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

Mengzhao Wang[1], Xiaoliang Xu[1], Qiang Yue[1], Yuxiang Wang[1,*]
[1]Hangzhou 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, offerings 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(11): 1964-1978, 2021.
doi:10.14778/3476249.3476255

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Lsyhprum/WEAVESS. Our full version has been made available at https://arxiv.org/abs/2101.12631.

1 INTRODUCTION

Nearest Neighbor Search (NNS) is a fundamental building block in various application domains [7, 8, 35, 62, 65, 74, 102, 109], such as information retrieval [31, 110], pattern recognition [26, 54], data mining [41, 44], machine learning [21, 25], and recommendation systems [64, 76]. With the explosive growth of datasets' scale and the inevitable *curse of dimensionality*, accurate NNS cannot meet actual requirements for efficiency and cost [57]. 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 (a tradeoff [56]).

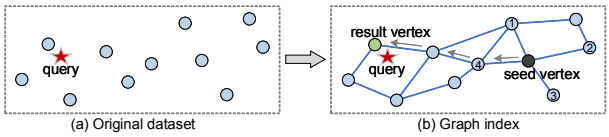


Figure 1: A toy example for the graph-based ANNS algorithm.

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 [37, 42]; tree-based [8, 80]; quantization-based [49, 68]; and graph-based [35, 62] algorithms. Recently, graph-based algorithms have emerged as a highly effective option for ANNS [6, 10, 38, 69]. Thanks to graph-based ANNS algorithms' extraordinary ability to express neighbor relationships [35, 97], they only need to evaluate fewer points of dataset to receive more accurate results [35, 57, 62, 66, 104].

As Figure 1 shows, graph-based ANNS algorithms build a graph index (Figure 1(b)) on the dataset (Figure 1(a)), the vertices in the graph correspond to the points of the dataset, and neighboring vertices (marked as x, y) are associated with an edge by evaluating their distance $\delta(x, y)$, 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 [44, 62]) is selected as the result vertex r, and we can conduct ANNS from this seed vertex. Specifically, if $\delta(n, q) < \delta(r, q)$, 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., $\forall n, \delta(n, q) \geq \delta(r, q)$) 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 vs efficiency [10, 57], which is probably why they enjoy widespread use among high-tech companies (e.g., Microsoft [24], Alibaba [104], and Yahoo [44]).

1.1 Motivation

The problem of graph-based ANNS on high-dimensional and large-scale data has been studied intensively across the literature [35]. Dozens of algorithms have been proposed to solve this problem from different optimizations [33, 34, 40, 51, 60, 62, 66]. For these algorithms, existing surveys [10, 57, 79] provide some meaningful

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476255

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, [57] includes a few graph-based algorithms (only three), [10] focuses on efficiency vs accuracy tradeoff, [79] 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[92, 95, 99]. 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 base graphs from different perspectives, i.e., Delaunay Graph (DG) [32], Relative Neighborhood Graph (RNG) [85], K-Nearest Neighbor Graph (KNNG) [69], and Minimum Spanning Tree (MST) [55]. Some algorithms, such as HNSW [62], DPG [57] can be categorized into RNG-based (DPG and HNSW) group working off RNG. 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 analyze graph-based ANNS algorithms from two coarse-grained components, i.e., construction and search [73, 79], which hinders insight into the key components. However, they can be divided into many fine-grained components such as *neighbor selection* [34, 35] and *routing* [13, 87] (**§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 [34]). Additionally, the key performance of completely different algorithms may be dominated by the same fine-grained component (e.g., the *neighbor selection* of NSG [35] and HNSW [62]). Such unusual but key discoveries occur by analyzing the components in detail to clarify which part of an algorithm mainly works.

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 [40, 59, 60], which primarily reflects related algorithms' search performance [56]. With the explosion of data scale and increasing requirements to update, the index construction performance has received more and more attention [104]. Related metrics such as graph quality (we define it in **§5.1**) [19], average out-degree, and so on indirectly affect the index construction performance. From our abundance of experiments (**§5**), we gain a novel discovery: higher graph quality does not necessarily achieve better search performance. For instance, HNSW [62] and DPG [57] 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 [28, 33–35].

I4: Diversified datasets are essential for graph-based ANNS algorithms' scalability evaluation. Some graph-based 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 (**§5**), 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. Evaluating and analyzing different scenarios' datasets leads to understanding performance differences better for 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 graph-based algorithms. 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 (**§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 diverse datasets. In terms of **I3**, we perform a thorough evaluation of algorithms and components in **§5**, with abundant metrics. For **I4**, we investigate different algorithms' scalability over different datasets (eight real-world and 12 synthetic datasets), covering multimedia data such as video, voice, image, and text.

(4) We discuss the recommendations, guidelines, improvement, tendencies, and challenges about graph-based ANNS algorithms. Based on our investigation, we provide some rule-of-thumb recommendations about the most suitable scenario for each single algorithm, along with useful guidelines to optimize algorithms, thus designing an algorithm obtains the state-of-the-art performance. Then we analyze graph-based ANNS algorithms' promising research directions and outstanding challenges (**§6**).

Table 1: Notations used in this paper

Notations	Descriptions
E^d	The Euclidean space with dimension d
$\lvert\cdot\rvert$	The cardinality of a set
S	A limited dataset in E^d, where every element is a vector
q	The query point in E^d; it is represented by a vector
$\delta(,)$	The Euclidean distance between points
$G(V,E)$	A graph index G where the set of vertices and edges are V and E, respectively
$N(v)$	The neighbors of the vertex v in a graph

2 PRELIMINARIES

Notations. Unless otherwise specified, relative notations appear in this paper by default as described in Table 1.

Modeling. For a dataset $S = \{s_0, s_1, \cdots, s_{n-1}\}$ of n points, each element s_i (denoted as x) in S is represented by a vector $\mathbf{x} = [x_0, x_1, \cdots, x_{d-1}]$ with dimension d. Using a similarity calculation of vectors with a similarity function on S, we can realize the analysis and retrieval of the corresponding data [22, 65].

Similarity function. For the two points x, y on dataset S, a variety of applications employ a distance function to calculate the similarity between the two points x and y [101]. The most commonly used distance function is the Euclidean distance $\delta(x, y)$ (l_2 norm) [79], which is given in Equation 1.

$$\delta(x, y) = \sqrt{\sum_{i=0}^{d-1}(x_i - y_i)^2}, \tag{1}$$

where x and y correspond to the vectors $\mathbf{x} = [x_0, x_1, \cdots, x_{d-1}]$, and $\mathbf{y} = [y_0, y_1, \cdots, y_{d-1}]$, respectively, here d represents the vectors' dimension. The larger the $\delta(x, y)$, the more dissimilar x and y are, and the closer to zero, the more similar they are [101].

2.1 Problem Definition

Before formally describing ANNS, we first define NNS.

Definition 2.1. **NNS.** Given a finite dataset S in Euclidean space E^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:

$$\mathcal{R} = \arg\min_{\mathcal{R}\subset S, |R|=k} \sum_{x\in\mathcal{R}} \delta(x, q). \tag{2}$$

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 [108]. 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 [94]. The ANNS problem is defined as follows:

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

Generally, we use recall rate $Recall@k = \frac{|\mathcal{R}\cap\tilde{\mathcal{R}}|}{k}$ to evaluate the search results' accuracy. ANNS algorithms aim to maximize $Recall@k$ 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 E^d, $G(V, E)$ 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 $u, v \in V$. Given a query q, seeds $\widehat{S}$, routing strategy, and termination condition, the graph-based ANNS initializes approximate k nearest neighbors $\tilde{\mathcal{R}}$ of q with $\widehat{S}$, then conducts a search from $\widehat{S}$ and updates $\tilde{\mathcal{R}}$ via a routing strategy. Finally, it returns the query result $\tilde{\mathcal{R}}$ once the termination condition is met.

