Graph Based K-Nearest Neighbor Search Revisited

JIADONG XIE, Dept. of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong, Hong Kong
JEFFREY XU YU, Dept. of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong, Hong Kong
YINGFAN LIU, School of Computer Science and Technology, Xidian University, Xian, China

The problem of k-nearest neighbor (k-NN) search is a fundamental problem to find the exact k nearest neighbor points for a user-given query point q in a d-dimensional large dataset D with n points, and the approximate k-NN (k-ANN) search problem is to find the approximate k-NN. Both are extensively studied to support real applications. Among all approaches, the graph-based approaches have been seen as the best to support k-NN/ANN in recent studies. The state-of-the-art graph-based approach, τ-MG, finds 1-NN, $\bar{p}_1$, over a graph index G_τ constructed for D based on a predetermined parameter τ where the distance between $\bar{p}_1$ and q is less than τ, and finds k-ANN based on the approach taken for 1-NN. There are some main issues in τ-MG and other graph-based approaches. One is that it is difficult to predetermine τ which can ensure to find 1-NN and can do it efficiently. This is because the accuracy/efficiency is related to the size of the graph index G_τ constructed. To achieve high accuracy is at the expense of efficiency. In addition, like all the other existing graph-based approaches, it does not have a theoretical guarantee to ensure k-NN for the same reason to use the same graph index, G_τ, for both 1-NN and k-NN ($k > 1$).

In this article, we propose a new graph-based approach for k-NN with a theoretical guarantee. We construct a labeled graph, $\mathcal{G}$, and we do not need to predetermine τ. Instead, we find 1-NN over a subgraph, $\mathcal{G}_\tau$, of $\mathcal{G}$, virtually constructed in a dynamic manner. Here, τ we use is query-dependent and can be smaller than τ, and the subgraph $\mathcal{G}_\tau$ is smaller than G_τ when $\tau = \tau$. We find k-NN in two phases. In the navigation phase, we find 1-NN, $\bar{p}_1$, of q over $\mathcal{G}_\tau$. In the second refinement phase, for $k > 1$, we explore the neighbors within the vicinity region of $\bar{p}_1$ in $\mathcal{G}$. Based on our solution for k-NN in theory, we propose new algorithms to support k-ANN efficiently in practice. We conduct extensive performance studies and confirm the effectiveness and efficiency of our new approach.

CCS Concepts: • **Information systems** → **Information retrieval query processing;**

Additional Key Words and Phrases: Approximate nearest neighbor search, high-dimensional vector

ACM Reference Format:
Jiadong Xie, Jeffrey Xu Yu, and Yingfan Liu. 2025. Graph Based K-Nearest Neighbor Search Revisited. *ACM Trans. Datab. Syst.* 50, 4, Article 14 (June 2025), 30 pages. https://doi.org/10.1145/3736716

This work was supported by the National Natural Science Foundation of China (No. 62002274).
Authors' Contact Information: Jiadong Xie, Dept. of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong, Hong Kong; e-mail: jdxie@se.cuhk.edu.hk; Jefrey Xu Yu, Dept. of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong, Hong Kong; e-mail: yu@se.cuhk.edu.hk; Yingfan Liu (Corresponding author), School of Computer Science and Technology, Xidian University, Xian, Shaanxi, China; e-mail: liuyingfan@xidian.edu.cn.

ACM 0362-5915/2025/06-ART14
https://doi.org/10.1145/3736716

1 Introduction

The problem of k-nearest neighbor (k-NN) search is a fundamental problem to find k nearest neighbor points for a user-given query point in a high-dimensional Euclidean vector space, and approximate k-NN (k-ANN) search has been extensively studied for real applications including information retrieval [40], recommendation [46], clustering [38], and large pre-trained language models [35]. To support k-NN/ANN with a focus on the efficiency, there are tree-based approaches [10, 11, 28, 63], quantization-based techniques [19, 23, 31, 39, 59], locality sensitive hashing-based approaches [18, 24, 25, 53, 64] and graph-based approaches [12, 16, 17, 22, 34, 37, 43, 50, 66]. Recent studies [4, 36, 37, 47, 56] have consistently demonstrated the superior performance of graph-based approaches in various large-scale applications. In this article, we study graph-based approaches.

All the graph-based approaches find k-NN/ANN over a directed graph $G = (V, E)$ constructed for a d-dimensional dataset D, where V is the set of n points in D, and an edge (u, v) in E represents that v is a near point of u. For a given query point q, it finds 1-NN/ANN by greedy routing along a path toward the nearest node in G starting from a random node in G; and it finds k-NN/ANN in a similar way by maintaining w ($> k$) nearest nodes while searching along paths toward the nearest node in G, and terminates if there are no closer nodes to q that can be found in the set of w nodes.

