A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search

Mengzhao Wang¹, Xiaoliang Xu¹, Qiang Yue¹, Yuxiang Wang¹,∗
¹Hangzhou Dianzi University, China
{mzwang,xxl,yq,lsswyx}@hdu.edu.cn

ABSTRACT
Approximate nearest neighbor search (ANNS) constitutes an important operation in a multitude of applications, including recommendation systems, information retrieval, and pattern recognition. In the past decade, graph-based ANNS algorithms have been the leading paradigm in this domain, with dozens of graph-based ANNS algorithms proposed. Such algorithms aim to provide effective, efficient solutions for retrieving the nearest neighbors for a given query. Nevertheless, these efforts focus on developing and optimizing algorithms with different approaches, so there is a real need for a comprehensive survey about the approaches’ relative performance, strengths, and pitfalls. Thus here we provide a thorough comparative analysis and experimental evaluation of 13 representative graph-based ANNS algorithms via a new taxonomy and fine-grained pipeline. We compared each algorithm in a uniform test environment on eight real-world datasets and 12 synthetic datasets with varying sizes and characteristics. Our study yields novel discoveries, offering several useful principles to improve algorithms, thus designing an optimized method that outperforms the state-of-the-art algorithms. This effort also helped us pinpoint algorithms’ working portions, along with rule-of-thumb recommendations about promising research directions and suitable algorithms for practitioners in different fields.

PVLDB Reference Format:
Mengzhao Wang, Xiaoliang Xu, Qiang Yue, Yuxiang Wang. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. PVLDB, 14(1): XXX-XXX, 2021. doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Lsyhprum/WEAVESS.

1 INTRODUCTION
Nearest Neighbor Search (NNS) is a fundamental building block in various application domains [7, 8, 38, 67, 70, 80, 108, 117], such as information retrieval [34, 118], pattern recognition [29, 57], data mining [44, 47], machine learning [24, 28], and recommendation systems [69, 82]. With the explosive growth of datasets’ scale and the inevitable curse of dimensionality, accurate NNS cannot meet actual requirements for efficiency and cost [61]. Thus, much of the literature has focused on efforts to research approximate NNS (ANNS) and find an algorithm that improves efficiency substantially while mildly relaxing accuracy constraints (an accuracy-versus-efficiency tradeoff [59]).

ANNS is a task that finds the approximate nearest neighbors among a high-dimensional dataset for a query via a well-designed index. According to the index adopted, the existing ANNS algorithms can be divided into four major types: hashing-based [40, 45]; tree-based [8, 86]; quantization-based [52, 74]; and graph-based [38, 67] algorithms. Recently, graph-based algorithms have emerged as a highly effective option for ANNS [6, 10, 41, 75]. Thanks to graph-based ANNS algorithms’ extraordinary ability to express neighbor relationships [38, 103], they only need to evaluate fewer points of dataset to receive more accurate results [38, 61, 67, 72, 112].

As Figure 1 shows, graph-based ANNS algorithms build a graph index (Figure 1(b)) on the original dataset (Figure 1(a)), the vertices in the graph correspond to the points of the original dataset, and neighboring vertices (marked as x, y) are associated with an edge by evaluating their distance δ(x, y), where δ is a distance function. In Figure 1(b), the four vertices (numbered 1–4) connected to the black vertex are its neighbors, and the black vertex can visit its neighbors along these edges. Given this graph index and a query q (the red star), ANNS aims to get a set of vertices that are close to q. We take the case of returning q’s nearest neighbor as an example to show ANNS’ general procedure: Initially, a seed vertex (the black vertex, it can be randomly sampled or obtained by additional approaches [47, 67]) is selected as the result vertex r, and we can conduct ANNS from this seed vertex. Specifically, if δ(n, q) < δ(r, q), where n is one of the neighbors of r, r will be replaced by n. We repeat this process until the termination condition (e.g., ∀n, δ(n, q) ≤ δ(r, q)) is met, and the final r (the green vertex) is q’s nearest neighbor. Compared with other index structures, graph-based algorithms are a proven superior tradeoff in terms of accuracy versus efficiency [10, 38, 61, 65, 67], which is probably why they enjoy widespread use among high-tech companies nowadays (e.g., Microsoft [98, 100], Alibaba [38, 112], and Yahoo [47, 48, 89]).

1.1 Motivation
The problem of graph-based ANNS on high-dimensional and large-scale data has been studied intensively across the literature [38]. Dozens of algorithms have been proposed to solve this problem from different optimizations [36, 37, 43, 54, 65, 67, 72]. For these
algorithms, existing surveys [10, 61, 85] provide some meaningful explorations. However, they are limited to a small subset about algorithms, datasets, and metrics, as well as studying algorithms from a macro perspective, and the analysis and evaluation of intra-algorithm components are ignored. For example, [61] includes a few graph-based algorithms (only three), [10] focuses on efficiency vs accuracy tradeoff, [85] only considers several classic graphs. This motivates us to carry out a thorough comparative analysis and experimental evaluation of existing graph-based algorithms via a new taxonomy and micro perspective (i.e., some fine-grained components). We detail the issues of existing work that ensued.

I1: Lack of a reasonable taxonomy and comparative analysis of inter-algorithms. Many studies in other fields show that an insightful taxonomy can serve as a guideline for promising research in this domain[99, 102, 105]. Thus, a reasonable taxonomy needs to be established, to point to the different directions of graph-based algorithms (§3). The index of existing graph-based ANNS algorithms are generally derivatives of four classic base graphs from different perspectives, i.e., Delaunay Graph (DG) [35], Relative Neighborhood Graph (RNG) [92], K-Nearest Neighbor Graph (KNNG) [75], and Minimum Spanning Tree (MST) [58]. Some representative ANNS algorithms, such as KGraph [31], HNSW [67], DPG [61], SPTAG [27] can be categorized into KNNG-based (KGraph and SPTAG) and RNG-based (DPG and HNSW) groups working off the base graphs upon which they rely. Under this classification, we can pinpoint differences between algorithms of the same category or different categories, to provide a comprehensive inter-algorithm analysis.

I2: Omission in analysis and evaluation for intra-algorithm fine-grained components. Many studies only compare and analyze graph-based ANNS algorithms from two coarse-grained components, i.e., construction and search [79, 85], which hinders insight into the key components. Construction and search, however, can be divided into many fine-grained components such as candidate neighbor acquisition, neighbor selection [37, 38], seed acquisition [9, 36], and routing [14, 94] (we discuss the details of these components in §4). Evaluating these fine-grained components (§5) led to some interesting phenomena. For example, some algorithms’ performance improvements are not so remarkable for their claimed major contribution (optimization on one component) in the paper, but instead by another small optimization for another component (e.g., NSSG [37]). Additionally, the key performance of completely different algorithms may be dominated by the same fine-grained component (e.g., the neighbor selection of NSG [38] and HNSW [67]). Such unusual but key discoveries occur by analyzing the components in detail to clarify which part of an algorithm mainly works in practice, thereby assisting researchers’ further optimization goals.

I3: Richer metrics are required for evaluating graph-based ANNS algorithms’ overall performance. Many evaluations of graph-based algorithms focus on the tradeoff of accuracy vs efficiency [43, 64, 65], which primarily reflects related algorithms’ search performance [59]. With the explosion of data scale and increasingly frequent requirements to update, the index construction efficiency and algorithm’s index size have received more and more attention [112]. Related metrics such as graph quality (it can be measured by the percentage of vertices that are linked to their nearest neighbor on the graph) [21], average out-degree, and so on indirectly affect the index construction efficiency and index size, so they are vital for comprehensive analysis of the index performance.

From our abundance of experiments (see §5 for details), we gain a novel discovery: higher graph quality does not necessarily achieve better search performance. For instance, HNSW [67] and DPG [61] yield similar search performances on the GIST1M dataset [1]. However, in terms of graph quality, HNSW (63.3%) is significantly lower than DPG (99.2%) (§5). Note that DPG spends a lot of time improving graph quality during index construction, but it is unnecessary; this is not uncommon, as we also see it in [31, 36–38].

I4: Diversified datasets are essential for graph-based ANNS algorithms’ scalability evaluation. Some graph-based ANNS algorithms are evaluated only on a small number of datasets, which limits analysis on how well they scale on different datasets. Looking at the evaluation results on various datasets (see §5 for details), we find that many algorithms have significant discrepancies in terms of performance on different datasets. That is, the advantages of an algorithm on some datasets may be difficult to extend to other datasets. For example, when the search accuracy reaches 0.99, NSG’s speedup is 125× more than that of HNSW for each query on Msong [2]. However, on Crawl [3], NSG’s speedup is 80× lower than that of HNSW when it achieves the same search accuracy of 0.99. This shows that an algorithm’s superiority is contingent on the dataset rather than being fixed in its performance. Evaluating and analyzing different scenarios’ datasets leads to understanding performance differences better for graph-based ANNS algorithms in diverse scenarios, which provides a basis for practitioners in different fields to choose the most suitable algorithm.

1.2 Our Contributions
Driven by the aforementioned issues, we provide a comprehensive comparative analysis and experimental evaluation of representative graph-based algorithms on carefully selected datasets of varying characteristics. It is worth noting that we try our best to reimplement all algorithms using the same design pattern, programming language and tricks, and experimental setup, which makes the comparison fairer. Our key contributions are summarized as follows.

(1) We provide a new taxonomy of the graph-based ANNS algorithms based on four base graphs. For I1, we classify graph-based algorithms based on four base graphs (§3), which brings a new perspective to understanding existing work. On this basis, we compare and analyze the features of inter-algorithms, make connections if different algorithms use similar techniques, and elaborate upon the inheritance and improvement of relevant algorithms, thus exhibiting diversified development roadmaps (Table 2 and Figure 3).

(2) We present a unified pipeline with seven fine-grained components for analyzing graph-based ANNS algorithms. As for I2, we break all graph-based ANNS algorithms down to seven fine-grained components in an unified pipeline (Figure 4): We divide construction into initialization, candidate neighbor acquisition, neighbor selection, connectivity, and seed preprocessing components, and divide search into seed acquisition and routing components (§4). This not only allows us to have a deeper understanding of the algorithm, but also to achieve a fair evaluation of a component by controlling other components’ consistency in the pipeline (§5).

(3) We conduct a comprehensive evaluation for representative graph-based ANNS algorithms with more metrics and...
ANNS techniques have been used in various multimedia applications. This paper mainly focuses on in-memory core algorithms. For a dataset where \(|\cdot|\) denotes the size of a set and \(\mathcal{R}\) denotes a graph (the index \(\mathcal{I}\) in Definition 2.2) constructed on \(S\), \(\forall v \in V\) that uniquely corresponds to a point \(x\) in \(S\). Here \(\forall (u, v) \in E\) represents the neighbor relationship between \(u\) and \(v\), and \(\forall q \in Q\). Given a query \(q\), seeds \(\mathcal{S}\), routing strategy, and termination condition, the graph-based ANNS initializes approximate \(k\) nearest neighbors \(\mathcal{R}\) of \(q\) with \(\mathcal{S}\), then conducts a search from \(\mathcal{S}\) and updates \(\mathcal{R}\) via a routing strategy. Finally, it returns the query result \(\mathcal{R}\) once the termination condition is met.

2 Overview of Graph-Based ANNS

2.1 Problem Definition

Before formally describing ANNS, we first define NNS.

Definition 2.1. NNS. Given a finite dataset \(S\) in Euclidean space \(\mathbb{R}^d\) and a query \(q\), NNS obtains \(k\) nearest neighbors \(\mathcal{R}\) of \(q\) by evaluating \(\delta(x, q)\) where \(x \in S\). \(\mathcal{R}\) is described as follows:

\[
\mathcal{R} = \arg\min_{\mathcal{R} \subseteq S, |\mathcal{R}| = k} \sum_{x \in \mathcal{R}} \delta(x, q).
\]

As the volume of data grows, \(|S|\) becomes exceedingly large (ranging from millions to billions in scale), which makes it impractical to perform NNS on large-scale data because of the high computational cost [116]. Instead of NNS, a large amount of practical techniques have been proposed for ANNS, which relaxes the guarantee of accuracy for efficiency by evaluating a small subset of \(S\) [101]. The ANNS problem is defined as follows:

Definition 2.2. ANNS. Given a finite dataset \(S\) in Euclidean space \(\mathbb{R}^d\), and a query \(q\), ANNS builds an index \(I\) on \(S\). It then gets a subset \(C\) of \(S\) by \(I\), and evaluates \(\delta(x, q)\) to obtain the approximate \(k\) nearest neighbors \(\mathcal{R}\) of \(q\), where \(x \in C\).

Generally, we use recall rate \(\text{Recall}@k = \frac{|\mathcal{R}\cap \mathcal{C}|}{k}\) to evaluate the search results’ accuracy. ANNS algorithms aim to maximize \(\text{Recall}@k\) while making \(C\) as small as possible (e.g., \(|C|\) is only a few thousand when \(|S|\) is millions on the SIFT1M [1] dataset). As mentioned earlier, ANNS algorithms based on graphs have risen in prominence because of their advantages in accuracy versus efficiency. We define graph-based ANNS as follows.

Definition 2.3. Graph-based ANNS. Given a finite dataset \(S\) in Euclidean space \(\mathbb{R}^d\), \(G(V, E)\) denotes a graph (the index \(I\) in Definition 2.2) constructed on \(S\), \(\forall v \in V\) that uniquely corresponds to a point \(x\) in \(S\). Here \(\forall (u, v) \in E\) represents the neighbor relationship between \(u\) and \(v\), and \(\forall q \in Q\). Given a query \(q\), seeds \(\mathcal{S}\), routing strategy, and termination condition, the graph-based ANNS initializes approximate \(k\) nearest neighbors \(\mathcal{R}\) of \(q\) with \(\mathcal{S}\), then conducts a search from \(\mathcal{S}\) and updates \(\mathcal{R}\) via a routing strategy. Finally, it returns the query result \(\mathcal{R}\) once the termination condition is met.
3.1 Base Graphs for ANNS

The four base graphs that graph-based ANNS algorithms depend on are the foundation for analyzing these algorithms. Next, we will give a formal description of each base graph, and visually show their differences through a toy example in Figure 2.

**Delaunay Graph (DG).** In Euclidean space \( E^d \), the DG \( G(V, E) \) constructed on dataset \( S \) satisfies the following conditions: For \( \forall e \in E \) (e.g., the yellow line in Figure 2(a)), where its corresponding two vertices are \( x, y \), there exists a circle (the red circle in Figure 2(a)) passing through \( x, y \), and no other vertices inside the circle, and there are at most three vertices (i.e., \( x, y, z \)) on the circle at the same time (see [35] for DG’s standard definition). DG ensures that the ANNS always return precise results [67], but the disadvantage is that DG is almost fully connected when the dimension \( d \) is extremely high, which leads to a large search space [38, 43].

**Relative Neighborhood Graph (RNG).** In Euclidean space \( E^d \), the RNG \( G(V, E) \) built on dataset \( S \) has the following property: For \( x, y \in V \), if \( x \) and \( y \) are connected by edge \( e \in E \), then \( \forall z \in V \), with \( \delta(x, y) < \delta(x, z) \), or \( \delta(x, y) < \delta(z, y) \). In other words, \( z \) is not in the red lune in Figure 2(b) (for RNG’s standard definition, refer to [92]). Compared with DG, RNG cuts off some redundant neighbors (close to each other) that violate its aforementioned property, and makes the remaining neighbors distribute omnidirectionally, thereby reducing ANNS’ distance calculations [67]. However, the time complexity of constructing RNG on \( S \) is \( O(|S|^3) \) [49].