2.2 Scope Illustration

To make our survey and comparison focused yet comprehensive, we employ some necessary constraints.

Graph-based ANNS. We only consider algorithms whose index structures are based on graphs for ANNS. Although some effective algorithms based on other structures exist, these methods' search performance is far inferior to that of graph-based algorithms. Over time, graph-based algorithms have become mainstream.

Dataset. ANNS techniques have been used in various multimedia fields. To comprehensively evaluate the performance of comparative algorithms, we select a variety of multimedia data, including video, image, voice, and text (§5). The base data and query data comprise feature vectors extracted by deep neural network (such as VGG [81] for image), and the ground-truth data comprise the query's 20 or 100 nearest neighbors calculated in E^d by linear scanning.

Core algorithms. This paper mainly focuses on in-memory core algorithms. For some hardware (e.g., GPU [105] and SSD [82]), heterogeneous (e.g., distributed deployment [27]), and machine learning (ML)-based optimizations [13, 56, 72] (see §5.5 for the evaluation of a few ML-based optimizations), we do not discuss these in detail, keeping in mind that core algorithms are the basis of these optimizations. In future work, we will focus on comparing graph-based ANNS algorithms with GPU, SSD, ML and so on.

3 OVERVIEW OF GRAPH-BASED ANNS

In this section, we present a taxonomy and overall analysis of graph-based ANNS algorithms from a new perspective. To this end, we first dissect several classic base graphs [18, 84], including Delaunay Graph [12, 32], Relative Neighborhood Graph [47, 85], K-Nearest Neighbor Graph [9, 69] and Minimum Spanning Tree [55, 77]. After that, we review 13 representative graph-based ANNS algorithms working off different optimizations to these base graphs.

3.1 Base Graphs for ANNS

We 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 [32] for DG's standard definition). DG ensures

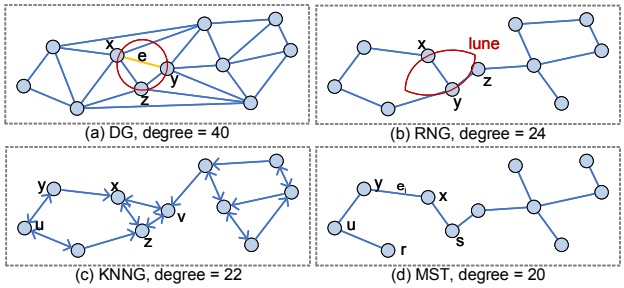


Figure 2: Schematic diagram of different base graphs' construction results on the same dataset with dimension $d = 2$.

that the ANNS always return precise results [62], but the disadvantage is that DG is almost fully connected when the dimension d is extremely high, which leads to a large search space [35, 40].

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 [85]). 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 [62]. However, the time complexity of constructing RNG on S is $O(|S|^3)$ [46].

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 $\sum_{i=1}^{|E|} w(e_i)$ on dataset S, where the two vertices associated with $e_i \in E$ are x and y, $w(e_i) = \delta(x, y)$. If $\exists e_i, e_j \in E, w(e_i) = w(e_j)$, then MST is not unique [63]. Although MST has not been adopted by most current graph-based ANNS algorithms, HCNNG [66] 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 [35, 60]. For example, in Figure 2(d), when search goes from s to r, it must detour with $s \rightarrow x \rightarrow y \rightarrow u \rightarrow r$. This can be avoided if there is an edge between s and r.

3.2 Graph-Based ANNS Algorithms

We outline 13 graph-based ANNS algorithms (**A1–A13**) based on the above 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 [14, 15, 53]. However, they rely heavily on DG's quality and exist the *curse of dimensionality* [60]. Therefore, some algorithms add an RNG approximation on DG to diversify the distribution of neighbors [62].

A1: Navigable Small World graph (NSW). NSW [60] 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 [62]. 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. However, NSW provides limited best tradeoff between efficiency and effectiveness, because its search complexity is poly-logarithmic [67, 71].

A2: Hierarchical Navigable Small World graphs (HNSW). An improvement direction is put forth by [17, 61] to overcome NSW's poly-logarithmic search complexity. Motivated by this, HNSW [62] 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). However, its multilayer structure significantly increases the memory usage and makes it difficult to scale to larger datasets [35]. Meawhile, [58] experimentally verifies that the hierarchy's advantage fades away as intrinsic dimension goes up (>32).

A3: Fast Approximate Nearest Neighbor Graph (FANNG). An occlusion rule is proposed by FANNG [40] to cut off redundant neighbors (approximate RNG). Unlike HNSW's approximation to RNG (HNSW only considers a small number of vertices returned by greedy search), FANNG's occlusion rule is applied to all other points on the dataset except the target point, which leads to high construction complexity. Thus, two intuitive optimizations of candidate neighbor acquisition are proposed to alleviate this problem [40]. To improve the accuracy, FANNG uses backtrack to the second-closest vertex and considers its edges that have not been explored yet.

A4: Neighborhood Graph and Tree (NGT). NGT [43] is a library for performing high-speed ANNS released by Yahoo Japan. It contains two construction methods. One is to transform KNNG into Bi-directed KNNG (BKNNG), which adds reverse edges to each directed edge on KNNG [44]. The other is constructed incrementally like NSW (approximate to DG) [44]. The difference from NSW is range search (a variant of greedy search) used during construction. Both of the aforementioned methods make certain hub vertices have a high out-degree. Therefore, NGT uses three degree-adjustment methods to alleviate this problem, and within the more effective path adjustment is an approximation to RNG (see our complete version for proof [96]) [45]. This reduces memory overhead and improves search efficiency. NGT obtains the seed vertex through the VP-tree [45], and then uses the range search to perform routing. Interestingly, the NGT-like path adjustment and range search are also used by the k-DR algorithm in [7] (see [96] for details).