We summarize the existing approaches together with ours in Table 1, from both theoretical and practical viewpoints, where $\bar{p}_1$ represents the nearest neighbor of the query q and δ represents the Euclidean distance. In theory, the majority of existing graph-based approaches do not have a theoretical guarantee for finding 1-NN (e.g., HNSW [43], SSG [16], and DPG [37]). Some approaches do have theoretical guarantees for 1-NN but are with constraints which are impractical. MRNG [17] provides a theoretical guarantee to find 1-NN of a query point q only if q exists in D (i.e., $\delta(q, \bar{p}_1) = 0$), which is hard to meet. FANNG [22] and τ-MG [50] relax the condition but request the 1-NN, $\bar{p}_1$, of a query point q that exists near q in D within a distance less than a predefined parameter τ (denoted as $\delta(q, \bar{p}_1) < \tau$). It is also hard to predetermine such τ to ensure that 1-NN for any query point q can be found, as it is unknown where query point q can be when we build index. Even more, the greedy routing (can be seen as a special case of beam search where the beam width is 1) of FANNG and τ-MG to search 1-NN is dependent on the predefined τ in index construction, which means the greedy routing can not find $\bar{p}_1$ of query q if $\delta(q, \bar{p}_1) \geq \tau$. Delaunay graph (DG) [34] is the dual graph of the Voronoi diagram, which guarantees to find the nearest neighbor of arbitrary query point (i.e., $\delta(q, \bar{p}_1) < +\infty$) by greedy routing. However, DG becomes a complete graph when the dimension is large [22]. Furthermore, to the best of our knowledge, there is no study on the theoretical guarantee for k-NN for arbitrary query point. In practice, the results in Table 1 demonstrate that τ-MNG exhibits the highest efficiency in finding both $1/k$-ANN among existing approaches with the similar size of indexes when the recall reaches 0.9. MRNG, τ-MNG, and DG, although capable of ensuring the finding of 1-NN with or without certain constraints, necessitates a substantial construction time (exceeding 48 hours) and incurs high search costs. Therefore, no existing study has demonstrated the ability to guarantee the finding of $1/k$-NN in theory while achieving exceptional efficiency at a high recall in practice.

In this article, we propose a new graph-based approach, and we discuss our approach regarding τ-MG [50], as it is the state-of-the-art. The main contributions of this work are summarized below. ❶ We revisit the graph-based approaches [17, 50] for k-NN/ANN, and address four main issues regarding k-NN. (τ-Issue) For 1-NN, it needs a predetermined τ to construct graph G_τ with which it searches 1-NN, $\bar{p}_1$, of a query point q, on the condition that the distance between q and $\bar{p}_1$ is less than τ. We then explain why predetermine such τ is difficult. On the one hand, it can find 1-NN if G_τ constructed is large enough with a large τ, but such a large G_τ will lead to inefficiency. On the other hand, it can find 1-NN efficiently if G_τ constructed is small with a small τ, but it

Table 1. A Comparison of Graph Based Approaches

Methods	Theory				Practice (Recall=0.9 in SIFT1M)		
	Conditions for Guarantee to Hold		Search Algorithm		QPS		Index Size
	1-NN	k-NN ($k > 1$)	1-NN	k-NN ($k > 1$)	1-ANN	10-ANN	
DPG [37]	No error guarantee	No error guarantee	Beam Search	Beam Search	7,370	7,253	130 MB
SSG [16]	No error guarantee	No error guarantee	Beam Search	Beam Search	7,673	7,431	128 MB
HNSW [43]	No error guarantee	No error guarantee	Beam Search	Beam Search	8,728	8,704	180 MB
Vamana [30]	No error guarantee	No error guarantee	Beam Search	Beam Search	8,064	7,912	130 MB
MRNG [17]	When $\delta(q, \bar{p}_1) = 0$	No error guarantee	Greedy Routing	Beam Search	-	-	-
NSG [17]	No error guarantee	No error guarantee	Greedy Routing	Beam Search	8,380	8,024	120 MB
FANNG [22]	When $\delta(q, \bar{p}_1) < \tau$	No error guarantee	Greedy Routing	Beam Search	4,059	3,856	131 MB
τ-MG [50]	When $\delta(q, \bar{p}_1) < \tau$	No error guarantee	Greedy Routing	Beam Search	-	-	-
τ-MNG [50]	No error guarantee	No error guarantee	Greedy Routing	Beam Search	8,451	8,118	130 MB
DG [34]	**When $\delta(q, \bar{p}_1) < +\infty$**	No error guarantee	Greedy Routing	Beam Search	-	-	-
LMG (ours)	**When $\delta(q, \bar{p}_1) < +\infty$**	**Ensure exact result**	**Adaptive Search**	**Refine Search**	-	-	-
ALMG (ours)	No error guarantee	No error guarantee	**Adaptive Beam Search**	**Refine Search**	**9,894**	**9,772**	120 MB