**K-Nearest Neighbor Graph (KNNG).** Each point in dataset \( S \) is connected to its nearest \( K \) points to form a KNNG \( G(V, E) \) in Euclidean space \( E^d \). As Figure 2(c) \( (K = 2) \) shows, for \( x, y \in V \), \( x \in N(y) = \{x, u\} \), but \( y \notin N(x) = \{z, v\} \), where \( N(y), N(x) \) are the neighbor sets of \( y \) and \( x \), respectively. Therefore, the edge between \( y \) and \( x \) is a directed edge, so KNNG is a directed graph. KNNG limits the number of neighbors of each vertex to \( K \) at most, thus avoiding the surge of neighbors, which works well in scenarios with limited memory and high demand for efficiency. It can be seen that KNNG does not guarantee global connectivity in Figure 2(c), which is unfavorable for ANNS.

**Minimum Spanning Tree (MST).** In Euclidean space \( E^d \), MST is the \( G(V, E) \) with the smallest \( \sum_{e \in E} w(e) \) on dataset \( S \), where the two vertices associated with \( e \in E \) are \( x \) and \( y \), \( w(e) = \delta(x, y) \). If \( \exists e_i, e_j \in E \), \( w(e_i) = w(e_j) \), then MST is not unique [68]. Although MST has not been adopted by most current graph-based ANNS algorithms, HCNNG [72] confirms MST’s effectiveness as a neighbor selection strategy for ANNS. The main advantage for using MST as a base graph relies on the fact that MST uses the least edges to ensure the graph’s global connectivity, so that keeping vertices with low degrees and any two vertices are reachable. However, because of a lack of shortcuts, it may detour when searching on MST [38, 65]. For example, in Figure 2(d), when search goes from \( s \) to \( r \), it must detour with \( s \to x \to y \to u \to r \). This can be avoided if there is an edge between \( s \) and \( r \).

### 3.2 Graph-Based ANNS Algorithms

Although the formal definition of base graphs facilitates theoretical analysis, it is impractical for them to be applied directly to ANNS [38]. Obviously, their high construction complexity is difficult to scale to large-scale datasets. This has become even truer with the advent of frequently updated databases [60]. In addition, it is difficult for base graphs to achieve high search efficiency in high-dimensional scenarios [38, 43, 77]. Thus, a number of graph-based ANNS algorithms tackle improving the base graphs from one or several aspects. Next, we outline 13 representative graph-based ANNS algorithms (A1–A13) based on the aforementioned four base graphs and their development roadmaps (Figure 3). Table 2 summarizes some important properties about algorithms.

**DG-based and RNG-based ANNS algorithms (NSW, HNSW, FANNG, NGT).** To address the high degree of DG in high dimension, some slight improvements have been proposed [15, 16, 56]. However, they rely heavily on DG’s quality and exist the curse of dimensionality [65]. Therefore, some algorithms add an RNG approximation on DG to diversify the distribution of neighbors [67].

**A1: Navigable Small World graph (NSW).** NSW [65] constructs an undirected graph through continuous insertion of elements and ensures global connectivity (approximate DG). The intuition is that the result of a greedy traversal (random seeds) is always the nearest neighbor on DG [67]. The long edges formed in the beginning of construction have small-world navigation performance to ensure search efficiency, and the vertices inserted later form short-range edges, which ensure search accuracy. NSW also achieved excellent results in the maximum inner product search [63, 71]. However, according to the evaluation of [77], NSW provides limited best tradeoffs between efficiency and effectiveness compared to non-graph-based indexes, because its search complexity is poly-logarithmic [73]. In addition, NSW uses undirected edges to connect vertices, which results in vertices in dense areas acting as the “traffic hubs” (high out-degrees), thus damaging search efficiency.

**A2: Hierarchical Navigable Small World graphs (HNSW).** An improvement direction is put forth by [19, 66] to overcome NSW’s poly-logarithmic search complexity. Motivated by this, HNSW [67] generates a hierarchical graph and fixes the upper bound of each vertex’s number of neighbors, thereby allowing a logarithmic complexity scaling of search. Its basic idea is to separate neighbors to different levels according to the distance scale, and the search is an iterative process from top to bottom. For an inserted point, HNSW not only selects its nearest neighbors (approximate DG), but also considers the distribution of neighbors (approximate RNG). HNSW has been deployed in various applications [17, 22, 57] because of its
Table 2: Summary of important representative graph-based ANNS algorithms

| Algorithm          | Base Graph | Edge      | Build Complexity       | Search Complexity |
|--------------------|------------|-----------|------------------------|-------------------|
| KGraph [31]        | KNNG       | directed  | O(|S|^3)                | O(|S|^3)          |
| NGT [46]           | KNNG+DG+RNG| directed  | O(|S|^3)                | O(|S|^3)          |
| SPTAG [27]         | KNNG+RNG   | directed  | O(|S| log(|S|)+|P|)    | O(|S|^2)          |
| NGW [65]           | DG         | undirected| O(|S| log(|S|)+|P|)    | O(|S|^2)          |
| IEH [54]           | KNNG       | directed  | O(|S|^3 log(|S|)+|P|)  | O(|S|^2)          |
| FANN [43]          | RNG        | directed  | O(|S|^2 log(|S|))      | O(|S|^2)          |
| HNSW [67]          | DG+RNG     | directed  | O(|S| log(|S|))        | O(|S|^2)          |
| EFANNA [36]        | KNNG       | directed  | O(|S|^3)                | O(|S|^3)          |
| DPG [61]           | KNNG+RNG   | undirected| O(|S|^3+|S|^2)         | O(|S|^2)          |
| NG [38]            | KNNG+RNG   | directed  | O(|S|^3+|S|^2)         | O(|S|^2)          |
| HCNNG [72]         | MST        | directed  | O(|S| log(|S|))        | O(|S|^2)          |
| Vamana [88]        | RNG        | directed  | O(|S|^2)                | O(|S|^2)          |
| NSSG [37]          | KNNG+RNG   | directed  | O(|S|+|S|^2)           | O(|S|^2)          |

‡ c, t are the constants.
† Complexity is not informed by the authors; we derive it based on the related papers' descriptions and experimental estimates. See Appendix D for details.

Figure 3: Roadmaps of graph-based ANNS algorithms. The arrows from a base graph (green shading) to an algorithm (gray shading) and from one algorithm to another indicate the dependence and development relationships.

KNNG-based ANNS algorithms (SPTAG, KGraph, EFANNA, IEH). A naive construction for KNNG is exhaustively comparing all pairs of points, which is prohibitively slow and unsuitable for large dataset S. Some early solutions construct an additional index (such as tree [76] or hash [93, 111]), and then find the neighbors of each point through ANNS. However, such methods generally suffer from high index construction complexity [26]. This is because they ignore this fact: the queries belong to S in the graph construction process, but the queries of ANNS generally do not [100]. Thus, it is unnecessary to ensure a good result is given for general queries on additional index [100]. There are two types of representative solutions, which only focus on graph construction.

A3: Space Partition Tree and Graph (SPTAG). One is based on divide and conquer, and its representative is SPTAG [27], a library released by Microsoft. SPTAG hierarchically divides dataset S into subsets (through Trinary-Projection Trees [101]) and builds an exact KNNG over each subset. This process repeats multiple times to produce a more accurate KNNG on S. Moreover, SPTAG further improves KNNG’s accuracy by performing neighborhood propagation [100]. The early version of SPTAG added multiple KD-trees on S to iteratively obtain the seeds closer to the query [98]. However, on extremely high-dimensional S, the KD-trees will produce an inaccurate distance bound estimation. In response, the balanced k-means trees are constructed to replace the KD-trees [27]. Inspired by the universal superiority brought about by RNG, SPTAG has recently added the option of approximating RNG in the project [27].

A6: KGraph. The other is based on NN-Descent [32]; its basic idea is neighbors are more likely to be neighbors of each other [36]. KGraph [31] first adopts this idea to reduce KNNG’s construction complexity to O(|S|1.14) on dataset S. It achieves better search performance than NSW [18]. Therefore, some NN-Descent-based derivatives are developed to explore its potential [23, 114, 115].

A7: EFANNA and A8: IEH. Instead of random initialization during construction (such as KGraph), Extremely Fast Approximate Nearest Neighbor Search Algorithm (EFANNA) [36] first builds multiple KD-trees on S and better initializes the neighbors of each vertex through ANNS on these KD-trees, then executes NN-Descent. At the search stage, EFANNA also uses these KD-trees to obtain seeds that are closer to the query. The idea of initializing seeds through additional structures is inspired by Iterative Expanding Hashing (IEH) [54], which uses hash buckets to obtain better seeds. However, IEH’s KNNG is constructed by brute force in [54].
A9: Diversified Proximity Graph (DPG). To overcome the aforementioned issue, DPG [61] practices optimization to control neighbors’ distribution on KGraph. It sets the threshold of the angle between the neighbors of a vertex to make the neighbors evenly distributed in all directions of the vertex. This is only an approximate implementation of RNG from another aspect (see Appendix C for the proof). In addition, to deal with $S$ with a large number of clusters, DPG keeps bi-directed edges on the graph.

A10: Navigating Spreading-out Graph (NSG). Although DPG’s search performance is comparable to HNSW, it suffers from a large index [38]. To settle this problem and further improve search performance, NSG [38] proposes an edge selection strategy based on monotonic RNG (called MRNG), which is actually equivalent to HNSW’s (see Appendix A for the proof). Its construction framework is inspired by DPG; that is, to prune edges on KNNG. NSG ensures high construction efficiency by executing ANNS on KGraph to obtain candidate neighbors. NSG has been integrated into Alibaba’s Taobao e-commerce platform to combine superior index construction and search performance [38], and its billion-scale implementation version exceeds the current best FAISS [53].

A11: Navigating Satellite System Graph (NSSG). NSSG continues to explore the potential of pruning edges on KNNG, and proposes an edge selection strategy based on SSG [37]. When obtaining a vertex’s candidate neighbors, instead of conducting the ANNS like NSG, it gets the neighbors and neighbors’ neighbors of the vertex on KNNG, which significantly improves construction efficiency. Both SSG and MRNG are approximations to RNG, but SSG is relatively relaxed when cutting redundant neighbors. Therefore, NSSG has a larger out-degree. Although [37] believes that SSG is more beneficial to ANNS than MRNG, we reach the opposite conclusion through a fairer evaluation (see §5.4 for details).

A12: Yamana. Microsoft recently proposed Yamana [88] to combine with solid-state drives (SSD) for billions of data. It analyzes the construction details of HNSW and NSG to extract and combine the better parts. Its construction framework is motivated by NSG. Instead of using KGraph to initialize like NSG, Yamana initializes randomly. When selecting neighbors, Yamana improves the HNSW’s strategy by adding a parameter $\alpha$ to increase the edge selection’s flexibility and executing two passes with different $\alpha$. Experiments show that its result graph has a shorter average path length when searching, which works well with SSD.

MST-based ANNS algorithms (HCNNG).

A13: HCNNG. Different from the aforementioned techniques, a recent method called Hierarchical Clustering-based Nearest Neighbor Graph (HCNNG) [72] uses MST to connect the points on dataset $S$. It uses the same divide-and-conquer framework as SPTAG. The difference is that HCNNG divides $S$ through multiple hierarchical clusters, and all points in each cluster are connected through MST. HCNNG uses multiple global KD-trees to get seeds (like SPTAG and EFANNA). Then to improve search efficiency, rather than using traditional greedy search, it performs an efficient guided search.

4 COMPONENTS’ ANALYSIS

Despite the diversity of graph-based ANNS algorithms, they all follow a unified processing pipeline. As Figure 4 shows, an algorithm can be divided into two coarse-grained components: index construction (top) and search (bottom), which are adopted by most of the current work to analyze algorithms [43, 61, 65, 72]. Recent research has endeavored to take a deeper look at some fine-grained components [37, 38], prompting them to find out which part of an algorithm plays a core role and then propose better algorithms. Motivated by this, we subdivide the index construction and search into seven fine-grained components (C1–C7 in Figure 4), and compare all 13 graph-based algorithms discussed in this paper by them.

4.1 Components for Index Construction

The purpose of index construction is to organize the dataset $S$ with a graph. Existing algorithms are generally divided into three strategies: Divide-and-conquer [96], Refinement [32], and Increment [42] (see Appendix E). As Figure 4 (top) show, an algorithm’s index construction can be divided into five detailed components (C1–C5). Among them, initialization can be divided into three ways according to different construction strategies.

C1: Initialization.

Overview. The initialization of Divide-and-conquer is dataset division; it is conducted recursively to generate many subgraphs so that the index is obtained by subgraph merging [26, 84]. For Refinement, in the initialization, it performs neighbor initialization to get the initialized graph, then refines the initialized graph to achieve better search performance [36, 38]. While the Increment inserts points continuously, the new incoming point is regarded as a query, then it executes ANNS to obtain the query’s neighbors on the subgraph constructed by the previously inserted points [65, 67]; it therefore implements seed acquisition during initialization.

Definition 4.1. Dataset Division. Given dataset $S$, the dataset division divides $S$ into $m$ small subsets—i.e., $S_0, S_1, \ldots, S_{m-1}$, and $S_0 \cup S_1 \cdots \cup S_{m-1} = S$.

Data division. This is a unique initialization of the Divide-and-conquer strategy. SPTAG previously adopts a random division scheme, which generates the principal directions over points randomly sampled from $S$, then performs random divisions to make each subset’s diameter small enough [95, 100]. To achieve better division, SPTAG turns to TP-tree [101], in which a partition hyperplane is formed by a linear combination of a few coordinate axes with weights being -1 or 1. HCNNG divides $S$ by iteratively performing hierarchical clustering. Specifically, it randomly takes two points from the set to be divided each time, and performs division by calculating the distance between other points and the two [72].

Definition 4.2. Neighbor Initialization. Given dataset $S$, for $\forall p \in S$, the neighbor initialization gets the subset $C$ from $S \setminus \{p\}$, and initializes $N(p)$ with $C$.

Neighbor initialization. Only the initialization of the Refinement strategy requires this implementation. Both KGraph and Yamana implement this process by randomly selecting neighbors [31, 88]. This method offers high efficiency but the initial graph quality is too low. The solution is to initialize neighbors through ANNS based on hash-based [93] or tree-based [36] approaches. EFANNA deploys the latter; it establishes multiple KD-trees on $S$. Then, each point is treated as a query, and get its neighbors through ANNS on multiple KD-trees [36]. This approach relies heavily on extra index and increases the cost of index construction. Thus, NSG, DPG, and NSSG deploy the NN-Descent [32]; they first randomly select neighbors for each point, and then update each point’s neighbors.
with neighborhood propagation. Finally, they get a high-quality initial graph by a small number of iterations. Specially, FANNG and IEH initialize neighbors via linear scan.

**Definition 4.3. Seed Acquisition.** Given the index \( G(V, E) \), the seed acquisition acquires a small subset \( \hat{S} \) from \( V \) as the seed set, and ANNS on \( G \) starts from \( \hat{S} \).

**Seed acquisition.** The seed acquisition of the index construction is **Increment** strategy’s initialization. The other two strategies may also include this process when acquiring candidate neighbors, and this process also is necessary for all graph-based algorithms in the search. For index construction, both NSW and NGT obtain seeds randomly [46, 65], while HNSW makes its seed points fixed from the top layer because of its unique hierarchical structure [67].