Table 2: Summary of important representative graph-based ANNS algorithms

Algorithm	Base Graph	Edge	Build Complexity	Search Complexity
KGraph [28]	KNNG	directed	$O(\|S\|^{1.14})$	$O(\|S\|^{0.54})^{\ddagger}$
NGT [43]	KNNG+DG+RNG	directed	$O(\|S\|^{1.14})^{\ddagger}$	$O(\|S\|^{0.59})^{\ddagger}$
SPTAG [24]	KNNG+RNG	directed	$O(\|S\| \cdot \log(\|S\|^{c} + t^{t}))^{\dagger}$	$O(\|S\|^{0.68})^{\ddagger}$
NSW [60]	DG	undirected	$O(\|S\| \cdot \log^2(\|S\|))^{\ddagger}$	$O(\log^2(\|S\|))^{\dagger}$
IEH [51]	KNNG	directed	$O(\|S\|^2 \cdot \log(\|S\|) + \|S\|^2)^{\ddagger}$	$O(\|S\|^{0.52})^{\ddagger}$
FANNG [40]	RNG	directed	$O(\|S\|^2 \cdot \log(\|S\|))$	$O(\|S\|^{0.2})$
HNSW [62]	DG+RNG	directed	$O(\|S\| \cdot \log(\|S\|))$	$O(\log(\|S\|))$
EFANNA [33]	KNNG	directed	$O(\|S\|^{1.13})^{\ddagger}$	$O(\|S\|^{0.55})^{\ddagger}$
DPG [57]	KNNG+RNG	undirected	$O(\|S\|^{1.14} + \|S\|)^{\ddagger}$	$O(\|S\|^{0.28})^{\ddagger}$
NSG [35]	KNNG+RNG	directed	$O(\|S\|^{\frac{1+\epsilon}{c}} \cdot \log(\|S\|) + \|S\|^{1.14})^{\dagger}$	$O(\log(\|S\|))$
HCNNG [66]	MST	directed	$O(\|S\| \cdot \log(\|S\|))$	$O(\|S\|^{0.4})^{\ddagger}$
Vamana [82]	RNG	directed	$O(\|S\|^{1.16})^{\ddagger}$	$O(\|S\|^{0.75})^{\ddagger}$
NSSG [34]	KNNG+RNG	directed	$O(\|S\| + \|S\|^{1.14})$	$O(\log(\|S\|))$

† 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 our complete version for details [96].

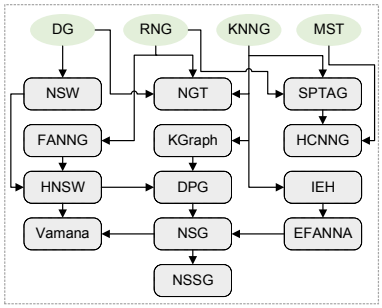


Figure 3: Roadmaps of graph-based ANNS algorithms. The arrows from a base graph (green shading) to an algorithm (gray shading) and from one algorithm to another indicate the dependence and development relationships.

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 [70] or hash [86, 103]), and then find the neighbors of each point through ANNS. However, such methods generally suffer from high index construction complexity [23]. There are two types of representative solutions, which only focus on graph construction.

A5: Space Partition Tree and Graph (SPTAG). One is based on divide and conquer, and its representative is SPTAG [24], a library released by Microsoft. SPTAG hierarchically divides dataset S into subsets (through Trinary-Projection Trees [94]) 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 [93]. The early version of SPTAG added multiple KD-trees on S to iteratively obtain the seeds closer to the query [91]. 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 [24].

A6: KGraph. The other is based on NN-Descent [29]; its basic idea is *neighbors are more likely to be neighbors of each other* [33]. KGraph [28] 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 [16]. Therefore, some NN-Descent-based derivatives are developed to explore its potential [20, 106, 107].

A7: EFANNA and A8: IEH. Instead of random initialization during construction (such as KGraph), Extremely Fast Approximate Nearest Neighbor Search Algorithm (**EFANNA**) [33] 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**) [51], which uses hash buckets to obtain better seeds. However, IEH's KNNG is constructed by brute force in [51].

KNNG-based and RNG-based ANNS algorithms (DPG, NSG, NSSG, Vamana). The early optimization of KGraph was limited to improving graph quality [33, 106]. Their intuition is that higher graph quality leads to better search performance. Hence, each vertex is only connected to K nearest neighbors without considering the distribution of neighbors. According to the comparative analysis of [58], if neighbors of a visiting vertex are close to each other, it will guide the search to the same location. That is, it is redundant to compare the query to all neighbors close to each other [40, 62].

A9: Diversified Proximity Graph (DPG). To overcome the aforementioned issue, DPG [57] 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 our complete version for the proof [96]). 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 [35]. To settle this problem and further improve search performance, NSG [35] proposes an edge selection strategy based on monotonic RNG (called MRNG), which is actually equivalent to HNSW's (see the full version for the proof [96]). 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.

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 [34]. 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 [34] 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: Vamana. Microsoft recently proposed Vamana [82] 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, Vamana initializes randomly. When selecting neighbors, Vamana improves the HNSW's strategy by adding a parameter α to increase the edge selection's flexibility

and executing two passes with different α. Experiments show that its result graph has a shorter average path length when searching.

MST-based ANNS algorithms (HCNNG).
A13: HCNNG. Different from the aforementioned techniques, a recent method called Hierarchical Clustering-based Nearest Neighbor Graph (HCNNG) [66] uses MST to connect the points on dataset S. 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 [40, 57, 60, 66]. 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*** [89], ***Refinement*** [29], and ***Increment*** [39] (see our full version [96]). 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 [23, 78]. For ***Refinement***, in the *initialization*, it performs *neighbor initialization* to get the initialized graph, then refines the initialized graph to achieve better search performance [33, 35]. 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 [60, 62]; 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., $S_0, S_1, \cdots, S_{m-1}$, 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 [88, 93]. To achieve better division, SPTAG turns to TP-tree [94], 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 [66].

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 Vamana implement this process by randomly selecting neighbors [28, 82]. 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 [86] or tree-based [33] 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 [33]. This approach relies heavily on extra index and increases the cost of index construction. Thus, NSG, DPG, and NSSG deploy the NN-Descent [29]; 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 $\widehat{S}$ from V as the seed set, and ANNS on G starts from $\widehat{S}$.

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 [43, 60], while HNSW makes its seed points fixed from the top layer because of its unique hierarchical structure [62].

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 [66, 93]. 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 [40, 51]. To solve this problem, NGT, NSW, HNSW, NSG, and Vamana all obtain candidate neighbors through ANNS. For a point $p \in S$, the graph G_{sub} (***Increment***) formed by the previously inserted points or the initialized graph G_{init} (***Refinement***), they consider p as a query and execute ANNS on G_{sub} or G_{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 [93], obtaining candidate neighbors through ANNS is overkill, because the query is in S for index construction, but the ANNS query generally does not belong to S. In contrast, KGraph, EFANNA, and NSSG use the neighbors of p and neighbors' neighbors on G_{init} as its candidate neighbors [34], which improves index-construction efficiency. DPG directly uses the neighbors of p on G_{init} as candidate

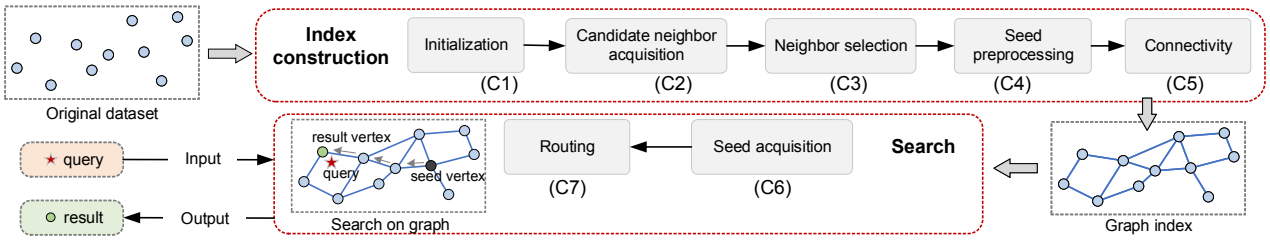


Figure 4: The pipeline of graph-based ANNS algorithms. An algorithm can be divided into two coarse-grained components: index construction, search. We subdivide the index construction into five fine-grained components (C1–C5), the search into two fine-grained components (C6–C7).

neighbors, but to obtain enough candidate neighbors, it generally requires G_{init} with a larger out-degree [57].

