a) $IF1$ distribution curve for all points on the graph index (on the Turing1M dataset). (b) Heatmap of IDG and $IF1$, there is a positive correlation between the two for small in-degree points (inside the red box). Each row of the matrix is normalized according to (a). *In-degree of individual points (about 0.3‰) is very large ($\geq maxM0$), and we truncate them.

process (before t_c in Fig. 5), and we have selected the current neighbors $Neighbor(x_c)$ of x_c. For each point in $Neighbor(x_c)$, we also need to add x_c as their neighbor when the number of their neighbors is less than $maxM0$. The number of points that finally successfully added x_c as a neighbor is $IF1_{x_c}$, obviously $IF1_{x_c} \leq OF1_{x_c}$.

- $IF2_{x_c}$: It is the in-degree of x_c after time t_c. To be specific, when x_c is already in the graph index (after t_c in Fig. 5), x_c may also become a candidate of other insertion points (the green points). These number of these insertion points, which add the edge from them to x_c, is $IF2_{x_c}$.
- $OF2_{x_c}$: It is the out-degree of x_c after time t_c. To be specific, when x_c is already in the graph index (after t_c in Fig. 5), and there are some points (yellow points) added x_c as a neighbor. Then we also need to add these points as the neighbor of x_c when the number of $Neighbor(x_c)$ is less than $maxM0$. The number of new neighbors for x_c is $OF2_{x_c}$.

As shown in Fig. 5, $IF2_{x_c}$ and $OF2_{x_c}$ depend on whether

Algorithm 3 Reverse connection enhancement algorithm

Input: Base vectors X, the number of maximum neighbors $maxM0$, starting point p_s
Output: Complete graph index G_N

```
1:  for each point $p_{in} \in X$ do
2:      $Q^{in}_{cand} \leftarrow$ greedy search$(q, G_s, p_s, efc, efc)$
3:      $Q^{in}_{sel} \leftarrow$ SelectNeighborByRNG$(p_{in}, Q^{in}_{cand}, maxM0)$
4:      $setNeighbor(p_{in}) \leftarrow Q^{in}_{sel}$
5:      $Q_{add} \leftarrow Q^{in}_{cand} \setminus Q^{in}_{sel}$
6:      for each point $p_r \in Neighbor(p_{in})$ do
7:          $addNeighbor(pr) \leftarrow p_{in}$
8:      end for
9:      while $Indegree(p_{in}) < maxM0$ do
10:         $p_r \leftarrow Q_{add}$
11:         $addNeighbor(pr) \leftarrow p_{in}$
12:     end while
13: end for
14: return $G_N$
```

x_c can be found by other points (green points). Obviously, the greater the in-degree $IF1_{x_c}$ at t_c, the greater the probability of x_c being found after t_c. Fig. 6(b) demonstrates the relationship between $IDG(\cdot)$ and $IF1$. If $IF1$ of some points (inside the red box) is small, then their final in-degree $IDG(\cdot)$ are also small. For these points, there is a positive correlation between $IDG(\cdot)$ and $IF1$.

The connectivity of the point x_c in the full graph index G_N largely depends on $IF1_{x_c}$. At the same time, there is a **restrictive relation** between $IF1_{x_c}$ and $OF1_{x_c}$, i.e., $IF1_{x_c} \leq OF1_{x_c}$. The RNG-based neighbor selection strategy can maintain the high efficiency of the graph index [18], which results in 80.5% of the points with $OF1$ less than 2/5 of $maxM0$ (as shown in Fig. 6(a), because the distributions of $IF1$ and $OF1$ are almost identical). The existence of restrictive relation between $IF1_{x_c}$ and $OF1_{x_c}$ leads to poor connectivity of the graph G_N.

