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, daiguohao1992@gmail.com, {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 methods have attracted the attention of many researchers due to their superiority compared to non-graph-based methods. Compared with traditional recommender systems, mobile recommender systems have higher latency requirements. 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 is poor, which in turn affects the search performance. (2) Redundant search. The existing search algorithm uses 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, 87.4% 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 state-of-the-art graph-based algorithm, our solutions can improve the search speed up to 1.21x when the recall rate equals 0.95.

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

I. Introduction

The Approximate Nearest Neighbor Search (ANNS) is widely used in many fields. Compared with the brute method, approximate methods reduce the search time by a few orders of magnitude with a slight accuracy loss [1]. ANNS meets the requirements of different scenarios by exploring the trade-off between accuracy and latency. The graph-based ANNS methods show great potential in recent years and achieve a high recall rate in a short search time [2]. The graph-based ANNS methods build an approximate nearest neighbor graph, and then use the graph as an index for greedy search [3].

Compared with traditional recommender systems, mobile recommender systems have higher latency requirements [4]. HNSW still faces the following challenges, which make it difficult to meet the requirements. The first challenge is that **the connectivity of the graph is poor**. 84.6% of the recall rate loss is caused by poor connectivity of the graph. Poor connectivity is mainly reflected in the existence of some points with small in-degree on the graph. 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 varies greatly and follows the long-tailed distribution. When applying the sufficiently long search steps, more than 40% of the queries only need 12.6% of the actual search step, revealing heavy redundant search overhead.

To tackle these challenges, we propose two optimization solutions to **make the graph structure stronger** and **make the search process 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 algorithm. Then we indicate that the existing construction algorithm restricts the in-degree of each point, which leads to the poor connectivity of the graph. 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. Our method can improve the search speed up to 1.21x when the recall rate equals 0.95.

II. 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

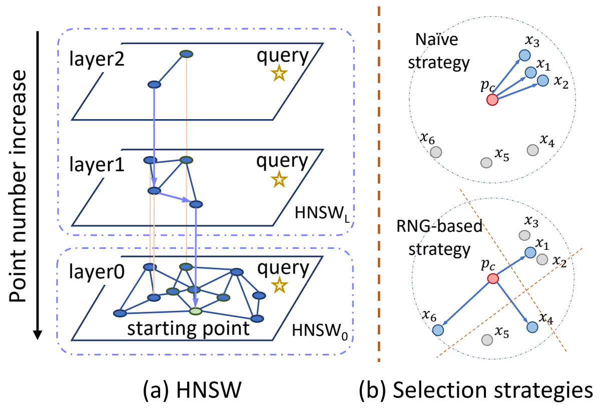


Fig. 1. (a) HNSW [6] is composed of multiple layers. The starting search point of HNSW$_0$ is determined by HNSW$_L$. (b) The intuitive comparison of neighbor selection strategies (naïve strategy and RNG-based strategy).

through edges. These points and the edges between them construct the graph. If there exists an edge from x_i to x_j in the graph, then 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 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 methods lies in the algorithms used in the construction process.

A. NSG

Fu et al. [5] 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.

B. HNSW

Hierarchical Navigable Small World graphs (HNSW) [6] 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. As shown in the Fig. 1(a), the HNSW is composed of multiple graph layers, and there are connections between points in the same graph layer (abbreviated it layer).

The construction process for each layer can be divided into three stages: the get candidate stage, the forward connection stage, and the reverse connection stage. The get candidate stage is to find some nearest points for insertion point p_c as neighbor candidates Q_{cand}^{in}. The forward connection stage select Q_{sel}^{in} as p_c neighbors through the neighbor selection strategies (as shown in Fig. 1(b)). The naïve strategy connects the nearest points. Another strategy based on Relative Neighborhood Graph (RNG) [7] selects a part of points from the candidates, and these points are as scattered as possible with respect to p_c. The RNG-based neighbor selection strategy (and its variants) is widely used in graph-based ANNS construction algorithms due to its high efficiency. In the reverse connection stage, for all neighbors of p_c, we use the same strategy to reconnect their neighbors.