**Definition 4.4. Candidate Neighbor Acquisition.** Given a finite dataset \( S \), point \( p \in S \), the candidate neighbor acquisition gets a subset \( C \) from \( S \setminus \{ p \} \) as \( p \)’s candidate neighbors, and \( p \) get its neighbors \( N(p) \) from \( C \)—that is, \( N(p) \subset C \).

**C2: Candidate neighbor acquisition.** The graph constructed by the **Divide-and-conquer** generally produce candidate neighbors from a small subset obtained after dataset division. For a subset \( S_i \subset S \) and a point \( p \in S_i \), SPTAG and HCNNG directly take \( S_i \setminus \{ p \} \) as candidate neighbors [72, 100]. Although \( |S| \) may be large, the \( |S_i| \) obtained by the division is generally small. However, **Refinement** and **Increment** do not involve the process of dataset division, which leads to low index construction efficiency for IEH and FANNG to adopt the naive method of obtaining candidate neighbors [43, 54]. To solve this problem, NGT, NSW, HNSW, NSG, and Vanama all obtain candidate neighbors through ANNS. For a point \( p \in S \), the graph \( G_{\text{sub}} \) (**Increment**), formed by the previously inserted points or the initialized graph \( G_{\text{init}} \) (**Refinement**), they consider \( p \) as a query and execute ANNS on \( G_{\text{sub}} \) or \( G_{\text{init}} \), and finally return the query result as candidate neighbors of \( p \). This method only needs to access a small subset of \( S \). However, according to the analysis of [100], obtaining candidate neighbors through ANNS is overlapped, because the query is in \( S \) for index construction, but the ANNS query generally does not belong to \( S \). In contrast, KGraph, EFANNA, and NISSG use the neighbors of \( p \) and neighbors’ neighbors on \( G_{\text{init}} \) as its candidate neighbors [37], which improves index-construction efficiency. DPG directly uses the neighbors of \( p \) on \( G_{\text{init}} \) as candidate neighbors, but to obtain enough candidate neighbors, it generally requires \( G_{\text{init}} \) with a larger out-degree [61].

**Definition 4.5. Neighbor Selection.** Given a point \( p \) and its candidate neighbors \( C \), the neighbor selection obtains a subset of \( C \) to update \( N(p) \).

**C3: Neighbor selection.** The current graph-based ANNS algorithms mainly consider two factors for this component: distance and space distribution. Given \( p \in S \), the distance factor ensures that the selected neighbors are as close as possible to \( p \), while the space distribution factor makes the neighbors distribute as evenly as possible in all directions of \( p \). NSW, SPTAG\(^1\), NGT\(^2\), KGraph, EFANNA, and IEH only consider the distance factor and aim to build a high-quality graph index [32, 100]. HNSW\(^3\), FANNG, SPTAG\(^3\), and NSG\(^2\) consider the space distribution factor by evaluating the distance between neighbors, formally, for \( x \in C \), \( \forall y \in N(p) \), iff \( \delta(x, y) > \delta(y, p) \), \( x \) will join \( N(p) \) [38, 43]. To select neighbors more flexibly, Vamana adds the parameter \( \alpha \) so that for \( x \in C \), \( \forall y \in N(p) \), iff \( \alpha \cdot \delta(x, y) > \delta(y, p) \), \( \alpha \geq 1 \), \( x \) will be added to \( N(p) \) [88], so it can control the distribution of neighbors well by adjusting \( \alpha \).

**C4: Seed preprocessing.** Different algorithms may exist with different execution sequences between this component and the connectivity, such as NSW [65], NSG [38]. Generally, graph-based ANNS algorithms implement this component in a static or dynamic manner. For the static method, typical representatives are HNSW, NSG, Vanama, and NISSG. HNSW fixes the top vertices as the seeds, NSG and Vanama use the approximate centroid of \( S \) as the seed, and the seeds of NISSG are randomly selected vertices. While for the dynamic method, a common practice is to attach other indexes (i.e., for each query, the seeds close to the query are obtained through an additional index), SPTAG, EFANNA, HCNNG, and NGT build additional trees, such as KD-tree [27, 36], balanced k-means tree [27], and VP-tree [46]. IEH prepares for seed acquisition through hashing [54]. Then [33] compresses the original vector by OPQ [39] to obtain the seeds by quickly calculating the compressed vector. Random seed acquisition is adopted by KGraph, FANNG, NSW, and DPG, and they don’t need to implement seed preprocessing.

**C5: Connectivity. Incremental** strategy internally ensures connectivity (e.g., NSW). **Refinement** generally attaches depth-first

---

\(^1\)This refers to its original version—NGT-pang for NGT and SPTAG-KDT for SPTAG.

\(^2\)Although [38] distinguishes the neighbor selection of HNSW and NSG, we prove the equivalence of the two in Appendix A.

\(^3\)This refers to its optimized version—NGT-ong for NGT and SPTAG-BKT for SPTAG.
traversal to achieve this [38] (e.g., NSG). **Divide-and-conquer** generally ensures connectivity by multiply performing **dataset division** and subgraph construction (e.g., SPTAG).

### 4.2 Components for Search

We subdivide the search into two fine-grained components (C6–C7): seed acquisition and routing.

**C6: Seed acquisition.** Because the seed has a significant impact on search, this component of the search process is more concerned than the initialization of **Incremental** strategy. Some early algorithms obtain the seeds randomly, while state-of-the-art algorithms commonly use **seed preprocessing**. If the fixed seeds are produced in the preprocessing stage, it can be loaded directly at this component. If other index structures are constructed in the preprocessing stage, ANNS returns the seeds with the additional structure.

**Definition 4.6. Routing.** Given $G(V, E)$, query $q$, seed set $S$, the routing starts from the vertices in $S$, and then converges to $q$ by neighbor propagation along the neighbor $n$ of the visited point with smaller $\delta(n, q)$, until the vertex $r$ so that $\delta(r, q)$ reaches a minimum.

**Definition 4.7. Best First Search.** Given $G(V, E)$, query $q$, and vertices to be visited $C$, its maximum size is $c$ and the result set $R$. We initialize $C$ and $R$ with seed set $S$. For $x = \arg\min_{x \in C} \delta(x, q)$, best first search access $N(x)$, then $C \setminus \{x\}$ and $C \cup N(x)$. To keep $|C| = c$, $y = \arg\max_{y \in C} \delta(y, q)$ will be deleted. $\forall n \in N(x)$, if $\delta(n, q) < \delta(x, q)$, then $C \setminus \{x\}$ and $R \cup \{n\}$. The aforementioned process is performed iteratively until $R$ is no longer updated. (see Appendix F for the pseudocode)

**C7: Routing.** Almost all graph-based ANNS algorithms are based on a greedy routing strategy, including best first search (BFS) and its variants. NSW, HNSW, KGraph, IEH, EFANNA, DPG, NSG, NSSG, and Vanama use the original BFS to perform routing. Despite this method being convenient for deployment, it has two shortcomings: susceptibility to local optimum (S1) [14] and low routing efficiency (S2) [72]. S1 destroys the search results’ accuracy. For this problem, FANNNG adds backtracking to BFS, which slightly improves the search accuracy while significantly increasing the search time [43]. NGT alleviates S1 by adding a parameter $\epsilon$. On the basis of Definition 4.7, it cancels the size restriction on $C$ and takes $\delta(y, q)$ as the search radius $r$, for $\forall n \in N(x)$, if $\delta(n, q) < (1 + \epsilon) \cdot r$, then $n$ is added to $C$. Setting $\epsilon$ to a larger value can alleviate S1, but it will also significantly increase the search time [48]. SPTAG solves S1 by iteratively executing BFS. When a certain iteration falls into a local optimum, it will restart the search by selecting new seeds from the KD-tree [98]. HCNNG proposes using guided search to alleviate S2 rather than visiting all $N(x)$ like BFS, so guided search avoids some redundant visits based on the query’s location. Recently, some of the literature uses learning methods to perform routing [14, 59, 94]. These methods usually alleviate S1 and S2 simultaneously, but the adverse effect is that this requires extra training, and additional information also increases the memory overhead (see §5.5).

## 5 EXPERIMENTAL EVALUATION

This section presents an abundant experimental study of both individual algorithms (§3) and components (§4) extracted from the algorithms for graph-based ANNS. Because of space constraints, some of our experimental content is provided in appendix. Our evaluation seeks to answer the following question:

**Q1:** How do the algorithms perform in different scenarios? (§3.2–5.3)

**Q2:** Can an algorithm have the best index construction and search performance at the same time? (§5.2–5.3)

**Q3:** For an algorithm with the best overall performance, is the performance of each fine-grained component also the best? (§5.4)

**Q4:** How do machine learning-based optimizations affect the performance of the graph-based algorithms? (§5.5)

**Q5:** How can we design a better graph-based algorithm based on the experimental observations and verify its performance? (§6)

### 5.1 Experimental Setting

**Datasets.** Our experiment involves eight real-world datasets popularly deployed by existing works, which cover various applications such as video (UQ-V [5]), audio (Msong [2]), text (Crawl [3], GloVe [51], Enron [81]), and image (SIFT1M [1]). Their main characteristics are summarized in Table 3. # Base is the number of elements in the base dataset. LID indicates local intrinsic dimensionality, and a larger LID value implies a “harder” dataset [61]. Additionally, 12 synthetic datasets are used to test each algorithm’s scalability to different datasets’ performance (e.g., dimensionality, cardinality, number of clusters, and standard deviation of the distribution in each cluster [85]). Out of space considerations, please see the scalability evaluation in Appendix J.

All datasets in the experiment are processed into the base dataset, query dataset, and ground-truth dataset.

**Compared algorithms.** Our experiment evaluates 13 representative graph-based ANNS algorithms mentioned in §3, which are carefully selected from research literature and practical projects. The main attributes and experimental parameters of these algorithms are introduced in Appendix E and Appendix F.

**Evaluation metrics.** To measure the algorithm’s overall performance, we employ various metrics related to index construction and search. For index construction, we evaluate the index construction efficiency and size. Some index characteristics such as graph quality, average out-degree, and the number of connected components are recorded; they indirectly affect index construction efficiency and size. Given a proximity graph $G'(V', E')$ (graph index of an algorithm) and the exact graph $G(V, E)$ on the same dataset, we define graph quality of an index as $\frac{|E'|}{|E|}$ [21, 26, 97]. For search, we evaluate search efficiency, accuracy, and memory overhead. Search efficiency can be measured by queries per second (QPS) and speedup. QPS is the ratio of the number of queries ($q$) to the search time ($t$); i.e., $\frac{q}{t}$ [38]. Speedup is defined as $\frac{1}{|S|}$, where $|S|$ is the dataset’s size and is also the number of distance calculations of the linear scan for a query, and NDC is the number of distance calculations of an algorithm for a query (equal to $|C|$ in Definition 2.2) [72]. We use the recall rate to evaluate the search accuracy, which is
defined as $|R/T|$, where $R$ is an algorithm’s query result set, $T$ is the real result set, and $|R| = |T|$. We also measure other indicators that indirectly reflect search performance, such as the candidate set size during the search and the average query path length.

### Implementation setup
We reimplement all algorithms by C++; they were removed by all the SIMD, pre-fetching instructions, and other hardware-specific optimizations. To improve construction efficiency, the parts involving vector calculation are parallelized for index construction of each algorithm [11, 90]. All C++ source codes are compiled by g++ 7.3, and MATLAB source codes (only for index construction of a hash table in IEH [54]) are compiled by MATLAB 9.9. All experiments are conducted on a Linux server with a Intel(R) Xeon(R) Gold 5218 CPU at 2.30GHz, and a 125G memory.

### Parameters
Because parameters’ adjustment in the entire base dataset may cause overfitting [38], we randomly sample a certain percentage of data points from the base dataset to form a validation dataset. We search for the optimal value of all the adjustable parameters of each algorithm on each validation dataset, to make the algorithms’ search performance reach the optimal level. Note that high recall areas’ search performance primarily is concerned with the needs of real scenarios.

#### 5.2 Index Construction Evaluation
We build indexes of all compared algorithms in 32 threads on each real-world dataset. Note that we construct each algorithm with the parameters under optimal search performance.

### Construction efficiency
The construction efficiency is mainly affected by the construction strategy, algorithm category, and dataset. In Figure 5, the KNNG-based algorithms (e.g., KGraph and EFANNA) constructed by NN-Descent have the smallest construction time among all test algorithms, while the KNNG-based algorithms constructed by divide and conquer (e.g., SPTAG) or brute force (e.g., IEH) have higher construction time. The construction time of RNG-based algorithms vary greatly according to the initial graph. For example, when adding the approximation of RNG on KGraph (e.g., DPG and NSSG), it has a high construction efficiency. However, RNG approximation based on the KNNG built by brute force (e.g., FANNG) has miniscule construction efficiency (close to IEH). Note that Vamana is an exception; its ranking on different datasets has large differences. This is most likely attributable to its neighbor selection parameter $\alpha$ heavily dependent on dataset. The construction time of DG-based algorithms (e.g., NGT and NSW) shows obvious differences with datasets. On some hard datasets (e.g., GloVe), their construction time is even higher than FANNG.

### Index size and average out-degree
The index size mainly depends on the average out-degree (AD). Generally, the smaller the AD, the smaller the index size. As Figure 6 and Table 4 show, RNG-based algorithms (e.g., NSG) have a smaller index size, which is mainly because they cut redundant edges (the lower AD) during RNG approximation. KNNG-, DG-, and MST-based algorithms (e.g., KGraph, NSW, and HCNNG) connect all nearby neighbors without pruning superfluous neighbors, so they always have a larger index size. Additional index structures (e.g., the tree in NGT) will also increase related algorithms’ index size.

### Graph quality
The algorithm category and dataset are the main factors that determine graph quality (GQ). In Table 4, the GQ of KNNG-based algorithms (e.g., KGraph) outperform other categories. The approximation to RNG prunes some of the nearest neighbors, thereby destroying RNG-based algorithms’ GQ (e.g., NSG). However, this phenomenon does not happen with DPG, mostly because it undirects all edges. Interestingly, DG- and MST-based algorithms’ GQ (e.g., NSW and HCNNG) shows obvious differences with datasets; on simple datasets (e.g., Audio), they have higher GQ, but it degrades on hard datasets (e.g., GIST1M).

### Connectivity
Connectivity mainly relates to the construction strategy and dataset. Table 4 shows that DG- and MST-based algorithms have good connectivity. The former is attributed to the Increment construction strategy (e.g., NSW and NGT), and the latter benefits from its approximation to MST. Some RNG-based algorithms perform depth-first search (DFS) to ensure connectivity (e.g., NSG and NSSG). DPG adds reverse edges to make it have good connectivity. Unsurprisingly, KNNG-based algorithms generally have a lot of connected components, especially on hard datasets.

#### 5.3 Search Performance
All searches are evaluated on a single thread. The number of nearest neighbors recalled is uniformly set to 10 for each query, and Recall@10 represents the corresponding recall rate. Because of space constraints, we only list the representative results in Figure 7 and 8, and the others are displayed in Appendix O. Note that our observations are based on the results on all datasets.
Accuracy and efficiency. As illustrated in Figure 7 and 8, the search performance of different algorithms on the same dataset or the same algorithm on different datasets have large differences. Generally, algorithms capable of obtaining higher speedup also can achieve higher QPS, which demonstrates that the search efficiency of graph-based ANNS algorithms mainly depends on the number of distance evaluations during the search [113]. The search performance of RING- and MST-based algorithms (e.g., NSG and HCNNG) generally beats other categories by a large margin, especially on hard datasets (e.g., GloVe). KNNG- and DG-based algorithms (e.g., EFANNA and NSW) can only achieve better search performance on simple datasets, their performance drops sharply on hard datasets. Particularly, the search performance of SPTAG decreases dramatically with the increase of LID. This is most likely because it frequently regains entry through the tree during the search [98], we know that the tree has bad curse of dimensionality [61].