C. Our Method

In order to solve the above problem, we propose the reverse connection enhancement strategy (shown in Alg. 3). We add some connections to points with small in-degrees, which makes $IF1$ break the restrictive relation with $OF1$. Specifically, the incoming edge of point x_c is not only selected from its neighbors but also from a part of its candidates. Fig. 7 visually shows our method, x_c is the insertion point in the construction process, and the points (x_1, x_2, and x_3) are the candidates of x_c. Then we select x_1 and x_3 as the neighbors of x_c through the RNG-based neighbor selection strategy. The difference between our method and the previous method is mainly reflected in the following $IF1_{x_c}$. Since x_2 and x_4 are closer to x_3 than x_c, so x_c does not belong to $Neighbor(x_3)$. Due to the restriction of $OF1_{x_c}$ to $IF1_{x_c}$, the previous method can only find x_c through x_1, which means that it is difficult for other points to find x_c. Our method breaks the restrictive relationship between $IF1_{x_c}$ and $OF1_{x_c}$,

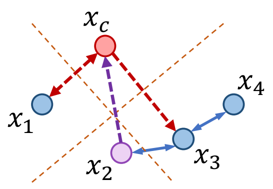

Fig. 7. Diagram of the reverse connection enhancement strategy. Before enhancement: we can only find the point x_c through point x_1. After enhancement: we can find the point x_c by either point x_1 or point x_2.

we select some points (such as x_2) from the candidates of x_c and then taking x_c as the neighbor of x_2 (connected by purple edges). Then we can find x_c by either point x_1 or point x_2. Compared with the previous method, our method greatly improves the connectivity of the graph. The detailed comparison results are shown in Section V.

IV. Query Aware Early Termination

A. Motivation

The existing search algorithm incurs heavy redundant search overhead, which is widely used in most graph-based ANNS methods [18], [20]. The parameter efs, which is the maximum length of the result queue during the search process, can control the relationship between search speed and search accuracy (recall rate). The larger efs, the higher accuracy, and the lower speed. The number of search steps for each query is approximately proportional to the efs. Here we define the *minimum search steps* as follows:

Definition 1. *(minimum search steps) Assuming the query q_i uses $efs = s$ to search on the graph index, its recall rate is $Recall_i$. There is a minimum value that keeps the recall rate of q_i does not decrease. Then the minimum value is the minimum search steps for q_i.*

We find that for different queries, their minimum search steps vary widely. However, for each query in the query vectors, the existing algorithm will use the same efs to search. Each query in the query vectors has a similar number of search steps. For the case of efs=200, the average number of search steps is 200.71, and the standard deviation is 1.91. We further illustrate this problem through Fig. 8, the average number of search steps for these queries is 300, but more than 40% of the queries have the minimum search steps of less than 39. Therefore, there will be redundancy in the search process. For these queries, 87% of the search overhead is redundant (34.8% redundant overhead for the whole). Next, we will analyze the causes of this problem in detail and propose our solutions.

B. Analysis

The redundant search problem is caused by the region connection relationship near the query in the graph index. The region means a part of the graph index near the query. Fig. 9 shows the distribution of the minimum search steps for the query vectors, the horizontal and vertical axes correspond to two different graph indexes, respectively. The two graph

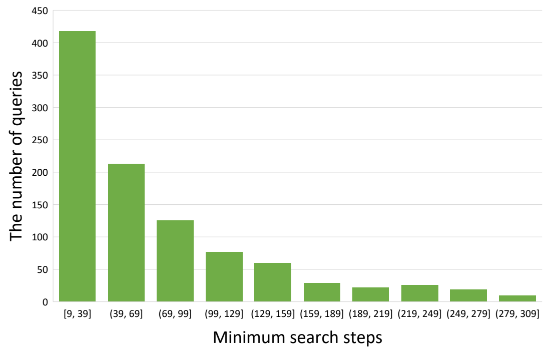

Fig. 8. Histogram of the distribution of the minimum search steps for queries. More than 40% of the queries have the minimum search steps of less than 39, but their actual search steps are around 300 (on the DEEP1M dataset).