cannot find 1-NN for many query points, i.e., the queries where $\delta(q, \bar{p}_1) > \tau$. ($k$-Issue) For k-NN ($k > 1$), it cannot ensure finding k-NN. (w-Issue) For k-NN, it is difficult to predetermine w to ensure k-NN exists in the set of w nodes found along a path toward 1-NN of a query point q. (G-Issue) It is difficult to use a single graph G_τ constructed for both 1-NN and k-NN ($k > 1$). For 1-NN, it requests G_τ to be small to search 1-NN, $\bar{p}_1$ of q efficiently; for k-NN, it requests G_τ to be large to contain enough neighbor nodes of $\bar{p}_1$ in order to find k-NN. ❷ To deal with the τ-Issue for 1-NN, different from τ, which is a global value used in [50] to construct G_τ, we use a query-dependent τ, denoted as τ, which is not needed to determine in the index construction. Even more, τ is always smaller than τ to find 1-NN for a query q. In other words, we only need a subgraph of G_τ induced from our constructed labeled monotonic graph (LMG), which is virtually generated in a dynamic manner to find 1-NN. ❸ To deal with the G-Issue, for k-NN, We decouple the two different roles used in a single graph, and we propose a two-phase approach over a graph $\mathcal{G}$ we propose in this article. In the first navigation phase (Adaptive Search), we find 1-NN as mentioned in ❷. In the second refinement phase (Refine Search), for $k > 1$, we explore the neighbors within the vicinity region of $\bar{p}_1$ in $\mathcal{G}$ induced from LMG. As a result, as shown in Table 1, our approach can provide theoretical guarantees of finding both 1-NN and k-NN for any arbitrary queries. ❹ As mentioned above, for the k-Issue, we ensure that we can find k-NN in theory. ❺ For the w-Issue, we show that we do not need w in our approach in theory. ❻ Based on our algorithm to find k-NN with a theoretical guarantee, we give algorithms to deal with k-ANN search problem efficiently in practice by constructing an approximate LMG (ALMG). The results in Table 1 demonstrate that our approach consistently achieve the best performance in both 1/k-ANN search, even with comparable or smaller index size. ❼ We further conduct extensive experimental studies, and verify that our approach is more effective and efficient.

The article is organized as follows. We give the preliminaries in Section 2, and discuss the main issues of existing graph-based approaches for k-NN search in Section 3. We propose a new k-NN approach in Section 4 with a theoretical guarantee without any constraint, and discuss a new way to support k-ANN in Section 5. We conduct extensive experimental studies to confirm the effectiveness and efficiency of our approach in Section 6, and discuss the related work in Section 7. We conclude this work in Section 8.

2 Preliminaries

In this article, we study the exact top-k nearest neighbors (k-NN) in theory and the approximate top-k nearest neighbors (k-ANN) in practice in the Euclidean space. Let $\mathbb{R}^d$ denote the

d-dimensional Euclidean space, and $\delta(u, v)$ be the L_2 norm (i.e., Euclidean distance) between two points u and v. Let $ball(u, r)$ be an open ball centered at u with radius r, $lune(u, v)$ be the common area of $ball(u, \delta(u, v))$ and $ball(v, \delta(u, v))$, i.e., $lune(u, v) = ball(u, \delta(u, v)) \cap ball(v, \delta(u, v))$ and use $Vol(ball(u, r))$ to denote the volume of $ball(u, r)$.

Definition 2.1 (k-NN Search Problem). Given a dataset D with n data points in $\mathbb{R}^d$ and a query point q in $\mathbb{R}^d$, the k-NN search problem aims at finding k points $(\bar{p}_1, \ldots, \bar{p}_k)$ in D where $\bar{p}_i$ is the exact i-th nearest neighbor of q in D.

Definition 2.2 (k-ANN Search Problem). Given a dataset D with n data points in $\mathbb{R}^d$, a query point q in $\mathbb{R}^d$ and a constant $\epsilon \geq 0$, the k-ANN search problem aims at finding k points $(p_1, \ldots, p_k)$ in D such that $\forall i \in [1, k], \delta(p_i, q) \leq (1 + \epsilon)\delta(\bar{p}_1, q)$, where $\bar{p}_1$ is the 1-th nearest neighbor of q in D.

We follow the similar assumptions given in [17, 50] for the theoretical analysis. (1) The n points in a given dataset D are uniformly distributed in a finite subspace of $\mathbb{R}^d$, for $d \geq 2$. (2) The point density is $O(\ln n)$. (3) There exists a constant ψ, such that $\psi V_D \geq Vol(Ball(\delta_{max}))$, where V_D denotes the volume of the minimum convex hull containing D and δ_{max} denotes the maximum distance between two points in D. (4) The distribution of user-given query points shares the same distribution as the data points in D.