Candidate set size (CS). There is a connection between the CS and algorithm category, dataset, and search performance. For most algorithms, we can set CS to obtain the target recall rate, but a few algorithms (e.g., SPTAG) reach the “ceiling” before the set recall rate. At this time, the recall rate hardly changes when we increase CS (i.e., a CS value with "+" in Table 5). The elements in a candidate set generally are placed in the cache because of frequent access during the search; so we must constrain the CS to a small value as much as possible because of the capacity’s limitation. Especially in the GPU, the CS will have a greater impact on the search performance [113].

In Table 5, DG-based and most RING-based algorithms (e.g., NCT and NSG) require a smaller CS. The CS of KNNG- and MST-based algorithms is related to the dataset, and the harder the dataset, the larger the CS (e.g., SPTAG). In general, algorithms with bad search performance have a larger CS (e.g., FANNG).

Memory overhead (MO). As Table 5 show, RING-based algorithms generally have the smallest memory overhead (e.g., NSG and NSSG). Some algorithms with additional index structures have high memory overhead (e.g., SPTAG and IEH). Larger AD and CS values also will increase the algorithms’ memory overhead (e.g., NSW and SPTAG-BKT). Overall, the smaller the algorithm’s index size, the smaller the memory overhead during search.

### 5.4 Components’ Evaluation

In this subsection, we evaluate representative components of graph-based algorithms on two real-world datasets with different difficulty. According to the aforementioned experiments, algorithms based on the **Refinement** construction strategy generally have better comprehensive performance. Therefore, we design a unified evaluation framework based on this strategy and the pipeline in Figure 4. Each component in the evaluation framework is set for a certain implementation to form a benchmark algorithm (see Appendix K for detailed settings). We use the `C # algorithm name` to indicate the corresponding component’s specific implementation. For example, **C1_NS** indicates that we use the initialization (C1) of NSG, i.e., the initial graph is constructed through NN-Descent.

Note that many algorithms have the same implementation for the same component (e.g., **C3_NS**, **C3_HNSW**, and **C3_FANNG**). We randomly select an algorithm to represent this implementation (e.g., **C3_HNSW**). The impact of different components on search performance and construction time are depicted in Figure 10 and Appendix M, respectively.

C1: Initialization. Figure 10(a) reports the impact of different graph index initialization methods on search performance. The search performance of **C1_NS** is much better than **C1_EFANNA** and **C1_KGraph**; and although **C1_NS** needs more construction time, it is worthwhile for such a large performance improvement. Moreover, a larger gap exists between **C1_NS** and the other two on GIST1M (harder), which shows that it has better scalability.

C2: Candidate neighbor acquisition. As shown in Figure 10(b), different candidate neighbor acquisition methods vary slightly. **C2_NS** has the best search performance, especially on GIST1M, with the price being more construction time. **C2_NS** usually has better search performance than **C2_DPG** under a similar construction time. It is worth noting that although DPG’s search performance on SIFT1M is better than HNSW’s in Figure 7, the search performance of **C2_HNSW** (i.e., **C2_NS**) exceeds that of **C2_DPG**.
Table 6: Index processing time (IPT) and memory consumption (MC) of ML1-based optimization.

| Algs. | Iq-V | Msong | Audio | SIFT1M | GIST1M | Crawl | Glove | Enron |
|-------|------|-------|-------|-------|--------|-------|-------|-------|
| NSSG  | 0.37 | 0.68  |       |       |        |       |       |       |
| NSSG+ML | 23.8 | 38.7  |       |       |        |       |       |       |

Figure 9: Speedup vs Recall@10 of graph-based ANNS algorithms under ML1-based optimizations.

Table 5: The candidate set size (CS), query path length (PL), and peak memory overhead (MO) during the search (the bold values are the best).

| Method       | C3_NSG | C3_NSSG | C3_HNSW | C3_DPG | C4_NSSG | C4_HCNNG | C4_NSG |
|--------------|--------|---------|---------|--------|---------|----------|--------|
| MC(GB)       | 1.00   | 0.95    | 0.85    | 0.85   | 0.85    | 0.85     | 0.85   |
| Speedup      | 0.00   | 0.00    | 0.00    | 0.00   | 0.00    | 0.00     | 0.00   |

C4: Seed preprocessing and C6: Seed acquisition. The C4 and C6 components are interrelated in all compared algorithms; that is, after specifying C4, C6 is also determined. Briefly, we use C4_NSSG to indicate C6_NSSG. As Figure 10(d) shows, the extra index structure to get the entry significantly impacts search performance. C4_NGT and C4_SPTAG-BKT have the worst search performance; they both obtain entry by performing distance calculations on an additional tree (we know that the tree index has a serious curse of dimensionality). Although C4_HCNNG also obtains entry through a tree, it only needs value comparison and no distance calculation on the KD-Tree, so it shows better search performance than C4_NGT and C4_SPTAG-BKT. C4_IEH adds the hash table to obtain entry, yielding better search performance. This may be because the hash can obtain entry close to the query more quickly than the tree. Meanwhile, C4_NSSG and C4_NSG still achieve high search performance without additional index. Note that there is no significant difference in index construction time for these methods.

C5: Connectivity. Figure 10(e) shows the algorithm with guaranteed connectivity has better search performance (e.g., C5_NSSG) than that without connectivity assurance (e.g., C5_Vamana). C7: Routing. Figure 10(f) shows different routing strategies’ impact on search performance. C7_NSW’s search performance is the best, and it is used by most algorithms (e.g., HNSW and NSG). C7_NGT has a precision “ceiling” because of the ε parameter’s limitation, which can be alleviated by increasing ε, but search efficiency will decrease. C7_FANNG can achieve high accuracy through backtracking, but backtracking also limits search efficiency. C7_HCNG avoids some redundant calculations based on the query position, however, this negatively affects search accuracy.
5.5 Machine Learning-Based Optimizations

Recently, machine learning (ML)-based methods are proposed to improve the speedup vs recall trade-off of the algorithms [14, 59, 78]. In general, they can be viewed as some optimizations on graph-based algorithms discussed above (such as NSG and NSW). We evaluate three ML-based optimizations on NSG and HNSW, i.e., ML1 [14], ML2 [59], and ML3 [78]. Because of space limitations, we only show the test results on ML1 in Table 6 and Figure 9, others share similar feature (see Appendix R for more details).

Analysis. ML-based optimizations generally obtain better speedup vs recall tradeoff at the expense of more time and memory. For example, the original NSG takes 55s and maximum memory consumption of 0.37 GB for index construction on SIFT100K; however, NSG optimized by ML1 takes 67,315s to process the index (even if we use the GPU for speedup), and the memory consumption is up to 23.8 GB. In summary, current ML-based optimizations have high hardware requirements and time cost, so their wide application is limited. Considering that most of the original graph-based algorithms can return query results in < 5ms, some high-tech companies (such as Alibaba, Facebook, and ZILLIZ) only deploy NSG without ML-based optimizations in real business scenarios [38, 55, 70].

6 DISCUSSION

According to the behaviors of algorithms and components on real-world and synthetic datasets, we discuss our findings as follows.

Recommendations. In Table 7, our evaluation selects algorithms based on best performance under different scenarios. NSG and NSSG have the smallest construction time and index size, so they are suitable for S1. KGraph, EFANNA, and DPG achieve the highest graph quality with lower construction time, so they are recommended for S2. For S3 (such as SSD [88]), DPG and HCNNNG are the best choices because their smaller average path length can reduce I/O times. On hard datasets (S4, large LID), HNSW, NSG, and HCNNNG show competitive search performance, while on simple datasets (S5), DPG, NSG, HCNNNG, and NSSG offer better search performance. For S6, we need a smaller candidate set size because of the cache’s limitation [113]; for now, NGT appears more advantageous. NSG and NSSG offer the smallest out-degree and memory overhead, so they are the best option for S7.

Guidelines. Intuitively, a practical graph-based ANNS algorithm should have: (H1) high construction efficiency; (H2) high routing efficiency; (H3) high search accuracy; and (L4) low memory overhead. For H1, we should not spend too much time improving graph quality, because the best graph quality is not necessary to achieve the best search performance. For H2, we should control the appropriate out-degree, diversify neighbors’ distribution (such as C4_HNSW), and reduce the cost of obtaining entries (like C4_IEH), to navigate quickly to the query’s nearest neighbors with a small number of distance calculations. In addition, we should avoid redundant distance calculations by optimizing the routing strategy (such as C7_HCNNNG) in the routing process. In terms of H3, to improve the search’s immunity from falling into the local optimum [14], we should reasonably design the distribution of neighbors during index construction, ensure connectivity (such as C5_NSNG), and optimize the routing strategy [94]. For L4, we can start by reducing the out-degree and candidate set size, and this can be achieved by improving the neighbor selection strategy (such as C3_NSNG) and routing strategy (like C7_HCNNNG).

Improvement. Based on our observations and Guidelines, we design an optimized algorithm that addresses H1, H2, H3, and L4 simultaneously. In the index construction phase, it initializes a graph with appropriate quality by NN-Descent (C1), quickly obtains candidate neighbors with C2_NSNG (C2), uses C3_NSNG to trim redundant neighbors (C3), randomly selects a certain number of entries (C4), and ensures connectivity through depth-first traversal (C5); in the search phase, it starts from the random entries (C6), and performs two-stage routing through C7_HCNNNG and C7_NSW in turn. As shown in Figure 11, the optimized algorithm surpasses the
state-of-the-art algorithms in terms of efficiency vs accuracy trade-off, while ensuring high construction efficiency and low memory overhead (see Appendix P for more details).

**Tendencies.** Over the last decade, graph-based ANNS algorithms have ranged from simple approximation of four classic base graphs (e.g., KGraph and NSW) to ANNS’s optimization (e.g., HNSW and NSG). Along the way, their performance—especially their search performance—has improved qualitatively. It is worth noting that almost all state-of-the-art algorithms are based on RNG (e.g., HNSW and NSG), and thus many approaches add an approximation of RNG on the basis of KNNG- or DG-based algorithms (see Figure 3). The RNG-based category is still a promising research direction for graph-based ANNS. The MST-based algorithm recently applied to graph-based ANNS, and it also achieves excellent results in our evaluation, especially on hard datasets. On the basis of the core algorithm discussed in this paper, researchers are refining and improving graph-based ANNS algorithms’ performance via hardware [53, 88, 113]. Other literatures add quantitative or distributed schemes to cope with data increases [30, 35]. To meet hybrid query requirements, the latest research adds structured attribute constraints to the search process of graph-based algorithms [104, 106].

**Challenges.** At present, almost all graph-based algorithms are oriented to raw data, which is the main reason why these algorithms have high memory usage. Determining how to organically combine data encoding or other methods with graph-based ANNS algorithms is a problem worth exploring. Compared with tree, hashing, and quantization, the graph-based algorithms have the highest index construction time [61], which adds difficulty with updating the graph index in real time. Also, figuring how to combine GPU acceleration or other methods with the graph-based ANNS algorithm to realize the real-time update of the graph index is worthy of an in-depth study. For data with different characteristics, the graph-based algorithms have different adaptability, and thus exhibit different performance levels. Finally, a major outstanding challenge is discerning how to adaptively select the optimal graph-based algorithm according to the dataset’s characteristics by learning.

7 CONCLUSIONS

In this paper, we consider 13 representative graph-based ANNS algorithms from a new taxonomy. We then divide all the algorithms into seven components for in-depth analysis. Next, we comprehensively evaluate and discuss all the algorithms’ performance on eight real-world datasets and 12 synthetic datasets. We also fairly evaluate each algorithm’s important components through a unified framework. In some ways, this work validates many previous empirical conclusions while leading to novel discoveries that will aid future researchers and practitioners. We also provide some rule-of-thumb recommendations about promising research directions and insightful principles to optimize algorithms.

Finally, we want to note that because of various constraints, our study only investigates core algorithms based on the main memory. Going forward, we will consider hardware (e.g., SSD and GPU) and machine learning optimizations, deploy distributed implementations, and add structured attribute constraints to ANNS.

ACKNOWLEDGMENTS

The National Key Research & Development Program (Number 2017YFC0820503) and National Natural Science Foundation of China (Number 62072149) supported this work. Additional funding was provided by the Primary Research & Development Plan of Zhejiang Province (Number 2021C03156 and 2021C02004), and the Public Welfare Research Program of Zhejiang (Number LGG19F020017).

REFERENCES

[1] Anon. 2010. Datasets for approximate nearest neighbor search. Retrieved October 05, 2020 from http://corpus-texmex.irisa.fr/

[2] Anon. 2011. Million Song Dataset Benchmarks. Retrieved April 15, 2020 from http://www.isf.tuwien.ac.at/mir/msd/

[3] Anon. unknown. Common Crawl. Retrieved April 15, 2020 from http://commoncrawl.org/

[4] Anon. unknown. TIMIT Audio. Retrieved April 15, 2020 from https://www.cs.princeton.edu/cass/demos.htm

[5] Anon. unknown. UQ-Fedio. Retrieved October 05, 2019 from http://staff.itee.uq.edu.au/hibenj/ADVVIDEO/

[6] Kazuo Aoyama, Atsunori Ogawa, Takaaki Hattori, Takaaki Hori, and Atsushi Nakamura. 2013. Graph index based query-by-example search on a large speech data set. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 8520–8524.

[7] Kazuo Aoyama, Kazumi Saito, Hiroshi Sawada, and Naonori Ueda. 2011. Fast approximate similarity search based on degree-reduced neighborhood graphs. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, 1052–1063.

[8] Akih Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. 2018. HD-Index: Pushing the Scalability-Accuracy Boundary for Approximate kNN Search in High-Dimensional Spaces. Proc. VLDB Endow. 11, 8 (2018), 906–919.

[9] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. Journal of the ACM (JACM) 45, 6 (1998), 891–923.

[10] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2017. ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In International Conference on Similarity Search and Applications. Springer, 34–49.

[11] Martin Aumüller and Matteo Ceccharello. 2019. Benchmarking Nearest Neighbor Search: Influence of Local Intrinsic Dimensionality and Result Diversity in Real-World Datasets. In EDM@ SDM. 14–23.

[12] Franz Aurenhammer. 1991. Voronoi diagrams—a survey of a fundamental geometric data structure. ACM Computing Surveys (CSUR) 23, 3 (1991), 345–405.

[13] Dmitry Baranchuk and Artem Babenko. 2019. Towards Similarity Graphs Constructed by Deep Reinforcement Learning. arXiv preprint arXiv:1911.12122 (2019).

[14] Dmitry Baranchuk, Dmitry Persiyanov, Anton Sinitsin, and Artem Babenko. 2019. Learning to Route in Similarity Graphs. In Proceedings of the 36th International Conference on Machine Learning, Vol. 97. PMLR, 475–484.

[15] Olivier Beaumont, Anne-Marie Kermarrec, Loris Marchal, and Étienne Rivière. 2007. VoroNet: A scalable object network based on Voronoi tessellations. In 2007 IEEE International Parallel and Distributed Processing Symposium. IEEE, 1–10.

[16] Olivier Beaumont, Anne-Marie Kermarrec, and Étienne Rivière. 2007. Peer-to-peer multidimensional overlays: Approximating complex structures. In International Conference On Principles Of Distributed Systems. Springer, 315–328.