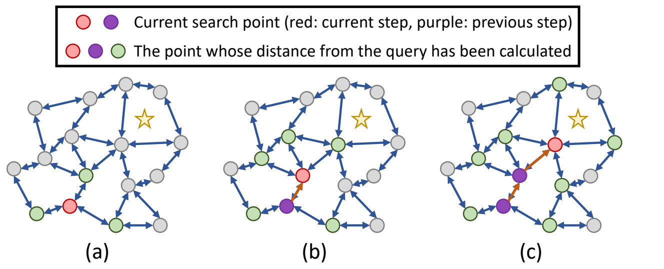

Current search point (red: current step, purple: previous step)
The point whose distance from the query has been calculated
(a)
(b)
(c)

Fig. 2. Schematic diagram of searching query nearest neighbors on a graph. (a)-(c) The first three steps of the search process.

During the search process, 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. 2. 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 the result queue 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 until the termination condition is met. Finally, the k points nearest to the query in the result queue Q_r are used as the search results.

III. Reverse Connection Enhancement

A. Motivation

In the graph-based ANNS methods, the connectivity of the graph is crucial to the final search results. If some points belong to the ground truth but are not found during the search process, this case will lead to a lower recall rate. Obviously, if the in-degree of a point is smaller, then it has a greater negative impact on the recall rate. We find the recall rate loss due to this situation accounts for 84.6% of the total recall rate loss. Then we indicate that the poor connectivity of graph is due to the restriction of the existing construction algorithm through degree analysis.

B. Degree Analysis

The construction process is to add the point (*insertion point*) in the base vectors X to the graph one by one. As shown in Fig. 3, at time t_c, the corresponding insertion point is x_c, the corresponding graph is G_c. The whole construction process can be divided into two parts: before t_c and after t_c. Then,

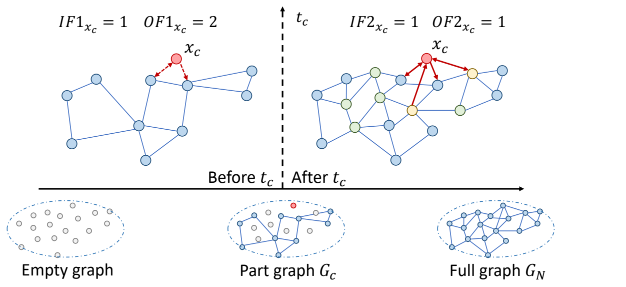


Fig. 3. The in-degree (out-degree) of point x_c is composed of two parts $IF1(OF1)$ and $IF2(OF2)$. $IF1(OF1)$ comes from the point (blue) in the graph before t_c, and $IF2(OF2)$ comes from the point (green) added to the graph after t_c.

the in-degree $IDG(x_c)$ and out-degree $ODG(x_c)$ of x_c are correspondingly expressed in two parts, as shown below:

$$IDG(x_c) = IF1_{x_c} + IF2_{x_c} \tag{1}$$

$$ODG(x_c) = OF1_{x_c} + OF2_{x_c} \tag{2}$$

- $OF1_{x_c}$: It is the out-degree of x_c before time t_c. To be specific, some nearest points of x_c found in the graph 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, 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 (after t_c in Fig. 3), 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 (after t_c in Fig. 3), 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. 3, $IF2_{x_c}$ and $OF2_{x_c}$ depend on whether 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. If $IF1$ of some points is small, then their final in-degree $IDG(\cdot)$ are also small. The connectivity of the point x_c in the full graph 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 [6], which also results in 80.5% of the points with $OF1$ less than 2/5 of $maxM0$. The existence of restrictive relation between $IF1_{x_c}$ and $OF1_{x_c}$ leads to poor connectivity of the graph G_N.

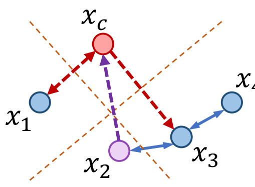


Fig. 4. Diagram of the reverse connection enhancement strategy. Before enhancement: we can only find the point x_c through point x_1. After enhancement: we can find the point x_c by either point x_1 or point x_2.

C. Our Method

In order to solve the above problem, we propose the reverse connection enhancement strategy. 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. 4 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 $OF1_{x_c}$, 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.

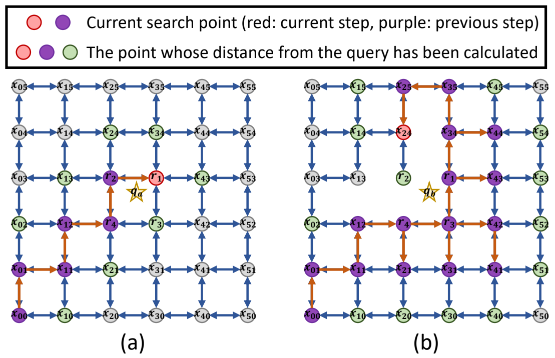
IV. Query Aware Early Termination

A. Motivation

The existing search algorithm incurs heavy redundant search overhead, which is widely used in most graph-based ANNS methods [5], [6]. 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, 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. For example, the average number of search steps for these queries is 309, but more than 40% of the queries have the minimum search steps of less than 39. For these queries, 87.4% of the search overhead is redundant (34.9% redundant overhead for the whole). Next, we will analyze the causes of this problem in detail and propose our solutions.