The state-of-the-art approaches are to construct a **proximity graph** (**PG**), $G = (V, E)$, for a given dataset D, and search over the proximity graph constructed to answer k-NN/ANN. Here, $G = (V, E)$ constructed is a directed graph, where V is a set of nodes representing all n data points in D, and E is a set of edges representing the proximity property such that, for a directed edge $(u, v) \in E$, v represents a point that is near to the point that u represents in D. Below, for simplicity, a graph is a PG graph, and we use p_i and v_i interchangeably as a node v_i in G represents a point p_i in D. Given a query point q, the PG-based approaches start with a random node in G constructed, and search the node nearest to q in G by searching along a monotonic path, which is defined below.

Definition 2.3 (Monotonic Path). A path $P = [v_1, v_2, \ldots, v_{|P|}]$ in G is a monotonic path for a given query q if $\delta(v_i, q) > \delta(v_{i+1}, q)$ for $\forall i \in [1, |P| - 1]$.

The monotonic path defined describes the proximity properties required for G to be constructed. A large number of edges in G will lead to traversing more nodes along a monotonic path to the nearest node of a given query point q in G, which is costly. Hence, to minimize G being constructed, the state-of-the-art approaches employ edge pruning strategies in G if the graph indexes ensure that there exist monotonic paths from any nodes in G reaching out to the nearest node for any query point q. In MRNG [17], an edge $(u, v) \in E(G)$ can be pruned if there exists a node $u' \in lune(u, v)$ such that $(u, u') \in E(G)$. In [50], a τ-MG graph, G_τ, is constructed by extending this pruning strategy in [17].

Definition 2.4 (τ-MG). Given a constant τ, a τ-monotonic graph (τ-MG), $G_\tau = (V, E_\tau)$, is a directed graph. Here, V is the set of nodes representing n points in D, and a directed edge (u, v) exists in E_τ if it satisfies the following conditions.

(i) If $\delta(u, v) \leq 3\tau$, then $(u, v) \in E_\tau$.
(ii) If $\delta(u, v) > 3\tau$, then $(u, v) \notin E_\tau$ iff E_τ contains (u, u') satisfying that $u' \in ball(u, \delta(u, v)) \cap ball(v, \delta(u, v) - 3\tau)$.

As proved in [50], a τ-MG, G_τ, as a PG graph constructed for dataset D, ensures that there exists a monotonic path for any given query point q to its 1-NN $\bar{p}_1$ in G_τ, if the distance between $\bar{p}_1$ and q in D satisfying $\delta(q, \bar{p}_1) < \tau$.

ALGORITHM 1: BeamSearch (G, q, k, w)

```
Input  : PG $G$, query point $q$, $k$ for top-$k$, and beam width $w$
Output : $k$-ANN of $q$
1 Randomly select a node $u \in G$ to insert into the set $S$;
2 while there exists an unvisited node in $S$ do
3     $u \leftarrow$ the nearest unvisited node to $q$ in $S$;
4     Mark $u$ visited;
5     for each unvisited $v$ of $(u, v) \in G$ that is not in $S$ do
6         Insert $v$ into $S$;
7     while $|S| > w$ do
8         Remove the farthest nodes to $q$ from $S$;
9 return the top-$k$ nearest nodes to $q$ in $S$;
```

Lemma 2.1 (Lemma 2 in [50]). *Let G_τ be a τ-MG of the dataset D, and let $\bar{p}_1$ be 1-NN of a given query point q. There exists a monotonic path in G_τ starting from any node in G_τ to $\bar{p}_1$ if $\delta(q, \bar{p}_1) < \tau$.*

Over a PG graph (e.g., τ-MG), G, constructed, a widely used greedy algorithm is presented in Algorithm 1 based on the monotonic path property to find the k-ANN for a query point q [16, 17, 22, 43, 50]. Algorithm 1 takes four inputs: a PG graph G, a query point q, a k value for top-k nearest points, and a beam width w with $w \geq k$. Initially, it randomly selects a node u in G and inserts it into set S, which is unvisited (line 1). In the while-loop (lines 2-8), if there exists an unvisited node in S, the beam search tries to find the nearest point of q in G following a monotonic path as follows. First, it selects the nearest unvisited node, u, to the query point q from S, and marks it visited (lines 3-4). Second, it adds every node, v, into S if there is a directed edge from u to v, and v is not in S which implies that v is unvisited yet (lines 5-6). Third, it deletes the farthest nodes to q from S if $|S|$ (the size of S) is greater than w (lines 7-8). When the algorithm breaks from the while-loop, the results can be obtained from S, i.e., the k nodes with the smallest distance to q (line 9). It is worth mentioning that a larger beam width w leads to higher accuracy to find k-ANN but comes with a higher time cost.