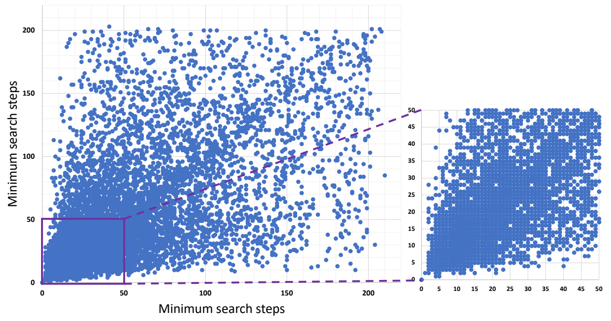

Fig. 9. The distribution of the minimum search steps for query vectors in different graph indexes, which only change the order of insertion points during the construction process. We find that for the same query, its minimum search steps varies greatly on different graph indexes (on the DEEP1M dataset).

indexes have the same base vectors and parameters, and we only change the order of insertion points during the construction process. As shown in Fig. 9, we find that for the same query, there is a huge difference (e.g., 3 to 190) between its minimum search steps. It means that the region connection relationship near the query causes this problem. According to the connection relationship, regions can be divided into two types: symmetric connection region (Fig. 10(a)) and asymmetric connection region (Fig. 10(b)). The symmetric connection means the neighbors of a certain point are evenly distributed in space. The asymmetric connection means the neighbors of a certain point are non-uniform distributed in space.

The **asymmetric connection region** requires more search steps to achieve the same recall rate compared with the symmetric connection region. When the current search point is very near to the query. For the symmetric connection region, the current search point can quickly perform a complete exploration of the nearby region of the query. Therefore, the symmetric connection region can find all the nearest points of the query with fewer search steps. For the asymmetric connection region, due to the lack of connection between the points near the query, the current search point takes some "detours" to find all the nearest points of the query. As shown

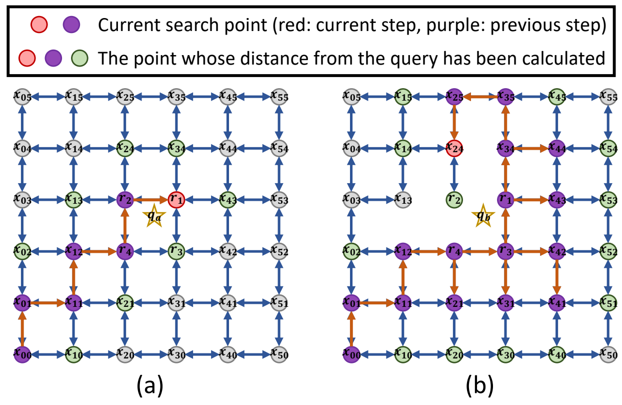
Current search point (red: current step, purple: previous step)
The point whose distance from the query has been calculated
q_a
q_b
r_1
r_2
r_3
r_4
(a)
(b)

Fig. 10. Schematic diagram of searching query nearest neighbors on a graph index. (a) When query q_a is located in a symmetric connection region, we can easily find all its nearest neighbors. (b) When query q_b is located in an asymmetric connection region, we need more search steps to find all the nearest neighbors.

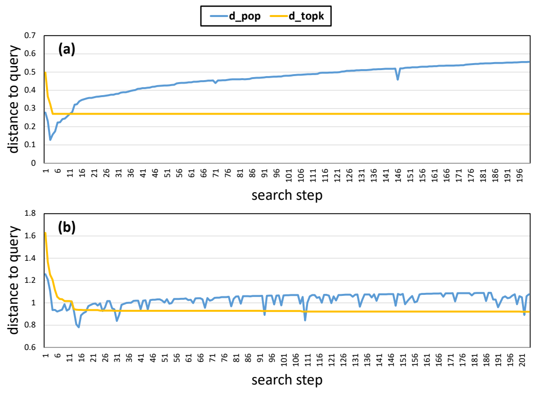

Fig. 11. The changes of d_{pop} and d_{topk} for two representative queries during the search process (on the DEEP1M dataset). (a) The query is located in a symmetric connection region, and its minimum search steps is 6. (b) The query is located in an asymmetric connection region, and its minimum search steps is 200.

in Fig. 10, the minimum search steps for q_a is only 7, and the minimum search steps for q_b is as high as 17. If we want to get the same high recall rate, there is redundancy in the search overhead of q_a.

Since the connection relationship in high-dimensional space is very complex, it is difficult to solve this problem from the construction algorithm. Therefore, we address this problem from the search algorithm perspective. We define two variables d_{pop} and d_{topk} to identify the connection relationship of the region. The distance here refers to the distance between a certain point and the query. For each step of the search process, d_{pop} represents the distance of the current search point, and d_{topk} represents the distance from the k-th nearest point in the result queue to the query.