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[1], KGraph, EFANNA, and IEH only consider the distance factor and aim to build a high-quality graph index [29, 93]. HNSW[2], 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 $N(p)$ [35, 40]. To select neighbors more flexibly, Vamana adds the parameter α 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 $N(p)$ [82], so it can control the distribution of neighbors well by adjusting α. DPG obtains a subset of C to minimize the sum of angles between any two points [57]. NSSG considers the space distribution factor by setting an angle threshold θ, for $x \in C$, $\forall y \in N(p)$, iff $\arccos(x,y) < \theta$, x will join $N(p)$. NGT[3] indirectly attains the even distribution of neighbors with path adjustment [45], which updates neighbors by judging whether there is an alternative path between point p and its neighbors on G_{init}. HCNNG selects neighbors for p by constructing an MST on $\{p\} \cup C$ [66].

C4: Seed preprocessing. Different algorithms may exist with different execution sequences between this component and the *connectivity*, such as NSW [60], NSG [35]. Generally, graph-based ANNS algorithms implement this component in a static or dynamic manner. For the static method, typical representatives are HNSW, NSG, Vamana, and NSSG. HNSW fixes the top vertices as the seeds, NSG and Vamana use the approximate centroid of S as the seed, and the seeds of NSSG 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 [24, 33], balanced k-means tree [24], and VP-tree [43]. IEH prepares for *seed acquisition* through hashing [51]. Then [30] compresses the original vector by OPQ [36] 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 traversal to achieve this [35] (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 $\widehat{S}$, the *routing* starts from the vertices in $\widehat{S}$, 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 $\mathcal{R}$. We initialize C and $\mathcal{R}$ with seed set $\widehat{S}$. For $\hat{x} = \arg\min_{x \in C} \delta(x,q)$, best first search access $N(\hat{x})$, then $C \setminus \{\hat{x}\}$ and $C \cup N(\hat{x})$. To keep $|C| = c$, $\hat{y} = \arg\max_{y \in C} \delta(y,q)$ will be deleted. $\forall n \in N(\hat{x})$, if $\delta(n,q) < \delta(\hat{z},q), \hat{z} = \arg\max_{z \in \mathcal{R}} \delta(z,q)$, then $\mathcal{R} \setminus \{\hat{z}\}$ and $\mathcal{R} \cup \{n\}$. The aforementioned process is performed iteratively until $\mathcal{R}$ is no longer updated. (see our complete version for the pseudocode [96])

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 Vamana use the original BFS to perform *routing*. Despite this method being convenient for deployment, it has two shortcomings: susceptibility to local optimum (S1) [13] and low routing efficiency (S2) [66]. S1 destroys the search results' accuracy. For this problem, FANNG adds backtracking to BFS, which slightly improves the search accuracy while significantly increasing the search time [40]. NGT alleviates S1 by adding a parameter ϵ. On the basis of Definition 4.7, it cancels the size restriction on C and takes $\delta(\hat{y},q)$ as the search radius r, for $\forall n \in N(\hat{x})$, if $\delta(n,q) < (1+\epsilon) \cdot r$, then n is added to C. Setting ϵ to a larger value can alleviate S1, but it will

[1]This refers to its original version—NGT-panng for NGT and SPTAG-KDT for SPTAG.
[2]Although [35] distinguishes the *neighbor selection* of HNSW and NSG, we prove the equivalence of the two in our complete version [96].
[3]This refers to its optimized version—NGT-onng for NGT and SPTAG-BKT for SPTAG.

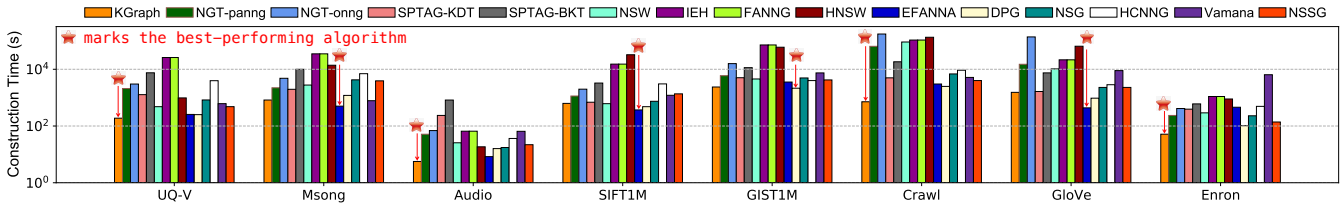


Figure 5: Index construction time of all compared algorithms on real-world datasets (the bar marked with a red star is the best).

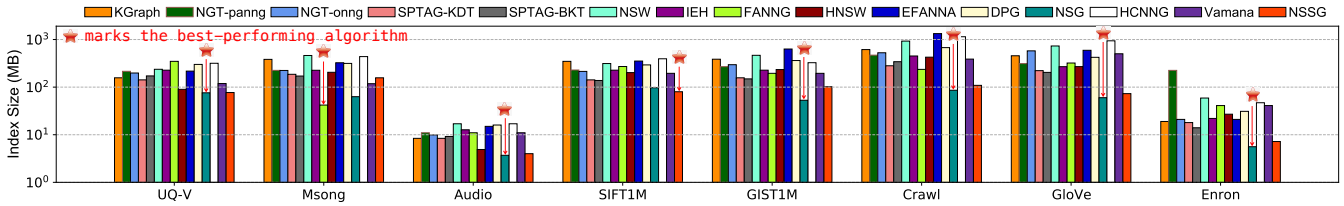


Figure 6: Index size of all compared algorithms on real-world datasets (the bar marked with a red star is the best).

also significantly increase the search time [45]. 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 [91]. HCNNG proposes using guided search to alleviate S2 rather than visiting all $N(\hat{x})$ like BFS, so guided search avoids some redundant visits based on the query's location.

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 [96]. Our evaluation seeks to answer the following question:

Q1: How do the algorithms perform in different scenarios? (§5.2–5.3)
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], ***Audio*** [4]), text (***Crawl*** [3], ***GloVe*** [48], ***Enron*** [75]), and image (***SIFT1M*** [1], ***GIST1M*** [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 [57]. 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 [79]). Out of space considerations, please see the scalability evaluation in our complete version [96]. 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 our complete version [96].

Table 3: Statistics of real-world datasets.

Dataset	Dimension	# Base	# Query	LID [34, 57]
UQ-V [5]	256	1,000,000	10,000	7.2
Msong [2]	420	992,272	200	9.5
Audio [4]	192	53,387	200	5.6
SIFT1M [1]	128	1,000,000	10,000	9.3
GIST1M [1]	960	1,000,000	1,000	18.9
Crawl [3]	300	1,989,995	10,000	15.7
GloVe [48]	100	1,183,514	10,000	20.0
Enron [75]	1,369	94,987	200	11.7

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 $\frac{|E' \bigcap E|}{|E|}$ [19, 23, 90]. 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., $\frac{\#q}{t}$ [35]. *Speedup* is defined as $\frac{|S|}{NDC}$, 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) [66]. We use the *recall* rate to evaluate the search accuracy, which is defined as $\frac{|R \cup T|}{|T|}$, 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, 83]. All C++ source codes are compiled by g++ 7.3, and MATLAB source codes (only for index construction of a hash table in IEH [51]) are compiled by MATLAB

Table 4: Graph quality (GQ), average out-degree (AD), and # of connected components (CC) on graph indexes (the blue values are the best).