[17] Callista Bee, Yuan-Jyue Chen, David Ward, Xiaomeng Liu, Georg Seelig, Karin Strauss, and Luis H Cze. 2020. Content-Based Similarity Search in Large-Scale DNA Data Storage Systems. bioRxiv (2020).

[18] Erik Bernhardsson. 2017. Benchmarking nearest neighbors. Retrieved September 13, 2020 from https://github.com/erikbern/ann-benchmarks

[19] Marian Boguna, Dmitri Krioukov, and Kimberly C Claffy. 2009. Navigability of complex networks. Nature Physics 5, 1 (2009), 74–80.

[20] Prosenjit Bose, Vida Dujmović, Ferran Hurtado, John Iacono, Stefan Langerman, Henk Meijer, Vera Sacristán, Maria Saumell, and David R Wood. 2012. Proximity graphs: E, Δ, L, ω and co. International Journal of Computational Geometry & Applications 22, 05 (2012), 439–469.

[21] Antoine Bossut, Anne-Marie Kermarrec, Nupur Mittal, and François Taiani. 2016. Being prepared in a sparse world: the case of KNN graph construction. In 2016 IEEE 32nd International Conference on Data Engineering (ICDE). IEEE, 241–252.

[22] Leonid Boytsov, David Novak, Yury Malkov, and Eric Nyberg. 2016. Off the beaten path: Let’s replace term-based retrieval with k-nn search. In Proceedings of the 25th ACM international on conference on information and knowledge management, 1099–1108.
[23] Brankica Bratik, Michael E. Houle, Vladimir Kurbalija, Vincent Oria, and MiloŠ Radovanović. 2018. NN-Descent on high-dimensional data. In Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics. 1–8.

[24] Yuan Cao, Heng Qi, Wenrui Zhou, Jian Liu, and Jie Gui. 2017. Binary hashing for approximate nearest neighbor search on big data: A survey. IEEE Access 6 (2017), 2039–2054.

[25] Edgar Chávez, Gonzalo Navarro, Ricardo Barzola-Yates, and José Luis Marroquin. 2001. Searching in metric spaces. ACM computing surveys (CSUR) 33, 3 (2001), 273–321.

[26] Jie Chen, Haw-ren Fang, and Yousaf Saad. 2009. Fast Approximate kNN Graph Construction for High Dimensional Data via Recursive Lanczos Bisection. Journal of Machine Learning Research 10, 9 (2009).

[27] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Limtang Zhao, and Jingdong Wang. 2018. SPATAG: A library for fast approximate nearest neighbor search. https://github.com/Microsoft/SPATAG

[28] Scott Cost and Steven Salzberg. 1993. A weighted nearest neighbor algorithm for learning with symbolic features. Machine learning 10, 1 (1993), 57–78.

[29] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. IEEE transactions on information theory 13, 1 (1967), 21–27.

[30] Shiyuan Deng, Xiao Yan, Ng Kw Kelvin, Chenjie Yang, and Jiang Cheng. 2019. Pyramid: A General Framework for Distributed Similarity Search on Large-scale Datasets. In 2019 IEEE International Conference on Big Data (Big Data). IEEE, 1066–1071.

[31] Wei Dong. 2011. KGraph: A Library for Approximate Nearest Neighbor Search. Retrieved July 12, 2020 from https://github.com/aaalgo/kggraph

[32] Wei Dong, Chankar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In Proceedings of the 20th international conference on World wide web. 577–586.

[33] Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. 2018. Link and retrieving multi-dimensional data. Journal of Intelligent Information Systems 13, 9 (2020), 1483–1497.

[34] Steven Fortune. 1995. Voronoi diagrams and Delaunay triangulations. In Computing in Euclidean geometry. World Scientific, 225–265.

[35] Conglong Li, Minjia Zhang, David G Andersen, and Yuxiong He. 2020. Improving Maximum Inner Product Search With the Navigating Spreading-out Graph. IEEE Transactions on Pattern Analysis and Machine Intelligence 42, 10 (2020), 2268–2285.

[36] Jie Li, Haifeng Liu, Chuanghua Gui, Jianyu Chen, Zhenying Ni, Ning Wang, and Yuan Chen. 2018. The Design and Implementation of a Real Time Visual Search System on JD e-commerce Platform. In Proceedings of the 19th International Middleware Conference Industry. 9–16.

[37] Libin Liu, Chuanhui Guo, Jianyu Chen, and Ning Wang. 2016. Optimizing Approximate Nearest Neighbor Search using Hierarchical Navigable Small World graphs and their relatives. Proc. VLDB Endow. 9, 8 (2016), 1069–1080.

[38] Javier Vargas Munoz, Marcos A Gonçalves, Zanoni Dias, and Ricardo da S Torres. 2019, 1 (2015), 1–12.

[39] Masajiro Iwasaki. 2016. Pruned bi-directed k-nearest neighbor graph for proximity search. In International Conference on Similarity Search and Applications. Springer, 20–33.

[40] Masajiro Iwasaki and Daisuke Miyazaki. 2018. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. arXiv preprint arXiv:1810.07355 (2018).

[41] Jerry W Jaromczyk and Miroslaw Kowaluk. 1991. Constructing the relative neighborhood graph in 3-dimensional Euclidean space. Discrete Applied Mathematics 31, 2 (1991), 181–187.

[42] Jerry W Jaromczyk and Godfried T Toussaint. 1992. Relative neighborhood graphs and their relatives. Proc. IEEE 80, 9 (1992), 1502–1517.

[43] Pennington Jeffrey, Socher Richard, and Manning Christopher. 2015. GloVe: Global Vectors for Word Representation. Retrieved April 15, 2020 from http://nlp.stanford.edu/projects/glove/

[44] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence 33, 1 (2011), 177–189.

[45] Masajiro Iwasaki and Daisuke Miyazaki. 2018. Optimization of indexing based on knn graph. arXiv preprint arXiv:1609.07228 (2016).

[46] Peng-Cheng Lin and Wan-Lei Zhao. 2019. Graph based Nearest Neighbor Search: Promises and Failures. arXiv preprint arXiv:1904.02077 (2019).

[47] Stanislav Morozov and Artem Babenko. 2018. Non-metric similarity graphs for nearest neighbor search. In 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. 3646–3654.

[48] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2019 (2019). Query by image and video content: The QBIC system. computer, 28, 9 (1995), 23–32.

[49] Jerzy W Jaromczyk and Godfried T Toussaint. 1992. Relative neighborhood graphs and their relatives. Proc. IEEE 80, 9 (1992), 1502–1517.

[50] Masajiro Iwasaki and Daisuke Miyazaki. 2018. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. arXiv preprint arXiv:1810.07355 (2018).
Given any point \( p \) in \( \mathbb{R}^d \), we define the k-nearest neighbor graph for \( p \) as the graph \( G(p) = (V(p), E(p)) \) where \( V(p) = \{ p \} \) and \( E(p) \) is the set of edges connecting \( p \) to its closest \( k \) neighbors in \( \mathbb{R}^d \).

APPENDIX

Appendix A. Proof for the equivalence of the neighbor selection strategies of HNSW and NGF

Notations. Given any point \( p \) on dataset \( S \), the candidate neighbor set of \( p \) obtained before neighbor selection is marked as \( C \) (see Definition 4.4 for the definition of candidate neighbor acquisition), and the neighbor selection (Definition 4.5) gets the final neighbor set \( N(p) \) from \( C \). Hence, \( B(p, \epsilon) = \{ x \in \mathbb{R}^d : ||x - p|| < \epsilon \} \) denotes an open sphere such that \( B(p, \epsilon) = B(p, \epsilon) \cap C \) is the neighbor set for \( p \) in \( \mathbb{R}^d \).
The neighbor selection strategy of HNSW. In the original paper of HNSW [67], the neighbor selection strategy is called heuristic neighbor selection. When selecting neighbors for the inserted point \( p \), HNSW regards \( p \) as a query to perform ANNS on the constructed partial graph index to obtain a certain amount of its nearest neighbors as candidate neighbors \( C \). Next, the heuristic neighbor selection iteratively gets the unvisited point \( m \) that has the smallest \( \delta(m, p) \) from \( C \), if \( \forall n \in N(p) \), \( \delta(m, n) > \delta(m, p) \) (Condition 1), then \( N(p) \cup \{m\} \), otherwise, \( m \) will be discarded. For more details, please see the Algorithm 4 in the original publication of HNSW [67].

The neighbor selection strategy of NSG. In the original paper of NSG [38], the neighbor selection strategy is called edge selection strategy of Monotonic Relative Neighborhood Graph (MRNG). When selecting neighbors for \( p \), MRNG gets the unvisited point \( m \) with the smallest \( \delta(m, p) \) from \( C \). If \( \text{lune}_{pm} \cap C = \emptyset \) or \( \forall u \in (\text{lune}_{pm} \cap C) \), \( u \notin N(p) \) (Condition 2), then \( N(p) \cup \{m\} \). For more details, please refer to the Algorithm 2 in the original publication of NSG [38].

Below we prove the equivalence of the neighbor selection of the two.

**Proof.** First, we prove that the neighbor selection of NSG can be derived from the neighbor selection of HNSW. For any point \( m \in C \) that can be added to \( N(p) \), we only need to prove that if Condition 1 is satisfied, then Condition 2 must be satisfied. For Condition 1: \( \forall n \in N(p) \), \( \delta(m, n) > \delta(m, p) \), we can infer that \( \forall n \in N(p) \) must satisfy \( n \notin B(m, \delta(m, p)) \), otherwise, there will be \( \exists n \in N(p) \) makes \( \delta(m, n) < \delta(m, p) \). Thus, we have

\[
\text{lune}_{pm} \cap N(p) = B(p, \delta(m, p)) \cap B(m, \delta(m, p)) \cap N(p) \\
= B(p, \delta(m, p)) \cap \emptyset \\
= \emptyset
\]

Since \( N(p) \) are all selected from \( C \), that is, \( N(p) \subseteq C \), below, we will discuss whether \( \text{lune}_{pm} \cap (C \setminus N(p)) = \emptyset \).

(1) Suppose \( \text{lune}_{pm} \cap (C \setminus N(p)) = \emptyset \). Then, we have

\[
\text{lune}_{pm} \cap C = \text{lune}_{pm} \cap ((C \setminus N(p)) \cup N(p)) \\
= (\text{lune}_{pm} \cap (C \setminus N(p))) \cup (\text{lune}_{pm} \cap N(p)) \\
= \emptyset \cup \emptyset \\
= \emptyset
\]

Condition 1 is satisfied.

(2) Suppose \( \text{lune}_{pm} \cap (C \setminus N(p)) \neq \emptyset \). Obviously, \( \text{lune}_{pm} \cap C \neq \emptyset \), because \( \text{lune}_{pm} \cap N(p) = \emptyset \), so \( \forall u \in (\text{lune}_{pm} \cap C) \) must have

\[
u \in \text{lune}_{pm} \cap (C \setminus N(p)) = (\text{lune}_{pm} \cap C) \setminus N(p)
\]

That is, \( u \notin N(p) \). Condition 1 is satisfied.

Therefore, if Condition 1 is established, then Condition 2 must be established.

Next, we prove that the neighbor selection of HNSW can be derived from the neighbor selection of NSG. For any point \( m \in C \) that can be added to \( N(p) \), we only need to prove that if Condition 2 is satisfied, then Condition 1 must be satisfied. For Condition 2: \( \text{lune}_{pm} \cap C = \emptyset \) or \( \forall u \in (\text{lune}_{pm} \cap C) \), \( u \notin N(p) \), we discuss the two cases of \( \text{lune}_{pm} \cap C = \emptyset \) and \( \forall u \in (\text{lune}_{pm} \cap C) \), \( u \notin N(p) \) separately.

**Figure 12:** The path adjustment of NGT [48].

(1) When \( \text{lune}_{pm} \cap C = \emptyset \) is established, because \( N(p) \subseteq C \), therefore, \( \text{lune}_{pm} \cap N(p) = \emptyset \). Since the unvisited point \( m \) with the smallest \( \delta(m, p) \) is taken from \( C \) every time, that is, \( \forall n \in N(p) \), \( \delta(n, p) < \delta(m, p) \), thus, \( \forall n \in N(p), n \in B(p, \delta(p, m)) \), and we have

\[
n \notin B(m, \delta(m, p)) \setminus B(p, \delta(p, m)) \\
= B(m, \delta(m, p)) \cap (B(m, \delta(m, p)) \cap B(p, \delta(p, m))) \\
= B(m, \delta(m, p)) \setminus \text{lune}_{pm}
\]

Thus, \( \forall n \in N(p), \delta(m, n) > \delta(m, p) \), Condition 1 is satisfied.

(2) When \( \forall u \in (\text{lune}_{pm} \cap C) \), \( u \notin N(p) \), it is easy to know that \( \forall n \in N(p), n \notin (\text{lune}_{pm} \cap C) \). Thus, \( n \notin \text{lune}_{pm} \), otherwise, if \( n \in \text{lune}_{pm} \), \( n \in C \) is known, then \( \exists n \in N(p) \) makes \( n \in (\text{lune}_{pm} \cap C) \), which contradicts the known. Since \( n \notin B(m, \delta(m, p)) \setminus \text{lune}_{pm} \), we have

\[
n \notin (B(m, \delta(m, p)) \setminus \text{lune}_{pm}) = B(m, \delta(m, p))
\]

Thus, \( \forall n \in N(p), \delta(m, n) > \delta(m, p) \), Condition 1 is satisfied.

Therefore, if Condition 2 is established, then Condition 1 must be established.

In summary, the neighbor selection strategies of HNSW and NSG are equivalent. □

**Appendix B. Proof for the path adjustment of NGT** is an approximation to the neighbor selection of RNG

**Path adjustment.** We can formally describe the path adjustment as follows. Given graph \( G(V, E) \), \( N(p) \) is the current neighbor set of \( p \in V \). As shown in Figure 12, if there is an alternative path \( p \rightarrow x \rightarrow n \) with length \( l = 2 \) (the number of edges) between \( p \) and \( n \), where \( n \in N(p) \), then do the following: If \( \max(\delta(p, x), \delta(x, n)) < \delta(p, n) \), then delete \( n \) from \( N(p) \), otherwise keep \( n \). If there is no alternative path between \( p \) and \( n \) or the length \( l = 2 \) of the alternative path, then \( n \) is also reserved. For more details about path adjustment, please refer to the original paper [48].

**The neighbor selection of RNG.** Given any point \( p \) on dataset \( S \), the candidate neighbor set of \( p \) is \( C \), gets the unvisited point \( m \) that has the smallest \( \delta(m, p) \) from \( C \), if \( \forall n \in N(p), \delta(m, n) > \delta(m, p) \), then \( N(p) \cup \{m\} \), otherwise, \( m \) will be discarded [43].

Next, we prove that the above path adjustment operation is an approximate implementation of neighbor selection of RNG. Given vertex \( p \) on \( G(V, E) \), the neighbors \( N(p) \) of \( p \) is sorted in ascending order according to the distance between each neighbor and \( p \), and the neighbors are visited in this order when conducting path adjustment. Therefore, the selection way of visited vertex for path
adjustment is consistent with neighbor selection of RNG. We only need to prove that the judgment criteria of the visited vertex cut or retained is an approximation to that of RNG.

Proof. We conduct the discussion based on the alternative path length \( l \) between \( p \) and \( n \).