We identify these two types of regions by the changes in d_{pop} and d_{topk} during the search process. When the current search point is very near to the query, if the curve d_{pop} is smooth relative to the curve d_{topk}, it means that the region connection relationship is symmetric. For the symmetric connection region, since the neighbors are very evenly distributed, the points near the query will be added to the search queue. Therefore, the curve d_{pop} is smooth relative to the curve d_{topk} (as shown in Fig. 11(a)). For the asymmetric connection region, the non-uniform distribution of points' neighbors results in no connections between some points within the region. In this case, at each next step, d_{pop} is first incrementally increased. Then HNSW will find a point closer to the current search point pass the "detour", the d_{pop} will drop. So there will be some fluctuations in the curve d_{pop} (as shown in Fig. 11(b)).

We illustrate that further with Fig. 10. For query q_a, when point r_1 is used as the current search point, since the connection of the region a is symmetric, the search process will spread outward with q_a as the center. The change of the distance between the current search point and query q_a on the region a is similar to the curve d_{pop} in Fig. 11(a). For query q_b, when point r_1 is used as the current search point, since the connection of the region b is asymmetric (point r_2 is not in the search queue), the search process will spread in the lower right and upper right directions relative to q_b. We can't find r_2 until the distance between point x_{24} and query q_b is calculated. The change of the distance between the current search point and query q_b on the region b is similar to the curve d_{pop} in Fig. 11(b).

C. Our Method

In order to solve the problem of redundancy in the search process, we propose the query-aware early termination strategy. Based on the above analysis, we indicate that the curve feature of d_{pop} relative to that of d_{topk} can be used to identify the region during the search process.

We design a prediction model to decide whether to stop early by analyzing the curve smoothness of d_{pop} relative to d_{topk} during the search process. To remove the effect of inconsistent distances between each query and its nearest neighbors, we additionally consider the d_{top1} feature. The features we used are as follows:

- d_{pop}: In the current step, the distance between the current search point and the query.
- d_{topk}: After the current step of the search is over, the distance from the k-th nearest point in the result queue to the query.
- d_{top1}: After the current step of the search is over, the distance from the nearest point in the result queue to the query.

For the complete workflow (as shown in Fig. 12), we do not need to add any additional structure to the graph index. In the training stage (Fig. 12(a)), we use the learning vectors as the training set (and also divide it apart as the validation set). For each data in the training set, we will all perform a complete search process according to the above search algorithm. We can easily get the distances (d_{pop}, d_{topk}, and d_{top1}) at each step as the train feature for the prediction model. Then we calculate the minimum search steps and compare it with the

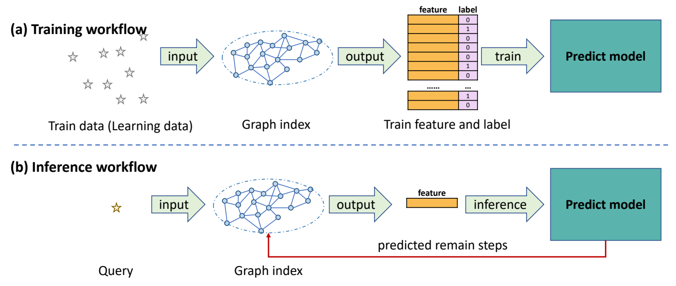

Fig. 12. Workflow for query-aware early termination strategy. (a) Training workflow, we first use the sampled data to search on the graph index, and then use the resulting dynamic features to train the prediction model during the search process. (b) Inference workflow, we integrate the trained model into the framework of graph index, and dynamically predict the remained search steps during the search process of query.

length of the curve to get the label. label=1 means "continue search", label=0 means "terminate search". In the inference stage (Fig. 12(b)), we collect the dynamic features in the search process and predict the remaining search steps through the trained prediction model.