3 The Main Issues of Existing k-NN Search Approaches

In this section, we discuss the main issues of the existing PG-based approaches for the k-NN search problem regarding the theoretical guarantee to find k-NN and the efficiency to find k-NN.

1-NN search: Given a rather large PG graph G constructed for a dataset D, DG [34] can find 1-NN of a query point q in $O(n)$ time, which implies that it needs to explore almost every node in G or every point in the corresponding D. To enhance efficiency, the majority of PG-based approaches (e.g., HNSW [43], SSG [16], DPG [37]) can find 1-ANN efficiently, but they cannot guarantee in theory to find 1-NN following the monotonic path for an arbitrarily given query point q. MRNG [17] guarantees to find 1-NN of a query point q on the condition that the query point q is a point in D ($q \in D$), and there is no guarantee in theory if this condition does not hold. For an arbitrary query point q which does not necessarily have to be a point in D, FANNG [22] and τ-MG [50] can find 1-NN $\bar{p}_1$ of q if $\delta(\bar{p}_1, q) < \tau$ in theory, and can find it efficiently. It is important to note that the condition of $\delta(\bar{p}_1, q) < \tau$ suggests that there must be a point within the distance of τ in D for a given query point q. Otherwise, there is no guarantee to find 1-NN of q.

Here, an issue is how to predetermine τ to find 1-NN in D for an arbitrary query point q. We conducted some tests to explore τ values for the 10,000 given queries over the SIFT1M dataset [1] to find 1-NN. The results are shown in Figure 1, where the x-axis is $\delta(q, \bar{p}_1)$ values, and the

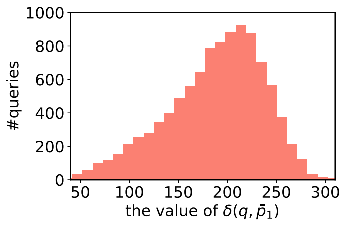


Fig. 1. The distribution of $\delta(q, \bar{p}_1)$ values for 10, 000 queries in SIFT1M dataset.

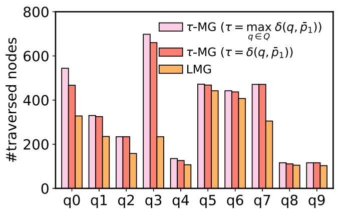


Fig. 2. The number of traversed nodes in SIFT10K dataset across different methods of 1-NN.

y-axis shows the number of queries that need such a $\tau = \delta(q, \bar{p}_1)$ value to find 1-NN. Note that τ should be greater than or equal to $\delta(q, \bar{p}_1)$ to find 1-NN. There are significant variations across different queries. Because τ is the value used to construct a τ-MG graph, G_τ, over which 1-NN is conducted, in theory, τ should be large enough, and need be $\max_{q \in Q} \delta(q, \bar{p}_1)$ for a set of queries Q, if we have known the set of queries Q. And τ might need to be even larger to handle any possible query points. There is a dilemma of choosing a proper τ. On the one hand, such a large τ value required in theory leads to constructing a large τ-MG graph G_τ with a smaller number of edges to be pruned, and therefore ends up with more nodes being traversed along a monotonic path to find 1-NN for a query point q. It cannot be efficient. On the other hand, aiming at finding 1-NN efficiently, it needs a smaller τ to construct a smaller τ-MG G_τ, which ends up impossible for τ to find 1-NN for many queries. In addition, the τ value predetermined to find 1-NN for more queries is at the expense of traversing more nodes along a monotonic path for many other queries. We conduct testing using the 10 given queries, Q, in SIFT10K [1] on two different indexes of τ-MG constructed with different building methods. First, for all 10 queries, we set $\tau = \max_{q \in Q} \delta(q, \bar{p}_1)$ to construct one τ-MG graph G_τ. Second, for each of the 10 queries, q_i, we set $\tau_i = \delta(q_i, \bar{p}_1)$ to construct a specific τ-MG graph, G_{τ_i} which is smaller than G_τ since $\tau_i \leq \tau$ always holds. Figure 2 presents the number of traversed nodes along the monotonic path for each 1-NN query conducted using beam search with $w = 1$. There are main differences. With τ-MG, G_τ, for $\tau \geq \tau_i$ for any query point q_i, it needs to traverse more nodes, which leads to less efficiency. This is because the edge (u, v) will be included in the τ-MG if $\delta(u, v) \leq 3\tau$. There is an issue.

The τ-Issue: As a result, by PG-based approaches, the 1-NN search with the guarantee of finding the nearest neighbor in theory (e.g., τ-MG) cannot be effectively used in practice to find 1-NN due to the efficiency. Therefore, finding 1-NN for an arbitrary query point with a theoretical guarantee remains unsolved in practice.