Alg.	UQ-V			Msong			Audio			SIFT1M			GIST1M			Crawl			GloVe			Enron		
	GQ	AD	CC	GQ	AD	CC	GQ	AD	CC	GQ	AD	CC	GQ	AD	CC	GQ	AD	CC	GQ	AD	CC	GQ	AD	CC
KGraph	0.974	40	8,840	1.000	100	3,086	0.994	40	529	0.998	90	331	0.995	100	39,772	0.927	80	290,314	0.949	100	183,837	0.992	50	3,743
NGT-panng	0.770	52	**1**	0.681	56	**1**	0.740	49	**1**	0.762	56	**1**	0.567	67	**1**	0.628	58	**1**	0.589	66	**1**	0.646	55	**1**
NGT-onng	0.431	47	**1**	0.393	55	**1**	0.412	45	**1**	0.424	53	**1**	0.266	75	**1**	0.203	66	**1**	0.220	124	**1**	0.331	53	**1**
SPTAG-KDT	0.957	32	27,232	0.884	32	110,306	0.999	32	996	0.906	32	23,132	0.803	32	290,953	0.821	32	672,566	0.630	32	594,209	0.983	32	7,500
SPTAG-BKT	0.901	32	71,719	0.907	32	42,410	0.992	32	61	0.763	32	82,336	0.435	32	45,9529	0.381	32	1,180,072	0.330	32	803,849	0.775	32	20,379
NSW	0.837	60	**1**	0.767	120	**1**	0.847	80	**1**	0.847	80	**1**	0.601	120	**1**	0.719	120	**1**	0.636	160	**1**	0.796	160	**1**
IEH	1.000	50	24,564	1.000	50	9,133	1.000	50	335	1.000	50	1,211	1.000	50	74,663	1.000	50	289,983	1.000	50	220,192	1.000	50	3,131
FANNG	1.000	90	3,703	0.559	**10**	15,375	1.000	50	164	0.999	70	256	0.998	50	47,467	0.999	30	287,098	1.000	70	175,610	1.000	110	1,339
HNSW	0.597	**19**	433	0.762	50	36	0.571	20	**1**	0.879	49	22	0.633	57	122	0.726	52	3,586	0.630	56	624	0.833	68	9
EFANNA	0.975	40	8,768	0.997	50	10,902	0.976	**10**	3,483	0.998	60	832	0.981	100	44,504	0.990	100	227,146	0.751	100	234,745	0.999	40	3,921
DPG	0.973	77	2	1.000	82	**1**	0.999	74	**1**	0.998	76	**1**	0.992	94	**1**	0.982	88	**1**	0.872	93	**1**	0.993	84	**1**
NSG	0.562	**19**	**1**	0.487	16	**1**	0.532	17	**1**	0.551	24	**1**	0.402	**13**	**1**	0.540	**10**	**1**	0.526	**12**	**1**	0.513	**14**	**1**
HCNNG	0.836	41	**1**	0.798	69	**1**	0.847	38	**1**	0.887	61	**1**	0.354	42	**1**	0.503	109	**1**	0.425	167	**1**	0.662	85	**1**
Vamana	0.034	30	5,982	0.009	30	2,952	0.185	50	**1**	0.021	50	82	0.016	50	209	0.020	50	730	0.024	110	3	0.234	110	**1**
NSSG	0.508	**19**	**1**	0.634	40	**1**	0.494	19	**1**	0.579	**20**	**1**	0.399	26	**1**	0.580	13	**1**	0.474	15	**1**	0.517	19	**1**

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 [35], 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 α 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 the full version [96]. 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 [105]. The search performance of RNG- 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

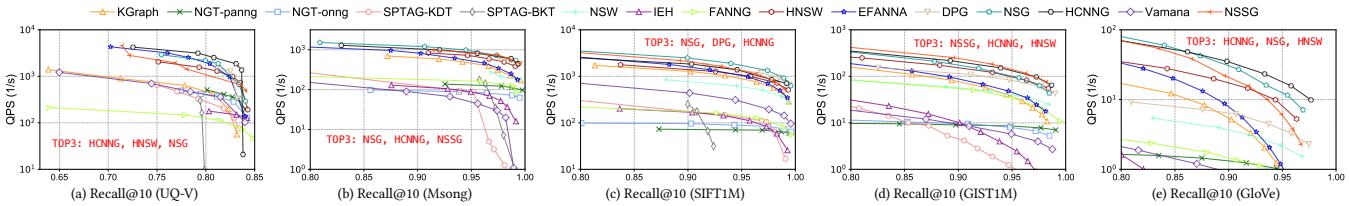


Figure 7: The Queries Per Second (QPS) vs Recall@10 of graph-based ANNS algorithms in high-precision region (top right is better).

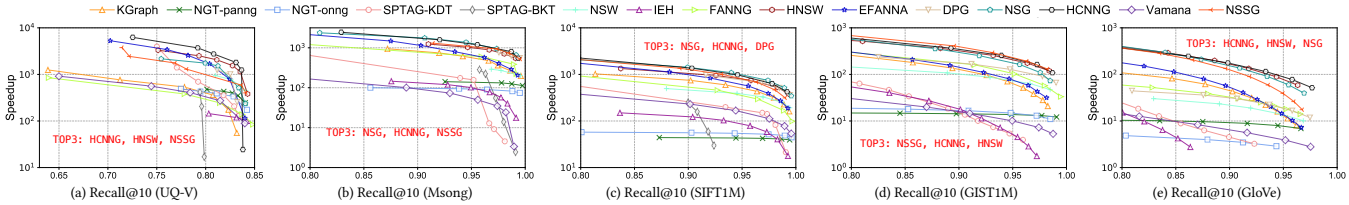


Figure 8: The Speedup vs Recall@10 of graph-based ANNS algorithms in high-precision region (top right is better).

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 [91], we know that the tree has bad *curse of dimensionality* [57].

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 [105]. In Table 5, DG-based and most RNG-based algorithms (e.g., NGT 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).

Query path length (PL). On large-scale datasets, it generally is necessary to use external storage to store the original data. Normally the PL determines the I/O number, which restricts the corresponding search efficiency [82]. From Figure 7 and Table 5, we see that algorithms with higher search performance generally have smaller PL (e.g., HCNNG), but algorithms with smaller PL do not necessarily have good search performance (e.g., FANNG). In addition, it makes sense that sometimes that an algorithm with a large average out-degree also has a small PL (e.g., NSW).

Memory overhead (MO). As Table 5 show, RNG-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 pipline in Figure 4. Each component in the evaluation framework is set for a certain implementation to form a benchmark algorithm (see our complete version for detailed settings [96]). We use the *C# + algorithm name* to indicate the corresponding component's specific implementation. For example, *C1_NSG* 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_NSG*, *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 9 and our complete version [96], respectively.