The training process of the prediction model mainly decides whether to stop the search process by analyzing the smoothness of the d_{pop} sequence. The specific process is as follows: (1) finding the rising intersection step of d_{pop} curve and d_{topk} curve. If from the st_{stable}-th step, d_{pop} is greater than d_{topk}, then st_{stable} is called the rising intersection step. (2) obtaining the d_{pop} sequence. Starting from the st_{stable}-th step, d_{pop} is added to the d_{pop} sequence in next each step until the difference between d_{pop} and d_{topk} is greater than the difference between d_{topk} and d_{top1}. (3) difference-by-difference operation for the d_{pop} sequence. For the d_{pop} sequence, we do n difference-by-difference operations on it (similar to the n-order derivative for the continuous curve). (4) comparing the variance of the d_{pop} sequence with threshold. Finally, we calculate the variance of the d_{pop} sequence and compare it with a certain threshold th_{dist}. If it is greater than the threshold, output "continue search", otherwise output "terminate search". During the actual search process (inference stage), we can easily collect these distances as the feature for prediction.

V. Experiment

In this section, we first compare our method with state-of-the-art ANNS methods. Then we perform ablation studies to further analyze the contribution of each technique. For each dataset and method, we conduct the experiments multiple times to get stable and reliable results.

A. Experiment Setup

1) Hardware Configuration: We conduct all the experiments on a CPU server, which is equipped with four Intel(R) Xeon(R) CPU E5-4650 v4 (14 cores, 28 threads, running at 2.20 GHz), 128 GB DDR4 memory, and 3TB SSD. The operating system is the Red Hat 4.8.5-16.

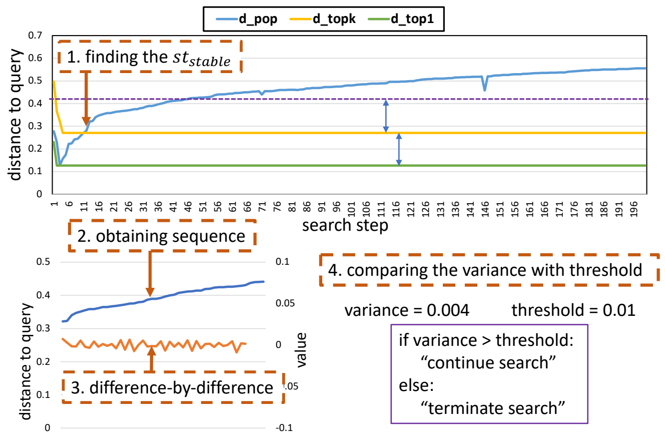

Fig. 13. The entire pipeline of prediction model. It consists of 4 steps: (1) finding the st_{stable}, (2) obtaining sequence, (3) difference-by-difference, and (4)comparing the variance with threshold.

TABLE II
Information of the experimental datasets, including the dimensions (D), the number of base vectors, the number of learning vectors, and the number of query vectors.

Dataset	D	Base vectors	Learning vectors	Query vectors
DEEP1M/10M	96	1,000,000/ 10,000,000	100,000	10,000
Turing1M/10M	100	1,000,000/ 10,000,000	-	100,000

2) Datasets: The datasets we used are shown in the Table. II, which are widely used in ANNS methods. The graph-based ANNS methods construct base data as a graph index and search the k nearest neighbors of query data on this graph index. The original dataset contains one billion data, we intercept the first one million/ten million data as the base vectors. For the DEEP dataset, we intercept several data from the rest as the learning vectors (used to train the prediction model).

- DEEP1M/10M [27]: DEEP dataset comes from a deep classification image model GoogLeNet, and each data is a 96-dimensional vector.
- Turing1M/10M [28]: Turing dataset is a new dataset being released by the Microsoft Turing team for the Billion-scale ANNS challenge competition [28]. It consists of Bing queries encoded by Turing AGI v5, and each data is a 100-dimensional vector.

3) Compared algorithms: For all algorithms, we adopted their code on Github and set the relevant parameters as they recommended. During the search process, we all employ a thread to compare the algorithms. During the build process, to save time, we build all indexes with all threads.