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

Fig. 5. Schematic diagram of searching query nearest neighbors on a graph. (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.

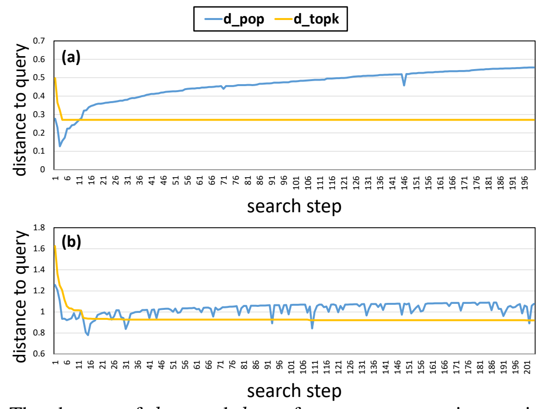


Fig. 6. The changes of d_{pop} and d_{topk} for two representative queries during the search process (on the DEEP1M dataset). (a) The query is located in a symmetric connection region, and its minimum search steps is 6. (b) The query is located in an asymmetric connection region, and its minimum search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

B. Analysis

The redundant search problem is caused by the region connection relationship in the graph. The region means a part of the graph near the query. According to the connection relationship, regions can be divided into two types: symmetric connection region (Fig. 5(a)) and asymmetric connection region (Fig. 5(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 in Fig. 5, 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.

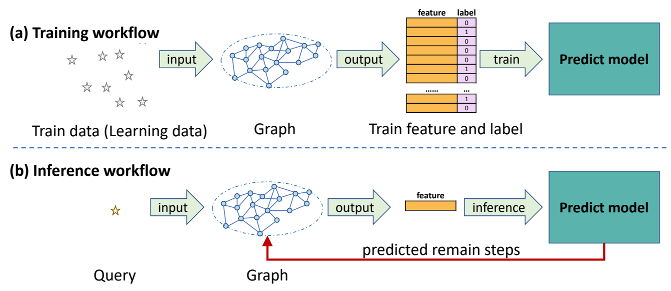


Fig. 7. Workflow for query-aware early termination strategy: (a) Training workflow and (b) Inference workflow.

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. We identify these two types of regions by the changes in d_{pop} and d_{topk} during the search process. For the symmetric connection region, the curve d_{pop} is smooth relative to the curve d_{topk}. For the asymmetric connection region, there will be some fluctuations in the curve d_{pop}.

We illustrate that further with Fig. 5. 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. 6(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. 6(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.

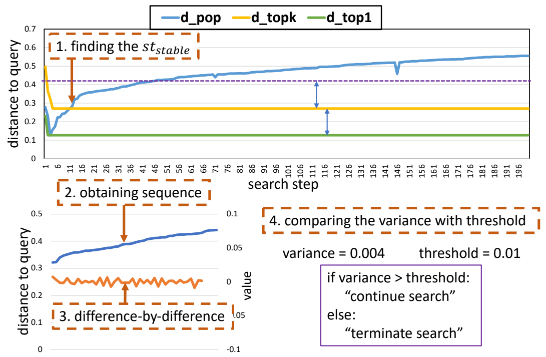


Fig. 8. The entire pipeline of prediction model. It consists of 4 steps: (1) finding the st_{stable}, (2) obtaining sequence, (3) difference-by-difference, and (4) comparing the variance with threshold.

For the complete workflow (as shown in Fig. 7), we do not need to add any additional structure to the graph. In the training stage (Fig. 7(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. In the inference stage (Fig. 7(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. As shown in Fig. 8, 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

A. Experiment Setup

1) Datasets: The datasets we used are shown in the Table. I, which are widely used in ANNS methods.

- DEEP1M/10M [8]: DEEP dataset comes from a deep classification image model GoogLeNet.
- Turing1M/10M [9]: Turing dataset consists of Bing queries encoded by Turing AGI v5.

2) 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 single thread to compare the algorithms.

- **HNSW** is a well-known graph-based algorithm based on a structure named Hierarchical NSW graph.
- **NSG** is based on a kNN graph, and builds a spreading-out graph with a navigating node as the starting point.