(1) As shown in Figure 12 (a), (b), and (c), when the length \( l = 2 \) of alternative path \( p \rightarrow x \rightarrow n \) between \( p \) and \( n \), there are two situations:

- If \( \max(\delta(p, x), \delta(x, n)) < \delta(p, n) \) (Figure 12 (a)), then \( n \) will be deleted. It is easy to know that \( \delta(p, n) > \delta(x, n) \), which is consistent with the neighbor selection of RNG.

- If \( \max(\delta(p, x), \delta(x, n)) > \delta(p, n) \) (Figure 12 (b) and (c)), then \( n \) will be reserved. At this time, when \( \delta(p, n) < \delta(x, n) \) (Figure 12(b)), the neighbor selection of RNG is met; When \( \delta(p, n) > \delta(x, n) \) (Figure 12(c)), then \( \delta(p, n) < \delta(p, x) \). Since \( \delta(x, n) \) is small, \( n \in N(x) \) has a high probability of occurrence. We know that \( x \in N(p) \), when judging whether there is an alternative path between \( p \) and \( x \), since the alternative path \( (p \rightarrow n \rightarrow x) \) exists, and \( \delta(p, n) < \delta(p, x) \) and \( \delta(n, x) < \delta(p, x) \), that is, \( \max(\delta(p, n), \delta(n, x)) < \delta(p, x) \), thus, \( x \) needs to be deleted. At this time, keeping \( n \) and deleting \( x \) is consistent with the result of the neighbor selection of RNG.

(2) As shown in Figure 12(d), when there is an alternative path with length \( l > 2 \) between \( p \) and \( n \), it means that the distance between \( p \) and \( n \) is likely to be farther, so that most of the neighbors of \( n \) and \( p \) are far away. Therefore, \( \delta(n, p) < \delta(u, u) \) is easy to hold for most \( u \), and \( u \) is one of the neighbors of \( p \) except \( n \). In this case, the results of keeping \( n \) is consistent with the neighbor selection of RNG.

(3) As shown in Figure 12(e), if there is no alternative path between \( p \) and \( n \), it means that there is a high probability that \( p \) is closer to \( n \), so \( \delta(p, n) < \delta(u, n) \) for most \( u \), where \( u \) is one of the neighbors of \( p \) except \( n \). In this case, the results of keeping \( n \) is consistent with the neighbor selection of RNG.

In summary, the path adjustment of NGT is an approximation to the neighbor selection of RNG.

Appendix C. Proof for the neighbor selection of DPG is an approximation to that of RNG

Overview. According to Appendix A and Appendix B, the neighbor selection of RNG can be approximately described by the neighbor selection of HNSW. Due to the equivalence of the neighbor selection strategies of HNSW and RNG, we can represent the neighbor selection of RNG by the neighbor selection of RNG. Therefore, we only need to prove that the neighbor selection of DPG is an approximation to that of RNG.

The neighbor selection of DPG. The construction of DPG is a diversification of the KGraph [31], followed by adding reverse neighbors. Given any a vertex \( p \) on \( G(V, E) \), \( C \) is the candidate neighbor set of \( p \), \( \theta(x, y) \) denotes \( \angle xpy \), where \( x, y \in C \). The neighbor selection of DPG aims to choose a subset \( N(p) \) of \( x \) vertices from \( C \) so that \( N(p) = \arg \max_{x \in C} \sum_{x, y \in C} \theta(x, y) \). For full details, please see the original publication of DPG [61].

Figure 13: The neighbor selection of NSG and DPG.

Lemma 7.1. Given the \( G(V, E) \) constructed by the neighbor selection of NSG, any a vertex \( p \in V \) and the neighbor set \( N(p) \) of \( p \) on \( G(V, E) \). \( \forall x, y \in N(p), \angle xpy \geq 60^\circ \).

We prove Lemma 7.1 as follows.

Proof. As shown in Figure 13(a), for the \( N(p) \) obtained from \( C \) by the neighbor selection of RNG (Appendix A), if \( \exists x, y \in N(p) \), and \( \angle xpy < 60^\circ \), then we can know that \( \angle xpy + \angle xyp > 120^\circ \) in \( \angle xyp \), thus, there must be \( \angle xpy > 60^\circ \) or \( \angle xyp > 60^\circ \).

Suppose \( \delta(p, y) > \delta(p, x) \), then \( \angle xpy > 60^\circ \) (i.e., the situation shown in Figure 13(a)), we know that \( \angle xpy > \angle xpy \), so \( \delta(p, y) > \delta(x, y) \), we have \( x \in B(y, \delta(p, y)) \). Since \( \delta(p, y) > \delta(p, x) \), so \( x \in B(p, \delta(p, y)) \), it is easy to know that \( x \in B(y, \delta(p, y)) \cap B(p, \delta(p, y)) = \text{line}_{xy} \). However, \( x \in N(p) \), according to the neighbor selection strategy of NSG (Appendix A), \( y \) cannot be added to \( N(p) \), which contradicts the known \( y \in N(p) \). When \( \delta(p, y) < \delta(p, x) \), we can just swap the positions of \( x \) and \( y \) above, and then get the same conclusion.

Therefore, Lemma 7.1 is proved.

Now we prove that the neighbor selection of DPG is an approximation to that of NSG.

Proof. As shown in Figure 13(a), when selecting neighbors for \( p \), it needs to determine whether a point in \( C = \{x, y, z, u, v\} \) can be added to \( N(p) \) according to the neighbor selection of NSG, and finally \( N(p) = \{x, z, v\} \). It can be seen that \( \forall x, y \in N(p), \angle xpy \geq 60^\circ \) from Lemma 7.1.

Figure 13(b) is based on the neighbor selection strategy of DPG, which selects \( k = 3 \) neighbors to \( N(p) \) from \( C = \{x, y, z, u, v\} \) to maximize the sum of angles between neighbors. From this, it can be inferred that \( \exists \theta \in [60^\circ, 180^\circ] \), for \( \forall v_1, v_2 \in N(p) \), \( \theta(v_1, v_2) \geq \theta \). When \( \theta \geq 60^\circ \), that is, \( \theta \geq 60^\circ \), which ensures the result of neighbor selection of NSG.

Therefore, we can say that the neighbor selection of DPG is an approximation to that of NSG, and thus an approximation to that of RNG.

Appendix D. Complexity analysis

Since the construction or search complexity of some graph-based ANNS algorithms were not informed by the authors, we deduce the approximate complexity based on the description of the algorithm and our experimental evaluation. In order to make the evaluation results more general, we carefully select the characteristics of the dataset, which are shown in Table 8. The standard deviation is the standard deviation of the distribution in each cluster.
The construction process of DPG includes two steps: (1) a Dimension and (2) a Partition process. As shown in Figure 14(i), the search complexity of HCNN is lower than that of DPG. In [54], the authors report that the construction complexity of HCNNG is about $O(|S|^{0.54})$, which is very close to KGraph. It shows that EFANNA’s optimization to KGraph only changes the constant factor of the construction complexity. Since both are constructing a KNN with high quality, the search complexity of KGraph (Table 8) and EFANNA (Table 9) are also very similar in Figure 14(a) and (g).

**KGraph.** We set Recall@10 = 0.99 and evaluate how the number of distance evaluations with the data size. As shown in Figure 14(a), the complexity of search on KGraph is about $O(|S|^{0.54})$. When the size of the dataset is small ($< 0.6 \times 10^4$), the search complexity of KGraph is lower than that of $O(|S|^{0.67}(|S|))$. However, as the size of the dataset increases, the search complexity of KGraph is slightly higher than $O(|S|^{0.67}(|S|))$.

**NGT.** There are two main versions of NGT, NGT-panning and NGT-onng respectively. The main difference between the two is that NGT-onng has one extra step that out-degree and in-degree adjustment than NGT-panng [46]. According to the evaluation results in Figure 14(b) and (c), NGT-onng requires a larger construction time than NGT-panng due to additional out-degree and in-degree adjustment. However, NGT-onng has not received better search performance than NGT-panng, the search complexity of the two is very close, and NGT-onng has higher efficiency in the evaluation dataset, which shows that the effectiveness of out-degree and in-degree adjustment needs to be further verified.

**SPTAG.** We implement two versions of SPTAG, SPTAG-KDT and SPTAG-BKT respectively. SPTAG-KDT is the original version, which corresponds to the description of the paper [98, 100, 101]. SPTAG-BKT is an optimized version based on SPTAG-KDT, the specific description can refer to the online project [27]. As shown in Figure 14(d), SPTAG-KDT ($O(|S|^{0.68})$) and SPTAG-KMT ($O(|S|^{0.57})$) have approximately the same search time complexity. However, under the same dataset size, SPTAG-BKT requires fewer distance evaluation times, that is, SPTAG-BKT mainly optimizes the constant part of search complexity.

**NSW.** In [65], the authors give that the search complexity of NSW is $O(|S|^{0.64}(|S|))$ by experiment. For index construction, NSW is based on the idea that graph structure is assembled by inserting elements one by one and connects them with a certain amount of nearest neighbors at each step. These nearest neighbors connected to each vertex is obtained by greedy search on the already constructed graph. As the time complexity of inserting each vertex is $O(|S|^{0.64}(|S|))$, the complexity of constructing NSW on the dataset S is $O(|S|^{2\cdot|S|})$. IEH. In [54], the authors report that the construction complexity of KNN table is $O(|S|^2\cdot d \cdot \log{|S|})$, where $d$ is the dimensionality and $d \ll |S|$. As we all know, the complexity of building a hash bucket is much less than $O(|S|^2\cdot \log{|S|})$. Therefore, the construction complexity of IEH is $O(|S|^2 \cdot \log{|S|} + |S|^2)$. In Figure 14(e), the search complexity of IEH is about $O(|S|^{0.52})$. However, under the same dataset size, IEH requires more distance evaluations than most other algorithms.

**EFANNA.** The construction process of EFANNA is very similar to KGraph except for the initialization. KGraph initializes the neighbors of each vertex randomly, while EFANNA uses KD-trees to initialize the neighbors more accurately. From Figure 14(f), we can see that the construction complexity of EFANNA is about $O(|S|^{1.13})$, which is very close to KGraph. It shows that EFANNA’s optimization to KGraph only changes the constant factor of the construction complexity. Since both are constructing a KNN with high quality, the search complexity of KGraph ($O(|S|^{0.54})$) and EFANNA ($O(|S|^{0.55})$) are also very similar in Figure 14(a) and (g).

**DPG.** The construction process of DPG includes two steps: (1) a KNNG construction and (2) the diversification of the KNNG. For the first step, the time complexity of constructing KNNG through NN-Descent on the dataset S is $O(|S|^{1.14})$. The second step is the process of pruning edges on the KNNG constructed in the previous step to maximize the angle between neighbors, followed by adding reverse edges (Appendix C). For the vertex p on KNN, p’s neighbor set is $C = \{v_0, v_1, \ldots, v_{k-1}\}$, in which elements are sorted in ascending order of distance from p, and $|C| = k$, the result neighbor set is $N(p)$ and initialized to $\emptyset$ for DPG. At the beginning of selecting neighbors for p (the first iteration), we add $v_0$ to $N(p)$ and $C \setminus \{v_0\}$ ($|C| = k - 1, |N(p)| = 1$); In the second iteration, we select $v_i \in C$ to maximize $\sum_{p \in p_0} \sum_{j \in C \setminus \{v_i\}} |C| = k - 1$. Since $|N(p)| = 2$, we add $v_i \in C$ to initialize new neighbors $v_0 \in p_0$ and $v_i \in p_0$ is maximized, then let $N(p) \cup \{v_i\}$ and $C \setminus \{v_i\}$ ($|C| = k - 3, |N(p)| = 3$), we need $2 \cdot (k - 2)$ calculations for obtaining $v_i$.

Therefore, if we select c points from C to N(p), the total number of calculations is

$$\sum_{m=0}^{c-1} m \cdot (k - m) = \frac{c(c - 1)}{2} \cdot (k - \frac{k - (c - 1)(2c - 1)}{6}) = \frac{c(c - 1)}{2} \cdot (k - \frac{k - (c - 1)(2c - 1)}{6}) = O(c^2 \cdot k)$$

Thus, the time complexity of executing the above process for all |S| points is $O(c^2 \cdot k \cdot |S|) = O(|S|^2)$. Therefore, the construction complexity of DPG is $O(|S|^{1.14} + |S|)$.

**HCNNG.** As shown in Figure 14(i), the search complexity of HCNNG is about $O(|S|^{0.28})$, which is obviously lower than KGraph. This confirms the effectiveness of diversification on DPG.

**Vamana.** As shown in Figure 14(j), the construction complexity of Vamana is about $O(|S|^{1.16})$, which is close to KGraph and EFANNA.
Among the algorithms that approximate RNG, Vamana achieves the lowest construction complexity. The search complexity of Vamana is about $O(|S|^{0.75})$ from Figure 14(k), which is even lower than some algorithms that only approximate KNNG (like KGraph). We do not receive the results achieved in the original paper [88].

**k-DG.** The time complexity of k-DR to construct an exact KNNG through linear scanning is $O(|S|^2)$, and on this basis, the time complexity of deleting neighbors reachable through alternative paths is $O(k \cdot |S|)$, where $k$ is the number of neighbors of each vertex on KNNG, and $k \ll |S|$. Therefore, the construction complexity of k-DR is $O(|S|^2 + k \cdot |S|)$. As shown in Figure 14(l), the search complexity of k-DR is about $O(\log^{5.51}(|S|))$, which is lower than NGT (the constant factor of k-DG’s complexity is an order of magnitude lower than NGT). In the experimental evaluation, we can also see that the search performance of k-DG is better than NGT on all real world datasets.

**Appendix E. Characteristics of compared algorithms**

We summarize some salient characteristics of compared algorithms in Table 9. In the first line, Construction is the construction strategies of the algorithms. Candidate is the candidate neighbor acquisition. Preprocessing is the seed preprocessing, and Seed is the seed acquisition. In the fourth column, “search” indicates that the algorithm obtains candidate neighbor by ANNS on graph, “expansion” indicates that the algorithm obtains candidate neighbor by neighbor propagation. For the fifth column, “distance” represents that the algorithm considers distance factor when neighbor selection to get as close as possible neighbors, “distribution” means that the algorithm considers distribution factor when neighbor selection so that the neighbors are evenly distributed. In the sixth and seventh columns, “true” indicates that the algorithm ensures corresponding process, “false” indicates that the algorithm does not ensure this process. As for the ninth column, BFS, GS, RS are the abbreviations of Best First Search, Guided Search, and Range Search respectively.

**Appendix F. Algorithms description of best first search**

We describe the execution process of BFS in Algorithm 1.

**Appendix G. Characteristics of the synthetic datasets**

We summarize the characteristics of the nine synthetic datasets in this paper in Table 10, including dimension, cardinality, number of clusters (# Cluster), and standard deviation of the distribution in each cluster (SD).

**Appendix H. Parameters of the compared algorithms**

The optimal parameters of all algorithms on our experimental datasets are available from our online github repository4.