- **HNSW**[1] is a well-known graph-based algorithm based on a structure named as Hierarchical NSW graph.
- **NSG**[2] is based on a kNN graph, and then build a

[1]https://github.com/nmslib/hnswlib
[2]https://github.com/ZJULearning/nsg

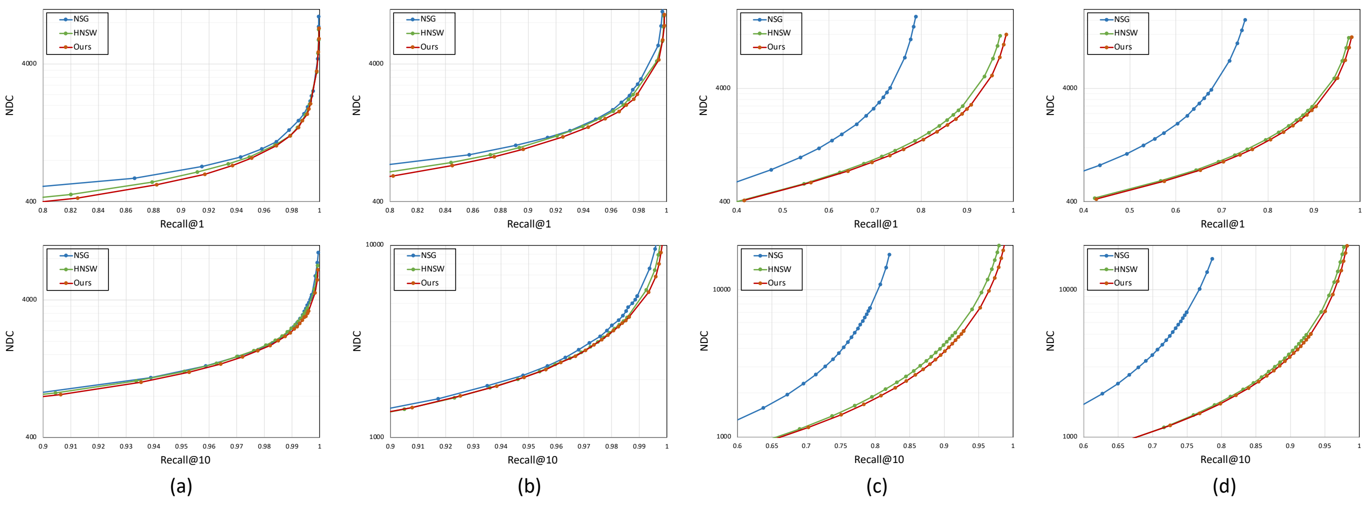

Fig. 14. Ours vs. HNSW and NSG on (a) DEEP1M, (b) DEEP10M, (c) Turing1M, and (d) Turing10M (down right is better).

spreading-out graph with a navigating node as the starting point.

B. Overall performance

We experiment with state-of-the-art algorithms (HNSW and NSG) and our method on multiple datasets that are widely used in ANNS algorithms. For NSG, we set the parameters for DEEP and Turing dataset according to its suggestion on Github (due to their similar dimensions). In the experiment, the parameters are set as follows: L = 40, R = 50, C = 500 for the all datasets. For HNSW, we selected the parameters with the best search performance for each dataset by grid search. In the experiment, the parameters are set as follows: M = 20, $maxM0$ = 40, efc = 300 for the all datasets.

Fig. 14 shows the overall performance comparison results between our method and current state-of-the-art algorithms. We compare the search performance at R@1 and R@10 on four datasets (DEEP1M, DEEP10M, Turing1M, and Turing10M), respectively. On the DEEP dataset (Fig. 14(a) and Fig. 14(b)), our method performs better than HNSW in the case of R@1. Our method almost matches the search performance of NSG and HNSW in the case of R@10. On the Turing dataset (Fig. 14(c) and Fig. 14(d)), our method is 1.21x and 2.71x faster than HNSW and NSG in search speed, respectively.

C. Ablation studies

1) Reverse connection enhancement: In the reverse connection enhancement strategy, there are two enhanced modes: from the random connection part of the candidate and the point where the insertion point cannot be reached. As shown in Fig. 15, we compare these two strategies with the baseline on the Turing1M dataset. We plot the speed up of the two strategies relative to the baseline search speed and the actual search speed of the baseline (the red curve).