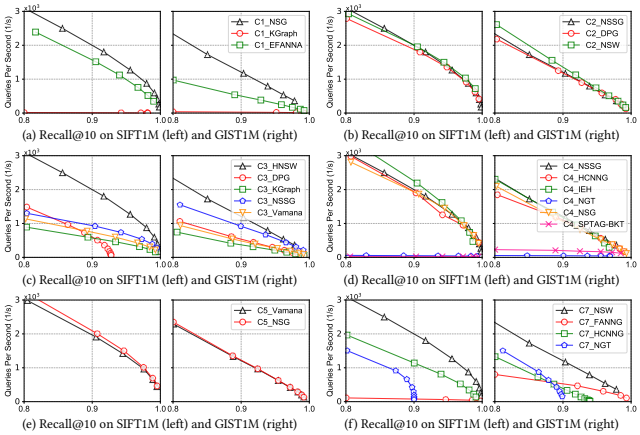
C1: Initialization. Figure 9(a) reports the impact of different graph index initialization methods on search performance. The search performance of *C1_NSG* is much better than *C1_EFANNA* and *C1_KGraph*; and although *C1_NSG* needs more construction time, it is worthwhile for such a large performance improvement. Moreover, a larger gap exists between *C1_NSG* and the other two on GIST1M (harder), which shows that it has better scalability.

C2: Candidate neighbor acquisition. As shown in Figure 9(b), different candidate neighbor acquisition methods vary slightly. *C2_NSW* has the best search performance, especially on GIST1M, with the price being more construction time. *C2_NSSG* obtains 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_NSW*) exceeds that of *C2_DPG*.

C3: Neighbor selection. Figure 9(c) depicts the impact of different neighbor selection schemes on search performance. Obviously, it shows better search performance for algorithms that consider the distribution of neighbors (e.g., *C3_HNSW*, *C3_NSSG*, *C3_DPG*, *C3_Vamana*) than those that do not consider this (e.g., *C3_KGraph*). Note that *C3_Vamana*'s performance is no better than *C3_HNSW*'s, as claimed in the paper [82]. NSSG[34] appears to have better search

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

Alg.	UQ-V			Msong			Audio			SIFT1M			GIST1M			Crawl			GloVe			Enron		
	CS	PL	MO	CS	PL	MO	CS	PL	MO	CS	PL	MO	CS	PL	MO	CS	PL	MO	CS	PL	MO	CS	PL	MO
KGraph	15,000+	1,375	1,211	50,442	1,943	2,036	701	105	55	139	52	900	2,838	411	4,115	50,000+	3,741	3,031	24,318	1,333	991	9,870	607	525
NGT-panng	65	79	1,423	**10**	144	1,927	**10**	33	63	**20**	438	933	**10**	1,172	4,111	**10**	5,132	3,111	**10**	2,281	928	**10**	83	535
NGT-onng	1,002	431	1,411	20	227	2,007	15	45	63	33	392	859	33	1,110	4,088	157	244	3,147	74	388	1,331	25	131	533
SPTAG-KDT	37,097	2,259	2,631	50,000+	11,441	2,091	61	107	91	7,690	1,227	1,048	50,000+	15,162	5,643	50,000+	12,293	8,872	50,000+	10,916	11,131	10,291	592	569
SPTAG-BKT	15,000+	10,719	5,114	97,089	11,119	1933	10	61	91	50,000+	8,882	4,587	50,000+	7,685	4,299	50,000+	10,851	6,643	50,000+	7,941	4,625	93,294	5,126	629
NSW	85	**38**	1,857	20	35	3,122	18	17	101	58	54	1,574	69	161	5,180	36	435	5,217	65	634	2,782	21	29	690
IEH	**29**	196	5,166	301	1,007	6,326	53	269	253	238	816	4,170	9,458	24,339	10,508	15,000+	5,928	10,913	15,000+	3,620	4,681	274	893	1,302
FANNG	1,072	86	1,395	594	245	**1,687**	1,462	195	58	1,377	92	825	3,007	269	3,917	8,214	2,152	2,639	9,000	1,062	850	2,084	152	548
HNSW	927	296	1,424	43	35	2,370	51	37	67	66	47	1,206	181	130	4,372	61	108	3,950	505	334	1,294	59	32	595
EFANNA	1,446	217	1,297	312	85	2,030	800	283	67	204	76	967	1,652	292	4,473	1311	180	3,848	22,349	1241	1,188	2,180	254	531
DPG	15,000+	1,007	1,352	16	**20**	1,965	**10**	**10**	62	37	**30**	851	55	**124**	4,091	67	761	3,089	84	792	956	89	60	538
NSG	354	156	**1,127**	106	90	1,714	63	47	**51**	101	85	653	867	826	**3,781**	345	723	**2,499**	814	1,875	**594**	118	138	**513**
HCNNG	15,000+	1,398	1,472	62	21	2,200	35	12	69	97	37	1,056	371	179	4,159	173	**61**	3,753	217	**95**	1,590	62	**20**	564
Vamana	1,049	346	1,164	40,596	7,155	1,763	68	30	57	493	263	748	8,360	3,127	3,916	53,206	7,465	2,786	22,446	2,157	1026	526	103	547
NSSG	310	122	1,129	39	40	1,807	65	42	**51**	255	157	**640**	280	270	3,829	13,810	12,892	2,524	3,846	3,047	605	458	236	514



(a) Recall@10 on SIFT1M (left) and GIST1M (right)
(b) Recall@10 on SIFT1M (left) and GIST1M (right)
(c) Recall@10 on SIFT1M (left) and GIST1M (right)
(d) Recall@10 on SIFT1M (left) and GIST1M (right)
(e) Recall@10 on SIFT1M (left) and GIST1M (right)
(f) Recall@10 on SIFT1M (left) and GIST1M (right)

Figure 9: Components' search performance under a unified framework on SIFT1M (a simple dataset) and GIST1M (a hard dataset).

performance than NSG in their experiment, so the researchers believe that *C3_NSSG* is better than *C3_NSG* (i.e., *C3_HNSW*). However, the researchers do not control the consistency of other components during the evaluation, which is unfair.

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 9(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 the best 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 9(e) shows the algorithm with guaranteed connectivity has better search performance (e.g., *C5_NSG*) than that without connectivity assurance (e.g., *C5_Vamana*).

Table 6: Index processing time (IPT) and memory consumption (MC) of ML1-based optimization.

Method		SIFT100K	GIST100K
IPT(s)	NSG	**55**	**142**
	NSG+ML1	55+67,260	142+45,600
MC(GB)	NSG	**0.37**	**0.68**
	NSG+ML1	23.8	58.7

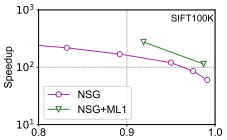


Figure 10: Speedup vs Recall@1 of NSG under ML1-based optimizations.

C7: Routing. Figure 9(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_HCNNG* 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 [13, 56, 72]. 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 [13], ML2 [56], and ML3 [72]. Because of space limitations, we only show the test results on ML1 in Table 6 and Figure 10, others share similar feature (see our full version [96] 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 graph-based algorithms can return query results in < 5ms, some high-tech companies only deploy NSG without ML-based optimizations in real scenarios [35, 52, 65].

6 DISCUSSION

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

Table 7: Recommendation of the algorithms in different scenarios.

Scenario	Algorithm
S1: A large amount of data updated frequently	NSG, NSSG
S2: Rapid construction of KNNG	KGraph, EFANNA, DPG
S3: Data is stored in external memory	DPG, HCNNG
S4: Search on hard datasets	HNSW, NSG, HCNNG
S5: Search on simple datasets	DPG, NSG, HCNNG, NSSG
S6: GPU acceleration	NGT
S7: Limited memory resources	NSG, NSSG

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 [82]), DPG and HCNNG are the best choices because their smaller average path length can reduce I/O times. On hard datasets (**S4**, large LID), HNSW, NSG, and HCNNG show competitive search performance, while on simple datasets (**S5**), DPG, NSG, HCNNG, and NSSG offer better search performance. For **S6**, we need a smaller candidate set size because of the cache's limitation [105]; 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 *C3_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_HCNNG*). In terms of **H3**, to improve the search's immunity from falling into the local optimum [13], we should reasonably design the distribution of neighbors during construction, ensure connectivity (such as *C5_NSG*), and optimize the routing strategy [87]. For **L4**, we can start by reducing the out-degree and candidate set size, and this can be achieved by improving the neighbor selection (such as *C3_NSG*) and routing strategies (like *C7_HCNNG*).