**KGraph.** When constructing the index, we search for the optimal values of KGraph’s five sensitive parameters ($K$, $L$, $\text{iter}$, $S$, $R$), and other parameters adopt the default values recommended by the author [31]. Increasing any of $K$, $L$, $S$ and $R$ has the effect of improving accuracy and slowing down speed at the same time. $\text{iter}$ is the number of iterations of the NN-Descent, the larger its value, the
NGT. It uses a method similar to NSW [65] to incrementally construct an approximate nearest neighbor graph (ANNG). The only difference from NSW is that the search algorithm used when NGT obtains candidate neighbors is range search. The parameter ε defaults to 1.1. In ANNG, the upper bound of the number of connected bidirectional edges is K for each vertex. Since ANNG is an undirected graph, the number of edges connected to each vertex may actually exceed K. On the basis of ANNG, NGT-panng [46] performs path adjustment (an approximation to RNG, see Appendix B for details) to cut redundant edges to ensure that the number of each vertex's neighbors is lower than the given parameter R. NGT-onng [46] first performs in-degree and out-degree adjustment on the basis of ANNG. The parameters involved are out_edge and in_edge, which respectively represent the number of outgoing and incoming edges (directed edges) of each vertex extracted from ANNG. Then, NGT-onng performs path adjustment like NGT-panng. For more details on these parameters, see [46–48].

SPTAG. There are mainly two implementation versions of SPTAG, one is the original version [98, 100, 101] (SPTAG-KDT) and the other is an improved version [27] (SPTAG-BKT). The graph index of SPTAG-KDT is a KNNG, and it uses KD-Tree to get the entry point when searching. SPTAG-BKT’s graph index adds the optimization of RNG on the basis of KNNG, and it uses k-means tree to get the entry point when searching. Detailed descriptions of relevant parameters can be found on the online web page.5

NSW. ef_construction controls the size of the candidate set, and it adjusts construction speed/quality tradeoff. max_mθ is the maximum size of bidirectional edge of each vertex, and it controls the index size of the NSW.

IEH. It contains three important parameters, i.e., p, k, s. p is the number of top nearest candidates, which are used for expansion in each iteration, k is the number of expansion, s is iteration number. According to the experiment evaluation in [54], p = 10, k = 50, s = 3 are reasonable for considering both high recall and low search time. However, our tests show that using the above recommended parameter values on most datasets does not receive the desired results. In order to get the specified recall rate, p must be increased.

Table 9: Characteristics of components within graph-based ANNS algorithms

| Algorithm | Construction | Initialization | Candidate | Neighbor Selection | Connectivity | Preprocessing | Seed | Routing |
|-----------|--------------|----------------|-----------|--------------------|--------------|---------------|------|---------|
| KGraph    | refinement   | random          | expansion | distance           | false        | false         | random | BFS     |
| NGT       | increment    | VP-tree         | search    | distance & distribution | false     | true          | VP-tree | RS      |
| SPTAG1    | divide-and-conquer | TP-tree | subspace   | distance & distribution | false     | true          | KD-tree | BFS     |
| SPTAG2    | divide-and-conquer | TP-tree | subspace   | distance & distribution | false     | true          | k-means tree | BFS    |
| NSW       | increment    | random          | search    | distance           | true        | false         | random | BFS     |
| IEH       | refinement   | brute force     | neighbors | distance           | false        | true          | hashing | BFS     |
| FANNNG    | refinement   | brute force     | neighbors | distance & distribution | false     | false         | random | BFS     |
| HNSW      | increment    | top layer       | search    | distance & distribution | false     | false         | top layer | BFS     |
| EFANNA    | refinement   | KD-tree         | expansion | distance           | true        | false         | KD-tree | BFS     |
| DPG       | refinement   | NN-Descent      | neighbors | distance           | false        | true          | random | BFS     |
| NSF       | refinement   | NN-Descent      | search    | distance & distribution | true     | true          | centroid | BFS     |
| HCNNG     | divide-and-conquer | clustering | subspace   | distance           | false        | false         | KD-tree | GS      |
| Yamana    | refinement   | random          | search    | distance & distribution | false     | true          | centroid | BFS     |
| NSSG      | refinement   | NN-Descent      | expansion | distance & distribution | true     | true          | random | BFS     |
| k-DR      | refinement   | brute force     | neighbors | distance & distribution | false     | false         | random | BFS or RS |
FANNG. L controls the size of candidate neighbors, and R is the maximum number of neighbors.

HNSW. \( M_0 \) and \( \text{ef} \_\text{construction} \) are used in the HNSW construction. \( M_0 \) is the maximum number of each vertex’s neighbors in the bottom layer. \( M \) controls the maximum number of each vertex’s neighbors in the high layer. \( \text{ef} \_\text{construction} \) is the size of the candidate set when selecting neighbors.

EFANNA. Different from the random initialization of KGraph, EFANNA initializes the neighbors of each vertex through KD-Tree. Therefore, two additional parameters are needed for EFANNA. \( nTrees \) controls the number of KD-Trees, and \( mLevel \) controls the maximum merged layers (see [36] for details).

DPG. DPG is acquired by diversifying KGraph’s neighbors, thus, there are also the parameters \( K, L, \text{iter}, S, R \) of KGraph. In addition, the upper bound on the number of neighbors at each point is fixed at \( K/2 \) during diversification. However, after adding the reverse edge operation, the number of neighbors at some points may surge back (\( \gg K/2 \)).

NSG. Similar to DPG, NSG also selects neighbors based on KGraph by appending an approximation to RNG. NSG has 3 additional parameters for the neighbor selection strategy, i.e., \( L, R, C \). \( L \) is the size of the candidate set when acquiring candidate neighbors for each vertex with the greedy search. The larger the \( L \), the closer the candidate neighbors to the target vertex, but the slower the acquisition operation. \( C \) controls the maximum size of the candidate neighbor set, \( R \) controls the index size of the graph, and \( \text{iter} \) controls the quality of the NGT, the larger the better,

HCNNG. Two parameters are used in the HCNNG construction: the number execution of hierarchical clustering procedures \( n \), and the minimum size of clusters \( n \). In addition, \( nTrees \) controls the number of KD-Tree for seed acquisition.

**Table 11: Maximum out-degree (D_max) and minimum out-degree (D_min) of the graph indexes of all compared algorithms**

| Alg. | UQ-V | Msong | Audio | SIFT1M | GIST1M | Crawl | GloVe | Enron |
|------|------|-------|-------|--------|--------|-------|-------|-------|
|      | D_max | D_min | D_max | D_min | D_max | D_min | D_max | D_min | D_max | D_min | D_max | D_min | D_max | D_min | D_max | D_min |
| KGraph | 40   | 40   | 100   | 100   | 40   | 40   | 90    | 90    | 40    | 40    | 90    | 90    | 80    | 80    | 100   | 100   | 50    | 50    |
| NGT-pang | 2,379 | 5    | 1,108 | 6     | 320  | 14   | 738   | 10    | 5,181 | 4     | 58,677 | 4     | 29,999 | 11    | 869   | 5     |
| NGT-ong | 2,663 | 3    | 1,935 | 10    | 500  | 7    | 849   | 5     | 6,798 | 6     | 120,928 | 4     | 60,115 | 26    | 1,242 | 6     |
| SPTAG-KDT | 32   | 32   | 32    | 32    | 32   | 32   | 32    | 32    | 32    | 32    | 32    | 32    | 32    | 32    | 32    | 32    |
| SPTAG-BKT | 32   | 32   | 32    | 32    | 32   | 32   | 32    | 32    | 32    | 32    | 32    | 32    | 32    | 32    | 32    | 32    |
| NSW | 880  | 30   | 5,334 | 60    | 1,130 | 40   | 1,901 | 40    | 16,693 | 60    | 245,301 | 60    | 195,123 | 80    | 5,013 | 80    |
| EIH | 50   | 50   | 50    | 50    | 50   | 50   | 50    | 50    | 50    | 50    | 50    | 50    | 50    | 50    | 50    | 50    |
| FANNG | 90   | 90   | 10    | 10    | 50   | 50   | 70    | 70    | 50    | 50    | 30    | 30    | 70    | 70    | 110   | 110   |
| HNSW | 40   | 40   | 80    | 1     | 50   | 1    | 50    | 1    | 60    | 1    | 70    | 1    | 60    | 1     | 80    | 1     |
| EFANNA | 40   | 40   | 50    | 50    | 10   | 10   | 60    | 60    | 100   | 100   | 100   | 100   | 100   | 100   | 40    | 40    |
| DPG | 460  | 50   | 821   | 50    | 359  | 50   | 389   | 50    | 8,981 | 50    | 120,942 | 50    | 65,073 | 50    | 1,189 | 50    |
| NSG | 32   | 21   | 1     | 30    | 30   | 30   | 30    | 30    | 47    | 1     | 1,068  | 1     | 125   | 1     | 62    | 1     |
| HCNNG | 187  | 3    | 209   | 10    | 92   | 10   | 150   | 11    | 85    | 2     | 208   | 9     | 294   | 24    | 190   | 8     |
| Yamana | 30   | 30   | 30    | 30    | 50   | 50   | 50    | 50    | 50    | 50    | 50    | 50    | 50    | 50    | 110   | 110   |
| NSSG | 20   | 70   | 1     | 20    | 5    | 20   | 5     | 41    | 1     | 61    | 1     | 31    | 1     | 31    | 1     |
| k-DR | 128  | 1    | 250   | 1     | 91   | 2    | 202   | 1     | 2,685 | 1     | 27,107 | 1     | 7,580 | 1     | 399   | 1     |

**Appendix I. Maximum and minimum out-degrees of the graph indexes of the compared algorithms**

Table 11 lists the maximum and minimum out-degrees of the graph index of each algorithm on the real-world dataset. In the search process, the algorithms can align the neighbor adjacent list to the same size (maximum out-degree), which will improve search efficiency by continuous memory access [37]. However, some algorithms whose maximum out-degree is too large easily exceed the memory limit and cannot take advantage of the above memory optimization (e.g., NSW, DPG, k-DR).

**Appendix J. Scalability of the graph-based ANNS algorithms**

**Dimensionality.** Table 12 reports that as the dimensionality increases, the CT of most algorithms increases. Interestingly, NSW and NGT are exactly the opposite of the above phenomenon. Note
In general, as the difficulty of dataset increases (i.e., the dimensionality or cardinality), the standard deviation at the beginning, and then drop. In particular, the QPS of RNG-based algorithms generally have better scalability when performing searches on datasets with different characteristics. While for index construction on these datasets, the algorithms based on NN-Descent have the highest efficiency. On high-dimensional datasets, KNNG- and RNG-based algorithms have similar search performance. As the scale, cluster, and standard deviation increase, the advantages of RNG- and MST-based algorithms over KNNG- and DG-based algorithms become more and more obvious. In addition, algorithms that include additional tree structures are generally difficult to obtain high search performance.

**Table 12: The construction time (CT) and queries per second (QPS) on synthetic datasets with different characteristics.**

| Alg.      | #Dimensionality | #Cardinality | #Clusters | Standard deviation |
|-----------|-----------------|--------------|-----------|-------------------|
|           | 8               | 32           | 128       |                   |
|           | CT              | QPS          | CT        | QPS               |
| KGraph    | 6               | 8,192        | 32        | 1,637             |
|           |                  | 43           | 561       | 499               |
|           |                  | 31           | 366       | 2,227             |
|           |                  | 6            | 32        | 1,637             |
|           |                  | 5            | 1,317     | 53                |
|           |                  | 3            | 3,137     | 3,317             |
|           |                  | 6            | 1,492     | 28                |
|           |                  | 45           | 420       | 420               |
| NGT-pang  | 377             | 1,643        | 145       | 343               |
|           |                  | 134          | 119       | 1,422             |
|           |                  | 9            | 1,677     | 115               |
|           |                  | 88           | 148       | 343               |
|           |                  | 155          | 464       | 155               |
| NGT-ong   | 191             | 1,322        | 161       | 174               |
|           |                  | 126          | 59        | 2439              |
|           |                  | 16           | 918       | 42                |
|           |                  | 123          | 36       | 161               |
|           |                  | 185          | 174       | 389               |
|           |                  | 176          | 184       | 174               |
| SPTAG-KDT | 138             | 70           | 243       | 39                |
|           |                  | 137          | 3           | 1,327             |
|           |                  | 108          | 4         | 243               |
|           |                  | 228          | 3         | 243               |
|           |                  | 31           | 1           | 1,327             |
|           |                  | 33           | 194       | 1,110             |
| SPTAG-BKT | 244             | 51           | 284       | 46                |
|           |                  | 342          | 2         | 342               |
|           |                  | 173          | 1         | 284               |
|           |                  | 137          | 1         | 46                |
|           |                  | 256          | 64         | 46                |
|           |                  | 64           | 46        | 64                |
|           |                  | 172          | 79         | 1301              |
|           |                  | 243         | 195       | 990               |
|           |                  | 77           | 4250      | 45                |
|           |                  | 1150         | 1,150     | 51                |
|           |                  | 1,150        | 2401      | 508               |
|           |                  | 1,075        | 4,361     | 1,177             |
|           |                  | 243         | 154       | 107               |
|           |                  | 1,637        | 4290      | 107               |
|           |                  | 1,075        | 154       | 107               |
|           |                  | 243         | 154       | 107               |
|           |                  | 1,637        | 4290      | 107               |
|           |                  | 1,075        | 154       | 107               |
|           |                  | 243         | 154       | 107               |
|           |                  | 1,637        | 4290      | 107               |
|           |                  | 1,075        | 154       | 107               |
|           |                  | 243         | 154       | 107               |
|           |                  | 1,637        | 4290      | 107               |
|           |                  | 1,075        | 154       | 107               |
|           |                  | 243         | 154       | 107               |
|           |                  | 1,637        | 4290      | 107               |
|           |                  | 1,075        | 154       | 107               |
|           |                  | 243         | 154       | 107               |
|           |                  | 1,637        | 4290      | 107               |
|           |                  | 1,075        | 154       | 107               |
|           |                  | 243         | 154       | 107               |
|           |                  | 1,637        | 4290      | 107               |
|           |                  | 1,075        | 154       | 107               |
|           |                  | 243         | 154       | 107               |
|           |                  | 1,637        | 4290      | 107               |
|           |                  | 1,075        | 154       | 107               |
|           |                  | 243         | 154       | 107               |
|           |                  | 1,637        | 4290      | 107               |
|           |                  | 1,075        | 154       | 107               |
|           |                  | 243         | 154       | 107               |
|           |                  | 1,637        | 4290      | 107               |
|           |                  | 1,075        | 154       | 107               |
|           |                  | 243         | 154       | 107               |
|           |                  | 1,637        | 4290      | 107               |
|           |                  | 1,075        | 154       | 107               |
Table 14: Index construction time (s) of the benchmark algorithm under different number of iterations.

| Dataset | iter=4 | iter=6 | iter=8 | iter=10 |
|---------|--------|--------|--------|---------|
| SIFT1M  | 164    | 269    | 408    | 532     |
| GIST1M  | 484    | 776    | 1,192  | 1,721   |

Table 15: Index construction time (s) of algorithms with different components.

| Component | Implementation Way | Dataset |
|-----------|--------------------|---------|
| C1        |                   | SIFT1M  |
| C1_NSNG   | 408                | 1,192   |
| C1_KGraph | 27                 | 90      |
| C1_EFANNA | 64                 | 212     |
| C2        |                   | GIST1M  |
| C2_NSGG   | 408                | 1,192   |
| C2_DPG    | 392                | 1,179   |
| C2_NSNG   | 547                | 1,719   |
| C3        |                   |         |
| C3_HNSW   | 408                | 1,192   |
| C3_DPG    | 1,065              | 22,175  |
| C3_KGraph | 1,304              | 29,164  |
| C3_Yamana | 1,236              | 24,932  |
| C4 & C6   |                   |         |
| C4_NSNG   | 408                | 1,192   |
| C4_HCNNG  | 493                | 1,465   |
| C4_IEH    | 458                | 1,299   |
| C4_NGT    | 432                | 1,305   |
| C4_NSNG   | 490                | 1,472   |
| C4_SPTAG-BKT | 559             | 1,774   |
| C5        |                   |         |
| C5_NSNG   | 408                | 1,192   |
| C5_Yamana | 406                | 1,261   |
| C7        |                   |         |
| C7_NSNG   | 408                | 1,192   |
| C7_FANNA  | 449                | 1,354   |
| C7_HCNNG  | 428                | 1,276   |
| C7_NGT    | 399                | 1,173   |

Appendix L. Evaluation of search performance under different NN-Descent iteration times

Table 4 confirms that the highest graph quality does not necessarily achieve the best search performance. Therefore, we set different NN-Descent iteration times to obtain different initial graph quality, so as to get an optimal iteration value of the benchmark algorithm for component evaluation during initialization. As shown in Figure 15, the search performance of the benchmark algorithm with different number of iterations shows the same trend on two real-world datasets. As the number of iterations increases, the search performance first increases and then decreases. This is consistent with the conclusion drawn in Table 4. As far as search performance is concerned, the graph quality is not the higher the better. In addition, according to Table 14, the greater the number of iterations, the higher the index construction time. In summary, we set the number of iterations of NN-Descent to 8 for the benchmark algorithm during initialization.