***k*-NN search:** Different from the 1-NN search problem, for k-NN ($k > 1$), there is no theoretical study to ensure that a τ-MG like graph index can be constructed with which k-NN for a given query point q can be found. We call it k-Issue.
The k-Issue: There is no theoretical study of existing graph-based approaches to find k-NN with guarantee.

From the viewpoint of efficiency, as shown in Algorithm 1, it considers that the k-NN of q exists in S (line 9). This is based on the observation that most routing steps in a PG-based approach are in the neighborhood of the nearest neighbor of q [52, 55, 65]. As a result, the k-NN will be located in the vicinity of top-1 ($\bar{p}_1$), and will be in S to be returned. However, it does not necessarily mean that it can find the k-NN always. We explain it using an example.

Example 3.1. Figure 4 shows an example of finding 2-NN for a query point q over a PG graph G with $n + 1$ nodes. Consider Algorithm 1. Suppose that it finds 1-NN (p_1) along a monotonic path

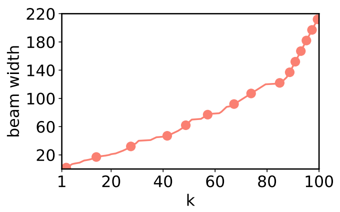


Fig. 3. Minimum required beam width on average for k-NN in SIFT10K.

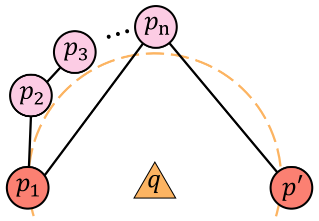


Fig. 4. An example of finding 2-NN search.

first and then widens the beam with w ($w \in [1, n]$), the nodes in S at the final of the beam search will be $S = (p_1, p_2, \ldots, p_w)$, where p_1 is closest and p_n is farthest to q. It is possible that the top-2, p', of q does not exist in S, as there is no edge from p_i to p', for $1 \le i < n$, because $p_n \in lune(p_i, p')$. In this case, it cannot find 2-NN if $w \le n$.

The beam width w should be greater than or equal to k, but as shown in Example 3.1, it is difficult to know how large w has to be. Figure 3 shows the minimum required beam width w on average to find the k-NN using the given 10 queries in SIFT10K. The required beam width increases at a much higher rate while k increases. Here, w is 216.71 on average for $k = 100$. It is important to note that the beam width w is specific for finding k-NN using SIFT10K. It is unknown what beam width it needs for any k-NN in any dataset. This is because k-NN of a query point q are not necessarily the neighbors of 1-NN ($\bar{p}_1$) of the query point q in a graph index. They can be in a significant distance from $\bar{p}_1$ due to the pruning strategies used when the PG graph is constructed.

The w-Issue: It is difficult to know the beam width required in Algorithm 1 to find k-NN over a PG-graph G.

The G-Issue: There exists a conflict over the same PG graph, G, constructed in achieving different goals for 1-NN and k-NN ($k > 1$) search problems. On one hand, for 1-NN search, it needs to construct a small G by pruning some neighbors, and uses it to find 1-NN following a monotonic path with low node degree efficiently. On the other hand, for k-NN search, it requires that the top-k' ($k' \ge k$) nearest points of 1-NN ($\bar{p}_1$) must be in its neighbors in G. The two goals cannot be achieved using a single PG graph index.

4 A New k-NN Approach

In this section, we give a new approach to deal with the four issues, namely, the τ-Issue, the k-Issue, the w-Issue, and the G-Issue, and then give the details.

The τ-Issue: As discussed in Section 3, we reconsider the role of τ used in τ-MG, which is a global value used to to construct a graph G_τ for every query point, which requires (i) prior knowledge of queries and (ii) a value greater than or equal to $\delta(q, \bar{p}_1)$ for every query point q and its 1-NN $\bar{p}_1$. Instead of a global τ, we propose a query-dependent τ, denoted as τ, which is the smallest τ used in τ-MG to find 1-NN for a query q. In other words, we find a monotonic path from a randomly selected node to find 1-NN of q over a subgraph, $\mathcal{G}_\tau$, dynamically generated over an underneath graph, $\mathcal{G}$, which we will explain later. We will show that $|E(\mathcal{G}_\tau)| \le |E(G_\tau)|$. This suggests that we can find 1-NN for an arbitrary query point efficiently, as confirmed in Figure 2 in which LMG (introduced in Section 4) is the number of traversed nodes by our new approach. It is important to mention that we do not need to predetermine the value of τ to construct a PG graph index. In the following, we use τ to indicate the τ is query-dependent, to differ it from the τ used in τ-MG.