TABLE I
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

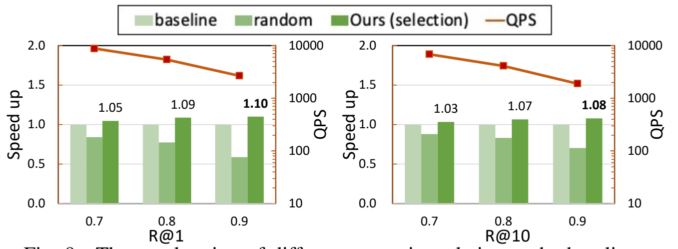


Fig. 9. The acceleration of different strategies relative to the baseline.

B. Overall performance

We experiment on state-of-the-art algorithms (HNSW and NSG) and our method on multiple datasets that are widely used in ANNS algorithms (as shown in Fig. 10). 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. 10(a) and Fig. 10(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. 10(c) and Fig. 10(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. 9, 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). 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 than the random strategy.

2) Scalability for query-aware strategy: To demonstrate the scalability of our query-aware early termination strategy. As shown in Table. II, we separately list the NDC when HNSW and our method achieve a high recall rate on the DEEP dataset,

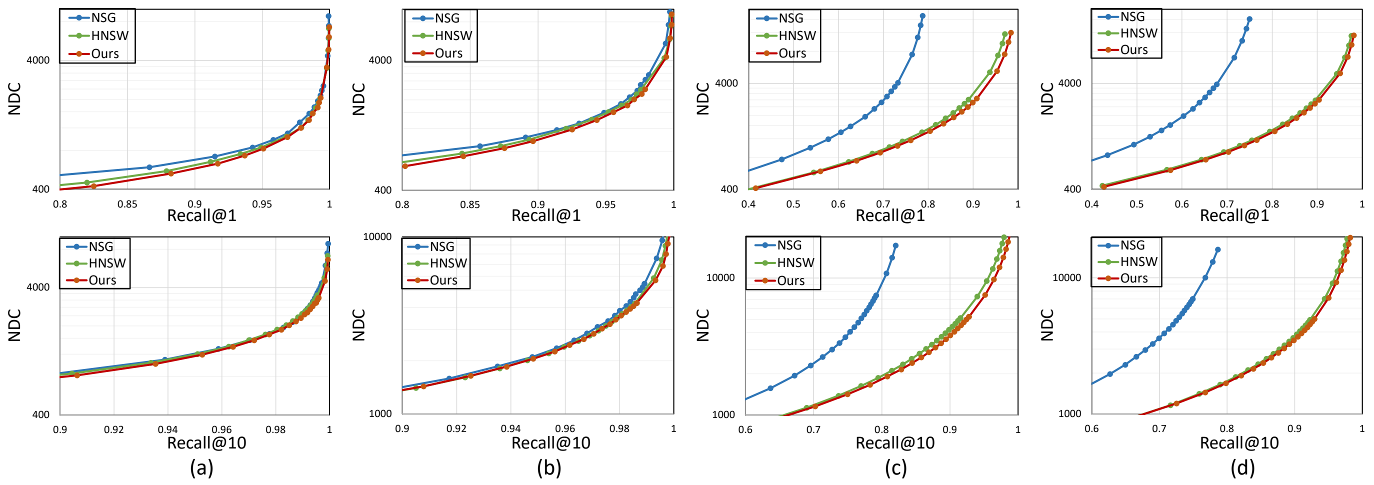


Fig. 10. Ours vs. HNSW and NSG on (a) DEEP1M, (b) DEEP10M, (c) Turing1M, and (d) Turing10M (down right is better).

TABLE II
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

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 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 up to 1.21x when the recall rate equals 0.95.

VII. Acknowledgement

This work was supported by National Natural Science Foundation of China (No. 62104128, U19B2019, 61832007); China Postdoctoral Science Foundation (No. 2019M660641); National Key R&D Program of China (No. 2017YFA02077600); Tsinghua EE Xilinx AI Research Fund; Beijing National Research Center for Information Science and Technology (BNRist); Beijing Innovation Center for Future Chips; Kuaishou Technology.

References

[1] 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.

[2] 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.

[3] M. Wang, X. Xu, Q. Yue, and Y. Wang, “A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search,” *arXiv preprint arXiv:2101.12631*, 2021.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] H. V. Simhadri *et al.*, “Billion-scale approximate nearest neighbor search challenge,” 2021.