Appendix M. Index construction performance of components

Table 15 lists the index construction time of the benchmark algorithm using different components.

Table 16: Index construction time (ICT, unit: s) and index size (IS, unit: MB) of k-DR and NGT on real-world datasets, and the bold values are optimal.

| Algorithm | UQ-V | Msong | Audio | SIFT1M | GIST1M | Crawl | GloVe | Enron |
|-----------|------|-------|-------|--------|--------|-------|-------|-------|
| k-DR      |      |       |       |        |        |       |       |       |
| IS        | 66   | 100   | 3.6   | 98     | 156    | 171   | 199   | 8.4   |
| IS        | 2,094| 2,224 | 51    | 1,142  | 5,938  | 64,507| 13,014| 236   |
| IS        | 215  | 225   | 11    | 229    | 269    | 470   | 312   | 225   |
| IS        | 3,017| 4,814 | 69    | 1,989  | 15,832 | 174,996| 138,348| 414   |
| IS        | 198  | 224   | 9.9   | 214    | 296    | 527   | 578   | 71    |

Table 17: Index and search information of k-DR and NGT on real-world datasets, and the bold values are better.

| Algorithm | UQ-V | Msong | Audio | SIFT1M | GIST1M | Crawl | GloVe | Enron |
|-----------|------|-------|-------|--------|--------|-------|-------|-------|
| k-DR      |      |       |       |        |        |       |       |       |
| GQ        | 0.574| 0.659 | 0.613 | 0.647  | 0.754  | 0.555 | 0.747 | 0.577 |
| AD        | 16   | 25    | 17    | 25     | 40     | 21    | 43    | 22    |
| CC        | 19,889| 1    | 1    | 2    | 27    | 10    | 1    | 6    |
| CS        | 1,440| 50    | 30    | 130    | 230    | 220   | 210   | 550   |
| PL        | 399  | 61    | 29    | 166    | 305    | 2,713 | 1,402 | 385   |
| MO        | 1,119| 1,751 | 51    | 657    | 3,883  | 2,582 | 730   | 516   |
| NGT-panng |      |       |       |        |        |       |       |       |
| GQ        | 0.770| 0.681 | 0.740 | 0.762  | 0.567  | 0.628 | 0.589 | 0.646 |
| AD        | 52   | 56    | 49    | 56     | 67     | 58    | 66    | 55    |
| CC        | 65   | 10    | 10    | 20     | 10     | 10    | 10    | 10    |
| CS        | 79   | 144   | 33    | 438    | 1,172  | 5,132 | 2,281 | 85    |
| PL        | 1,432| 1,927 | 63    | 933    | 4,111  | 3,111 | 928   | 535   |
| MO        | 0.431| 0.395 | 0.412 | 0.424  | 0.266  | 0.203 | 0.220 | 0.331 |
| NGT-onng  |      |       |       |        |        |       |       |       |
| GQ        | 0.472| 0.455 | 0.435 | 0.468  | 0.400  | 0.322 | 0.274 | 0.333 |
| AD        | 47   | 55    | 45    | 53     | 75     | 66    | 124   | 53    |
| CC        | 1    | 1     | 1     | 1     | 1      | 1     | 1     | 1     |
| CS        | 1,002| 20    | 15    | 35     | 137    | 74    | 25    | 25    |
| PL        | 431  | 527   | 45    | 392    | 1,110  | 244   | 388   | 131   |
| MO        | 1,411| 2,007 | 63    | 859    | 4,088  | 3,147 | 1,331 | 333   |

Appendix N. Evaluation and analysis of k-DR algorithm

Overview. [7] presents a fast approximate similarity search method that utilized a degree-reduced k-nearest neighbor graph (k-DR). k-DR first builds a simple KNNG \( G(V, E) \), then for any vertex \( x \) and its neighbor \( y \in N(x) \), k-DR generates an undirected edge between \( x \) and \( y \) only if the BFS algorithm (Algorithm 1) starting from \( y \) cannot find \( x \) along the already existing edges on \( G \); this is actually an approximation of RNG (similar to NGT, please refer to Appendix B for proof). In addition, k-DR extends the similar range search to NGT [46] for routing.

We analyze the construction and search complexity of k-DR in Appendix D, and summarize some important attributes of k-DR in Table 18. The characteristics of components within k-DR is depicted in Table 9. Appendix H discusses the parameters setting of k-DR. Index construction, search, and scalability performance evaluations are given in Table 11, Table 12, Table 16, Table 17, Figure 20, and Figure 21. In Table 17, GQ, AD, CC, CS, PL, and MO are the abbreviations of graph quality, average out-degree, connected components, candidate set size, query path length, and peak memory overhead respectively.

Analysis and discussion. Here we mainly discuss the performance difference between k-DR and NGT due to the similarity of the two. In Table 12, k-DR exceeds NGT on simple dataset by a big margin, however, as the difficulty of the dataset increases, the performance...
Appendix O. Trade-off for efficiency vs accuracy
In order to comprehensively evaluate the search performance of each algorithm, their trade-off curves of Queries Per Second (QPS) vs Recall@10 and Speedup vs Recall@10 were measured on eight real-world datasets in Figure 20 and Figure 21. It is the most important part for the search performance evaluation of graph-based ANNS algorithms as the key of ANNS is seek a good trade-off between efficiency and accuracy. We mainly focus on the performance of each algorithm in the high-precision area due to actual needs [37, 38]. On GIST1M, Crawl, and GloVe datasets, SPTAG-BKT falls into the accuracy "ceiling" before reaching Recall@10=0.80, so it is not shown in Figure 20 and Figure 21.

Appendix P. Performance evaluation of the optimized algorithm
We evaluate the performance of the optimized algorithm (OA) and the state-of-the-art algorithms for index construction and search on two real world datasets of different difficulty in the same environment. According to our assessment, OA achieves the best overall performance.

Index construction performance. As shown in Table 19, Table 20 and Table 21, compared with the state-of-the-art algorithms, the index construction efficiency of the optimized algorithm (OA) ranks very high (second only to DPG, but OA performs better than DPG in other aspects), which is mainly because OA is not committed to achieving high graph quality at an expensive time cost (Table 21), and its neighbor acquisition does not involve distance calculation. There is no additional structure attached to the graph index for OA, which makes it obtain a smaller index size; and that is also due to its smaller average out-degree. In addition, OA ensures the accessibility from the entries to any other point, which is backed up by the number of connected components. OA obtains the optimal out-degree (AD), and # of connected components (CC) on graph indexes of the optimized algorithm (OA) and the state-of-the-art algorithms.

Table 18: Important attributes of k-DR.

| Base Graph | Edge | Build Complexity | Search Complexity |
|------------|------|------------------|-------------------|
| KNNG+RNG   | undirected | $O(|S|^2 + k \cdot |S|)$ | $O(\log^{13}(|S|))$ |

Table 19: Index construction time (s) of the optimized algorithm (OA) and the state-of-the-art algorithms.

| Dataset      | Algorithm | ICT 1 | ICT 2 | CS | PL 1 | PL 2 | MO |
|--------------|-----------|-------|-------|----|------|------|----|
| SIFT1M       | OA        | 1.791 | 1.206 | 88 | 97   | 80   | 394 |
|              | NSG       | 2.503 | 2.207 | 79 | 102  | 326  | 234 |
|              | NSSG      | 4.931 | 4.725 | 88 | 97   | 80   | 394 |
|              | HCNNG     | 8.603 | 8.360 | 79 | 102  | 326  | 234 |
|              | HNSW      | 18.632| 18.332| 88 | 97   | 80   | 394 |
|              | DPG       | 3.526 | 3.423 | 79 | 102  | 326  | 234 |
|              | NSG       | 0.354 | 0.345 | 79 | 102  | 326  | 234 |
| GIST1M       | OA        | 12.440| 11.850| 37 | 53   | 102  | 326 |
|              | NSG       | 2.503 | 2.207 | 79 | 102  | 326  | 234 |
|              | NSSG      | 4.931 | 4.725 | 79 | 102  | 326  | 234 |
|              | HCNNG     | 8.603 | 8.360 | 79 | 102  | 326  | 234 |
|              | HNSW      | 18.632| 18.332| 79 | 102  | 326  | 234 |
|              | DPG       | 3.526 | 3.423 | 79 | 102  | 326  | 234 |
|              | NSG       | 0.354 | 0.345 | 79 | 102  | 326  | 234 |

Table 20: Index size (MB) of the optimized algorithm (OA) and the state-of-the-art algorithms.

| Dataset      | Algorithm | CS | PL | MO |
|--------------|-----------|----|----|----|
| SIFT1M       | OA        | 1.206| 20 | 1  |
|              | NSG       | 2.207| 24 | 1  |
|              | NSSG      | 4.725| 24 | 1  |
|              | HCNNG     | 8.360| 61 | 1  |
|              | HNSW      | 18.332| 61 | 1  |
|              | DPG       | 3.423| 76 | 1  |

Table 21: Graph quality (GQ), average out-degree (AD), and # of connected components (CC) on graph indexes of the optimized algorithm (OA) and the state-of-the-art algorithms.

| Dataset      | Algorithm | SIFT1M | GIST1M |
|--------------|-----------|--------|--------|
|              | GQ | AD | CC | GQ | AD | CC |
| SIFT1M       | OA | 0.549 | 20 | 1 | 0.402 | 18 | 1 |
|              | NSG | 0.551 | 24 | 1 | 0.402 | 13 | 1 |
|              | NSSG | 0.579 | 20 | 1 | 0.399 | 26 | 1 |
|              | HCNNG | 0.887 | 61 | 1 | 0.354 | 42 | 1 |
|              | HNSW | 0.879 | 49 | 22 | 0.633 | 57 | 122 |
|              | DPG | 0.998 | 76 | 1 | 0.992 | 94 | 1 |

Table 23: Index construction time (ICT, unit: s) and index size (IS, unit: MB) of Vamana under different trials (a, b, and c).

| Trial | Dataset | ICT | IS |
|-------|---------|-----|----|
|       | SIFT1M  |     |    |
| a     | OA      | 1.451| 1.786| 158| 2.657|
| b     | NSG     | 1.575| 1.736| 145| 2.756|
| c     | NSSG    | 1.378| 1.695| 139| 2.688|
| Average|        | 1.468| 1.739| 147| 2.674|
|       | GIST1M  |     |    |
| a     | OA      | 118 | 118 | 11  | 195 |
| b     | NSG     | 118 | 118 | 11  | 195 |
| c     | NSSG    | 118 | 118 | 11  | 195 |
| Average|        | 118 | 118 | 11  | 195 |
the convenience of description, we divide any vertex’s neighbors to “good neighbors” (GN) and “bad neighbors” (BN). If we initialize GN for each vertex, Vamana’s index construction efficiency can reach the optimal value, as GN enable Vamana to perform ANNS more efficiently when acquiring candidate neighbors. In contrast, if BN are initialized for each element, it will lead to the worst index construction efficiency. However, the probability of the above two situations happening is extremely low (close to zero). Actually, the ratio of GN and BN of each element is relatively stable. In general, Vamana’s ANNS performance with different initial graphs is close (see Figure 17). When ignoring the cost of neighbor selection (it does not affected by the initial graph), its index construction efficiency mainly depends on ANNS on the initial graph for obtaining candidate neighbors; so it is almost the same under different trails. 

NSSG. The seed acquisition component of NSSG is randomized. As shown in Figure 18, the search performance curves of NSSG under different experiments almost overlap. For the convenience of description, we also divide the seeds into “good seeds” (GS) and “bad seeds” (BS). Performing search from GS can get the best search performance, while starting search from BS will result in the worst search performance. Due to the sufficient amount of query data (> 200, Table 3), the probability that the randomly obtained seeds is GS or BS for all queries is almost zero; for a given batch of queries, the ratio of the two is stable. Therefore, random seeds with multiple repetitions produce similar search performance.

Appendix R. Evaluation and analysis of machine learn (ML) based methods

Overview. In general, ML-based approaches append additional optimizations on existing graph-based algorithms. For example, [14] (ML1) learns vertex representation on graph-based algorithms (e.g., NSW) to provide a better routing; [59] (ML2) performs ANNS on HNSW through learned adaptive early termination, it builds and trains gradient boosting decision tree models to learn and predict when to stop searching for a certain query; [78] (ML3) maps the dataset into a space of lower dimension while trying to preserve local geometry by ML, then it can be combined with any graph-based algorithms like HNSW or NSG.

Setting. We implement ML1 and ML3 optimization on NSG, and implement ML2 optimization on HNSW (HNSW is selected in the original paper [59]); considering that when ML2 is applied to NSG, some additional optimization is required (we will add ML2 to NSG soon for evaluation after solving these problems). We focus on the index processing time, memory consumption during index construction, and speedup vs recall trade-off of each method. Note that ML1’s index preprocessing training is very time-consuming and memory-consuming (more than 125G on SIFT1M), so we use GPU to accelerate the process and use the smaller SIFT100K and GIST100K datasets. In addition, the number of nearest neighbors recalled is uniformly set to 1 for each query due to the limitation of ML1 [14], and Recall@1 represents the corresponding recall rate.

Discussion. As shown in Table 24 and Figure 19, there ML-based optimizations generally obtain better speedup (or QPS) vs recall tradeoff than the original algorithms (NSG or HNSW) at the expense of more time and memory. ML2 only provides slight latency reduction in the high-precision area. It is worth noting that ML2 have a smaller impact on IPT and IS (compared to HNSW). The IPT of ML1 is significantly higher than the original NSG, and it also requires additional index overhead. Although ML3 improves the speedup vs recall tradeoff by a large margin, it also significantly increases memory consumption.
Figure 17: Speedup vs Recall@10 of Vamana under different trails (a, b, and c).

Figure 18: Speedup vs Recall@10 of NSSG under different trails (a, b, and c).

Figure 19: Speedup vs Recall@10 of ML-based methods.
Figure 17: The Queries Per Second (QPS) vs Recall@10 of graph-based ANNS algorithms with their optimal indices in high-precision region on the eight real world datasets (top right is better).

Figure 20: The Queries Per Second (QPS) vs Recall@10 of graph-based ANNS algorithms with their optimal indices in high-precision region on the eight real world datasets (top right is better).
Figure 21: The Speedup vs Recall@10 of graph-based ANNS algorithms with their optimal indices in high-precision region on the eight real world datasets (top right is better).