We choose the results with recall rate of 0.7, 0.8, and 0.9 in R@1 and R@10 cases, respectively. The randomly enhanced strategy wants to achieve the same recall rate as the baseline, but the search speed is slower. In the case of R@1, our method can speed up to 5%, 9%, and 10% compared to the baseline, respectively. This indicates that our connection enhancement strategy can achieve better connectivity on graph index than random strategy.

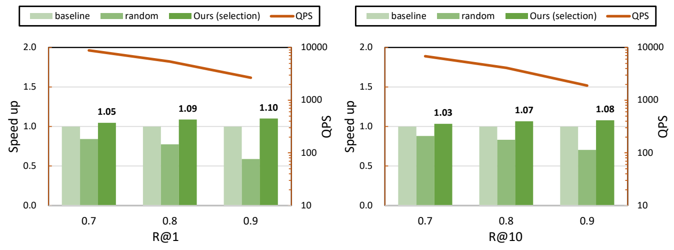

Fig. 15. Different reverse enhancements and relative acceleration of baseline, and the baseline search speed. When the recall rate is 0.9, our strategy can accelerate 10%.

2) Scalability for query-aware strategy: To demonstrate the scalability of our query-aware early termination strategy, we conduct experiments on both DEEP1M and DEEP10M datasets. For each dataset, we experiment with R@10 and R@100, respectively. As shown in Table. III, we separately list the NDC when HNSW and our method achieve a high recall rate on the DEEP dataset, and the proportion of search speed up.

We find that the greater the recall rate, the greater the proportion of search speed up. In particular, our strategy can speed up the search process by 1.29x when the recall rate R@10 = 0.999 on the DEEP1M dataset. And in other cases, there are varying degrees of search speed up.

VI. Conclusion

In this paper, we present a study of the graph-based approximate nearest neighbor search (ANNS) problem. We indicate two challenges in existing ANNS methods: poor

TABLE III
A COMPARISON OF THE NUMBER OF DISTANCE COMPUTATIONS (NDC) REQUIRED TO ADOPT THE QUERY-AWARE EARLY TERMINATION STRATEGY WITH THE BASELINE WHILE ACHIEVING THE SAME RECALL RATE.

Recall rate	DEEP1M-R@10			DEEP1M-R@100			DEEP10M-R@10			DEEP10M-R@100		
	dist. compution		speed up	dist. compution		speed up	dist. compution		speed up	dist. compution		speed up
	HNSW	Ours		HNSW	Ours		HNSW	Ours		HNSW	Ours	
0.99	2637.87	2475.86	1.07	4905.17	4611.04	1.06	5865.32	5368.36	1.09	10362.2	9709.18	1.07
0.995	3347.81	3067.41	1.09	6553.21	5965.97	1.10	7429.83	6659.18	1.12	14424.3	13233.8	1.09
0.999	7110.67	5500.62	**1.29**	12125.8	10328.2	1.17	15712.9	13204.6	1.19	27463.4	24713.9	1.11

connectivity and search redundancy. The poor connectivity is due to the existence of in-degree constraints of the points in the construction algorithm. Search redundancy is due to the fact that search algorithm (which are widely used) is not aware of the minimum search steps for the query. Therefore, in order to make the graph structure stronger, we propose a reverse connection enhancement strategy. We add some connections by judging whether the insertion point is reachable, so that the in-degree of the insertion point is no longer restricted. In order to make the search process smarter, we propose a query-aware early termination strategy. We reduce the search redundancy overhead by collecting dynamic distance features during the search process and assigning different search steps to each query. Finally, extensive experiments show that our method can improve the search speed by 1.06x - 1.29x.

REFERENCES

[1] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou, "Mobile recommender systems in tourism," *Journal of network and computer applications*, vol. 39, pp. 319–333, 2014.

[2] J. Suchal and P. Návrat, "Full text search engine as scalable k-nearest neighbor recommendation system," in *IFIP International Conference on Artificial Intelligence in Theory and Practice*, pp. 165–173, Springer, 2010.

[3] A. N. Papadopoulos and Y. Manolopoulos, *Nearest Neighbor Search:: A Database Perspective*. Springer Science & Business Media, 2006.