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_NSSG* (C2), uses *C3_NSG* to trim redundant neighbors (C3), randomly selects a certain number of entries (C4), and ensures connectivity (C5); in the search phase, it starts from the random entries (C6), and performs two-stage routing through *C7_HCNNG* and *C7_NSW* in turn. As shown in Figure 11, the optimized algorithm yields the state-of-the-art search performance, while ensuring high construction efficiency and low memory overhead (see [96] for more details).

Tendencies. 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 was applied to graph-based ANNS,

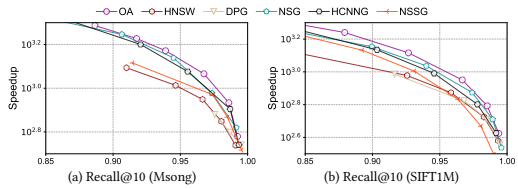


Figure 11: Speedup vs Recall@10 of the optimized algorithm (OA).

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 [50, 82, 105]. Other literatures add quantitative or distributed schemes to cope with data increases [27, 30]. To meet hybrid query requirements, the latest research adds structured attribute constraints to the search process of graph-based algorithms [98, 100].

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 [57], 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.ifs.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-Vedio.* Retrieved October 05, 2019 from http://staff.itee.uq.edu.au/shenht/UQVIDEO/

[6] Kazuo Aoyama, Atsunori Ogawa, Takashi 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*. 1055–1063.

[8] Akhil 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 Ceccarello. 2019. Benchmarking Nearest Neighbor Search: Influence of Local Intrinsic Dimensionality and Result Diversity in Real-World Datasets.. In *EDML@ 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, 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.

[14] 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.

[15] 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.

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

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

[18] 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, Δ, χ and ω. *International Journal of Computational Geometry & Applications* 22, 05 (2012), 439–469.

[19] Antoine Boutet, Anne-Marie Kermarrec, Nupur Mittal, and François Taïani. 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.

[20] Brankica Bratić, 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.

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

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

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

[24] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. 2018. *SPTAG: A library for fast approximate nearest neighbor search.* https://github.com/Microsoft/SPTAG

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

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

[27] Shiyuan Deng, Xiao Yan, KW Ng Kelvin, Chenyu Jiang, and James 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.

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

[29] Wei Dong, Charikar 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.

[30] Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. 2018. Link and Code: Fast Indexing With Graphs and Compact Regression Codes. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. 3646–3654.

[31] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, et al. 1995. Query by image and video content: The QBIC system. *computer* 28, 9 (1995), 23–32.

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

[33] Cong Fu and Deng Cai. 2016. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228* (2016).

[34] Cong Fu, Changxu Wang, and Deng Cai. 2021. High Dimensional Similarity Search with Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[35] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *Proc. VLDB Endow.* 12, 5 (2019), 461–474. https://doi.org/10.14778/3303753.3303754

[36] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized Product Quantization for Approximate Nearest Neighbor Search. In *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*. IEEE Computer Society, 2946–2953.

[37] Long Gong, Huayi Wang, Mitsunori Ogihara, and Jun Xu. 2020. iDEC: Indexable Distance Estimating Codes for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 13, 9 (2020), 1483–1497.

[38] Hakim Hacid and Tetsuya Yoshida. 2010. Neighborhood graphs for indexing and retrieving multi-dimensional data. *Journal of Intelligent Information Systems* 34, 1 (2010), 93–111.

[39] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. 2011. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

[40] Ben Harwood and Tom Drummond. 2016. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5713–5722.

[41] Qiang Huang, Jianlin Feng, Qiong Fang, Wilfred Ng, and Wei Wang. 2017. Query-aware locality-sensitive hashing scheme for l_p norm. *The VLDB Journal* 26, 5 (2017), 683–708.

[42] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.

[43] Masajiro Iwasaki. 2015. *Neighborhood Graph and Tree for Indexing High-dimensional Data.* Yahoo Japan Corporation. Retrieved August 22, 2020 from https://github.com/yahoojapan/NGT

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

[45] 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).

[46] Jerzy W Jaromczyk and Mirosław Kowaluk. 1991. Constructing the relative neighborhood graph in 3-dimensional Euclidean space. *Discrete Applied Mathematics* 31, 2 (1991), 181–191.

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

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

[49] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[50] Jie Ren, Minjia Zhang, and Dong Li. 2020. HM-ANN: Efficient Billion-Point Nearest Neighbor Search on Heterogeneous Memory. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*.

[51] Zhongming Jin, Debing Zhang, Yao Hu, Shiding Lin, Deng Cai, and Xiaofei He. 2014. Fast and accurate hashing via iterative nearest neighbors expansion. *IEEE transactions on cybernetics* 44, 11 (2014), 2167–2177.

[52] Jeff Johnson, Matthijs Douze, Hervé Jégou, and Hosseini Lucas. 2017. *FAISS: Facebook AI Similarity Search.* Retrieved September 13, 2020 from https://github.com/facebookresearch/faiss

[53] Jon Kleinberg. 2000. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. 163–170.

[54] Atsutake Kosuge and Takashi Oshima. 2019. An Object-Pose Estimation Acceleration Technique for Picking Robot Applications by Using Graph-Reusing k-NN Search. In *2019 First International Conference on Graph Computing (GC)*. IEEE, 68–74.

[55] Joseph B Kruskal. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society* 7, 1 (1956), 48–50.

[56] Conglong Li, Minjia Zhang, David G Andersen, and Yuxiong He. 2020. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2539–2554.

[57] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* (2019).

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

[59] Federico Magliani, Kevin McGuinness, Eva Mohedano, and Andrea Prati. 2019. An Efficient Approximate kNN Graph Method for Diffusion on Image Retrieval. In *International Conference on Image Analysis and Processing*. Springer, 537–548.

[60] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.

[61] Yury A Malkov and Alexander Ponomarenko. 2016. Growing homophilic networks are natural navigable small worlds. *PloS one* 11, 6 (2016), e0158162.

[62] Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* (2018).

[63] Luke Mathieson and Pablo Moscato. 2019. An Introduction to Proximity Graphs. In *Business and Consumer Analytics: New Ideas*. Springer, 213–233.

[64] Yitong Meng, Xinyan Dai, Xiao Yan, James Cheng, Weiwen Liu, Jun Guo, Benben Liao, and Guangyong Chen. 2020. PMD: An Optimal Transportation-Based User Distance for Recommender Systems. In *European Conference on Information Retrieval*. Springer, 272–280.

[65] Milvus. 2019. *An Open Source Vector Similarity Search Engine*. Retrieved September 20, 2020 from https://milvus.io/

[66] Javier Vargas Munoz, Marcos A Gonçalves, Zanoni Dias, and Ricardo da S Torres. 2019. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognition* 96 (2019), 106970.