The G-Issue: We decouple the two different roles used in a single τ-MG graph G_τ. First, to find 1-NN ($\bar{p}_1$), we dynamically generate a subgraph $\mathcal{G}_\tau$ over a graph $\mathcal{G}$ for a query point q. The graph $\mathcal{G}_\tau$ is virtually constructed and enlarged navigation along the monotonic path iteratively to find 1-NN of q. In the navigation phase, initially, we set a small τ to start, and the $\dot{\tau}$ can be adjusted iteratively, and the $\dot{\tau}$ used is proved to be less than or equal to the best τ used to construct τ-MG graph, G_τ. Second, to find the k-NN for $k > 1$, we explore the neighbors within the vicinity region of $\bar{p}_1$ in $\mathcal{G}$. We will analyze the property of $\mathcal{G}$, and show that the space complexity of $\mathcal{G}$ is the same as that of τ-MG G_τ. For the time of index construction, we will show that our new approach is the same as τ-MG if $\mathcal{G}$ is constructed using the same $\tau = \tau$.

The k-Issue: With the graph $\mathcal{G}$ constructed, as discussed above for the G-Issue, we do the k-NN search in two steps: navigation phase and refinement phase. The navigation phase to find 1-NN $\bar{p}_1$ using $\mathcal{G}_\tau$ for a query point q. The refinement phase is to explore the points within the vicinity region of $\bar{p}_1$ to find k-NN of q. We estimate an upper bound of the distance between the top-k nearest neighbor $\bar{p}_k$ of q and the top-1 nearest neighbor $\bar{p}_1$. Subsequently, we traverse the neighbors of $\bar{p}_1$ within this upper distance bound, and tighten the bound iteratively until meeting the termination condition. This iterative process can obtain the desired result for the k-NN search problem. It is important to note that we can find k-NN in theory, whereas τ-MG cannot always ensure it in theory.

The w-Issue: As shown in Algorithm 1, the beam width w needs to be given as input, and it is difficult to know the smallest w to find k-NN for any query point q beforehand. In the refinement phase of our approaches, we explore nodes within the vicinity region of $\bar{p}_1$ in $\mathcal{G}$. As discussed in the k-Issue, there is no need for w in our algorithm for finding k-NN. Instead, we utilize an upper bound for $\delta(q, \bar{p}_k) - \delta(q, \bar{p}_1)$ to determine the termination condition for achieving a theoretical guarantee. Figures 1 and 6 show that the value of $\delta(q, \bar{p}_k) - \delta(q, \bar{p}_1)$ is significantly smaller than $\delta(q, \bar{p}_1)$, indicating that the time required for the refinement phase is much less than that of the navigation phase.

4.1 A Labeled Monotonic Graph (LMG)

We define a new PG, and call it a LMG.

Definition 4.1 (LMG). Given a dataset D with n points, an LMG (Labeled Monotonic Graph), $\mathcal{G}$, is a complete labeled directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Lambda)$ defined as follows.

- $\mathcal{V} = D$, is a set of n nodes.
- $\mathcal{E} = \{(u, v) | u, v \in \mathcal{V} \wedge u \neq v\}$, is a set of edges, and $\mathcal{E} = \mathcal{E}_0 \cup \mathcal{E}_{>0}$ for $\mathcal{E}_0 \cap \mathcal{E}_{>0} = \emptyset$, where
  - $\mathcal{E}_{>0}$ is a set of edges $\{(u, v)\}$ iff there exists $u' \in \mathcal{V} \setminus \{u, v\}$ such that $(u, u') \in \mathcal{E}_0$ and $u' \in ball(u, \delta(u, v)) \cap ball(v, \delta(u, v))$;
  - $\mathcal{E}_0 = \mathcal{E} \setminus \mathcal{E}_{>0}$.
- $\Lambda = \{\lambda(u, v) \mid (u, v) \in \mathcal{E}\}$, is a set of edge labels.
  - The label $\lambda(u, v) \in \Lambda$ for an edge $(u, v) \in \mathcal{E}_0$ is defined as $\lambda(u, v) = 0$.
  - The label $\lambda(u, v) \in \Lambda$ for an edge $(u, v) \in \mathcal{E}_{>0}$ is defined as $\lambda(u, v) = (\delta(u, v) - \Delta(u, v))/3$, where $\Delta(u, v) = \min_{w \in N_<(u,v)} \delta(v, w)$ for $N_<(u, v) = \{u' \mid (u, u') \in \mathcal{E}_0 \wedge \delta(u, u') < \delta(u, v)\}$.