[4] Z. Lulu, G. Guohua, L. Kang, and H. Ajing, "Images matching algorithm based on surf and fast approximate nearest neighbor search," *Application Research of Computers*, vol. 30, no. 3, pp. 921–923, 2013.

[5] G. Shakhnarovich, T. Darrell, and P. Indyk, "Nearest-neighbor methods in learning and vision," in *Neural Information Processing*, 2005.

[6] M. E. Houle and M. Nett, "Rank-based similarity search: Reducing the dimensional dependence," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 1, pp. 136–150, 2014.

[7] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng, "Query-aware locality-sensitive hashing for approximate nearest neighbor search," *Proceedings of the VLDB Endowment*, vol. 9, no. 1, pp. 1–12, 2015.

[8] M. Douze, H. Jégou, and F. Perronnin, "Polysemous codes," in *European Conference on Computer Vision*, pp. 785–801, Springer, 2016.

[9] M. Zhang and Y. He, "Grip: Multi-store capacity-optimized high-performance nearest neighbor search for vector search engine," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1673–1682, 2019.

[10] A. Arora, S. Sinha, P. Kumar, and A. Bhattacharya, "Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces," *arXiv preprint arXiv:1804.06829*, 2018.

[11] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 11, pp. 2227–2240, 2014.

[12] J. Wang, N. Wang, Y. Jia, J. Li, G. Zeng, H. Zha, and X.-S. Hua, "Trinary-projection trees for approximate nearest neighbor search," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 2, pp. 388–403, 2013.

[13] 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 index," *Proceedings of the VLDB Endowment*, 2014.

[14] K. Terasawa and Y. Tanaka, "Spherical lsh for approximate nearest neighbor search on unit hypersphere," in *Workshop on Algorithms and Data Structures*, pp. 27–38, Springer, 2007.

[15] Y. Liu, J. Cui, Z. Huang, H. Li, and H. T. Shen, "Sk-lsh: an efficient index structure for approximate nearest neighbor search," *Proceedings of the VLDB Endowment*, vol. 7, no. 9, pp. 745–756, 2014.

[16] Y. Kalantidis and Y. Avrithis, "Locally optimized product quantization for approximate nearest neighbor search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2321–2328, 2014.

[17] A. M. Abdelhadi, C.-S. Bouganis, and G. A. Constantinides, "Accelerated approximate nearest neighbors search through hierarchical product quantization," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, pp. 90–98, IEEE, 2019.

[18] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[19] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems*, vol. 45, pp. 61–68, 2014.

[20] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," *Proceedings of the VLDB Endowment*, vol. 12, no. 5, pp. 461–474, 2019.

[21] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1475–1488, 2019.

[22] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *Proceedings of the 20th international conference on World wide web*, pp. 577–586, 2011.

[23] M. Aumüller, E. Bernhardsson, and A. Faithfull, "Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," in *International Conference on Similarity Search and Applications*, pp. 34–49, Springer, 2017.

[24] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *VLDB*, vol. 98, pp. 194–205, 1998.

[25] P. Wieschollek, O. Wang, A. Sorkine-Hornung, and H. Lensch, "Efficient large-scale approximate nearest neighbor search on the gpu," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2027–2035, 2016.

[26] Z. Song and N. Roussopoulos, "K-nearest neighbor search for moving query point," in *International Symposium on Spatial and Temporal Databases*, pp. 79–96, Springer, 2001.

[27] A. Babenko and V. Lempitsky, "Efficient indexing of billion-scale datasets of deep descriptors," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2055–2063, 2016.

[28] H. V. Simhadri *et al.*, "Billion-scale approximate nearest neighbor search challenge," 2021.

[29] M. A. Casey and M. Slaney, "Song intersection by approximate nearest neighbor search.," in *ISMIR*, vol. 6, pp. 144–149, 2006.

[30] B. Harwood and T. Drummond, "Fanng: Fast approximate nearest neighbour graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5713–5722, 2016.

[31] G. T. Toussaint, "The relative neighbourhood graph of a finite planar set," *Pattern recognition*, vol. 12, no. 4, pp. 261–268, 1980.