[67] Bilegsaikhan Naidan, Leonid Boytsov, and Eric Nyberg. 2015. Permutation Search Methods are Efficient, Yet Faster Search is Possible. *Proc. VLDB Endow.* 8, 12 (2015), 1618–1629.

[68] Zhibin Pan, Liangzhuang Wang, Yang Wang, and Yuchen Liu. 2020. Product quantization with dual codebooks for approximate nearest neighbor search. *Neurocomputing* (2020).

[69] Rodrigo Paredes and Edgar Chávez. 2005. Using the k-nearest neighbor graph for proximity searching in metric spaces. In *International Symposium on String Processing and Information Retrieval*. Springer, 127–138.

[70] Rodrigo Paredes, Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. 2006. Practical construction of k-nearest neighbor graphs in metric spaces. In *International Workshop on Experimental and Efficient Algorithms*. Springer, 85–97.

[71] Alexander Ponomarenko, Nikita Avrelin, Bilegsaikhan Naidan, and Leonid Boytsov. 2014. Comparative analysis of data structures for approximate nearest neighbor search. *Data analytics* (2014), 125–130.

[72] Liudmila Prokhorenkova and Aleksandr Shekhovtsov. 2020. Graph-based nearest neighbor search: From practice to theory. In *International Conference on Machine Learning*. PMLR, 7803–7813.

[73] DA Rachkovskij. 2018. Index Structures for Fast Similarity Search for Real Vectors. II. *Cybernetics and Systems Analysis* 54, 2 (2018), 320–335.

[74] Philipp M Riegger. 2010. *Literature survey on nearest neighbor search and search in graphs*. Ph.D. Dissertation. Citeseer.

[75] Stewart Russell, Manning Christopher, and Pennington Jeffrey. 2015. *Enron Email Dataset*. Retrieved April 15, 2020 from https://www.cs.cmu.edu/~./enron/

[76] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. 285–295.

[77] Michael Ian Shamos and Dan Hoey. 1975. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*. IEEE, 151–162.

[78] Larissa Capobianco Shimomura and Daniel S Kaster. 2019. HGraph: A Connected-Partition Approach to Proximity Graphs for Similarity Search. In *International Conference on Database and Expert Systems Applications*. Springer, 106–121.

[79] Larissa C Shimomura, Rafael Seidi Oyamada, Marcos R Vieira, and Daniel S Kaster. 2020. A survey on graph-based methods for similarity searches in metric spaces. *Information Systems* (2020), 101507.

[80] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.

[81] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[82] Suhas Jayaram Subramanya, F Devvrit, HV Simhadri, R Krishnawamy, and R Kadekodi. 2019. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In *NeurIPS 2019*. 1771–13781.

[83] Eric S Tellez, Guillermo Ruiz, Edgar Chavez, and Mario Graff. 2021. A scalable solution to the nearest neighbor search problem through local-search methods on neighbor graphs. *Pattern Analysis and Applications* (2021), 1–15.

[84] Godfried Toussaint. 2002. Proximity graphs for nearest neighbor decision rules: recent progress. In *Progress", Proceedings of the 34 th Symposium on the INTERFACE*. Citeseer.

[85] Godfried T Toussaint. 1980. The relative neighbourhood graph of a finite planar set. *Pattern recognition* 12, 4 (1980), 261–268.

[86] Takeaki Uno, Masashi Sugiyama, and Koji Tsuda. 2009. Efficient construction of neighborhood graphs by the multiple sorting method. *arXiv preprint arXiv:0904.3151* (2009).

[87] Javier A Vargas Muñoz, Zanoni Dias, and Ricardo da S. Torres. 2019. A Genetic Programming Approach for Searching on Nearest Neighbors Graphs. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*. 43–47.

[88] Nakul Verma, Samory Kpotufe, and Sanjoy Dasgupta. 2009. Which Spatial Partition Trees are Adaptive to Intrinsic Dimension?. In *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*. 565–574.

[89] Olli Virmajoki and Pasi Franti. 2004. Divide-and-conquer algorithm for creating neighborhood graph for clustering. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, Vol. 1. IEEE, 264–267.

[90] Dilin Wang, Lei Shi, and Jianwen Cao. 2013. Fast algorithm for approximate k-nearest neighbor graph construction. In *2013 IEEE 13th international conference on data mining workshops*. IEEE, 349–356.

[91] Jingdong Wang and Shipeng Li. 2012. Query-driven iterated neighborhood graph search for large scale indexing. In *Proceedings of the 20th ACM international conference on Multimedia*. 179–188.

[92] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. 2015. Learning to hash for indexing big data—A survey. *Proc. IEEE* 104, 1 (2015), 34–57.

[93] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. 2012. Scalable k-nn graph construction for visual descriptors. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1106–1113.

[94] Jingdong Wang, Naiyan Wang, You Jia, Jian Li, Gang Zeng, Hongbin Zha, and Xian-Sheng Hua. 2013. Trinary-projection trees for approximate nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 36, 2 (2013), 388–403.

[95] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. 2017. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence* 40, 4 (2017), 769–790.

[96] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. arXiv:2101.12631 [cs.IR]

[97] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, Vol. 98. 194–205.

[98] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data. *Proc. VLDB Endow.* 13, 12 (2020), 3152–3165.

[99] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[100] Xiaoliang Xu, Chang Li, Yuxiang Wang, and Yixing Xia. 2020. Multiattribute approximate nearest neighbor search based on navigable small world graph. *Concurrency and Computation: Practice and Experience* 32, 24 (2020), e5970.

[101] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. 2006. *Similarity search: the metric space approach*. Vol. 32. Springer Science & Business Media.

[102] Minjia Zhang and Yuxiong He. 2018. Zoom: Ssd-based vector search for optimizing accuracy, latency and memory. *arXiv preprint arXiv:1809.04067* (2018).

[103] Yan-Ming Zhang, Kaizhu Huang, Guanggang Geng, and Cheng-Lin Liu. 2013. Fast kNN graph construction with locality sensitive hashing. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 660–674.

[104] Kang Zhao, Pan Pan, Yun Zheng, Yanhao Zhang, Changxu Wang, Yingya Zhang, Yinghui Xu, and Rong Jin. 2019. Large-Scale Visual Search with Binary Distributed Graph at Alibaba. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2567–2575.

[105] Weijie Zhao, Shulong Tan, and Ping Li. 2020. SONG: Approximate Nearest Neighbor Search on GPU. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1033–1044.

[106] Wan-Lei Zhao. 2018. k-NN graph construction: a generic online approach. *arXiv preprint arXiv:1804.03032* (2018).

[107] Wan-Lei Zhao, Peng-Cheng Lin, and Chong-Wah Ngo. 2019. On the Merge of k-NN Graph. *arXiv preprint arXiv:1908.00814* (2019).

[108] Lei Zhou, Xiao Bai, Xianglong Liu, Jun Zhou, and Edwin R Hancock. 2020. Learning binary code for fast nearest subspace search. *Pattern Recognition* 98 (2020), 107040.

[109] Wenhui Zhou, Chunfeng Yuan, Rong Gu, and Yihua Huang. 2013. Large scale nearest neighbors search based on neighborhood graph. In *2013 International Conference on Advanced Cloud and Big Data*. IEEE, 181–186.

[110] Chun Jiang Zhu, Tan Zhu, Haining Li, Jinbo Bi, and Minghu Song. 2019. Accelerating Large-Scale Molecular Similarity Search through Exploiting High Performance Computing. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 330–333.