We illustrate the label $\lambda(u, v) \in \Lambda$ for an edge (u, v) in $\mathcal{E}_{>0}$ in Figure 5. The gray area represents the range of nodes in $N_<(u, v)$. The edge from u to any of nodes in $N_<(u, v)$ is in $lune(u, v)$, and is included in $\mathcal{E}_0$ with which the edge of (u, v) is pruned. Let w in $N_<(u, v)$ be the node with the smallest distance to v, $\Delta(u, v)$ represents this smallest distance of $\Delta(u, v) = \delta(v, w)$. Based on $\Delta(u, v)$ and the distance between u and v, the label $\lambda(u, v)$ is obtained. Note that the LMG subgraph, $\mathcal{G}_0 = (\mathcal{V}, \mathcal{E}_0)$, is identical to the τ-MG graph G_τ when $\tau = 0$ (Definition 2.4), and LMG $\mathcal{G}$ is identical

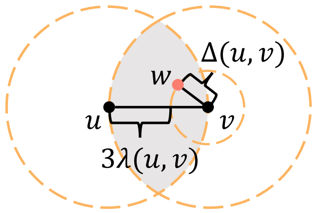


Fig. 5. An illustration of $\Delta(u,v)$ and $\lambda(u,v)$.

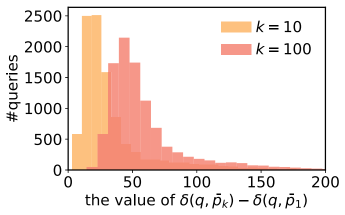


Fig. 6. The distribution of $\delta(q,\bar{p}_k) - \delta(q,\bar{p}_1)$ in SIFT1M.

to τ-MG G_τ when $\tau = \infty$. We will discuss the space issues later on. The key difference between a τ-MG G_τ and an LMG $\mathcal{G}$ is as follows. For G_τ, the graph size is tightly related to the search cost, whereas, for $\mathcal{G}$, the graph size is not related to the search cost, and we only need to search over a subgraph of $\mathcal{G}$ which is defined blow.

Definition 4.2 ($\dot{\tau}$-LMG). A $\dot{\tau}$-LMG graph, $\mathcal{G}_{\dot{\tau}} = (\mathcal{V}_{\dot{\tau}}, \mathcal{E}_{\dot{\tau}}, \Lambda)$, for a given τ, is a subgraph of $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Lambda)$ such that $\mathcal{V}_{\dot{\tau}} = \mathcal{V}$ and $\mathcal{E}_{\dot{\tau}} = \{(u,v) \in \mathcal{E} \mid \lambda(u,v) \le \dot{\tau}\}$.

Based on Definition 4.2, the following lemma is obvious.

Lemma 4.1. *$G_{\dot{\tau}_1}$ is a subgraph of G_{τ_2} if $\dot{\tau}_1 \leq \dot{\tau}_2$.*

Next, we give a lemma to show the relationship between a τ-MG G_τ (Definition 2.4) and a $\dot{\tau}$-LMG $\mathcal{G}_{\dot{\tau}}$ (Definition 4.2) when $\tau = \dot{\tau}$.

Lemma 4.2. *A $\dot{\tau}$-LMG, $\mathcal{G}_{\dot{\tau}} = (\mathcal{V}_{\dot{\tau}}, \mathcal{E}_{\dot{\tau}}, \Lambda)$, is identical to a τ-MG graph, $G_\tau(V_\tau, E_\tau, \Lambda)$, if $\dot{\tau} = \tau$.*

Proof Sketch. Note that $V_\tau = \mathcal{V}_{\dot{\tau}}$. We prove it regarding edges.

First, we show $G_\tau \subseteq \mathcal{G}_\tau$. For every edge (u,v) in E_τ, there are two cases in Definition 2.4. (i) When $\delta(u,v) \le 3\tau$, we have $\lambda(u,v) \le \delta(u,v)/3 \le \tau = \dot{\tau}$. Thus (u,v) will appear in $\mathcal{E}_{\dot{\tau}}$. (ii) When $\delta(u,v) > 3\tau$, there is no edge $(u,u\ )$ in E_τ, where $u' \in ball(u,\delta(u,v)) \cap ball(v,\delta(u,v) - 3\tau)$. Thus, we have

$$\lambda(u,v) = (\delta(u,v) - \Delta(u,v))/3 < (\delta(u,v) - (\delta(u,v) - 3\tau))/3 = \tau = \dot{\tau}.$$

This suggests that (u,v) will appear in $\mathcal{E}_{\dot{\tau}}$. Therefore, we have $E_\tau \subseteq \mathcal{E}_{\dot{\tau}}$.

Second, we show $\mathcal{G}_{\dot{\tau}} \subseteq G_\tau$. For every edge (u,v) in $\mathcal{E}_{\dot{\tau}}$, we prove it in two cases. (i) If $\delta(u,v) \le 3\dot{\tau} = 3\tau$, then

$$\lambda(u,v) = (\delta(u,v) - \Delta(u,v))/3 \le \delta(u,v)/3 \le 3\dot{\tau}.$$

This means that (u,v) appears in both E_τ and $\mathcal{E}_\tau$ since $\tau = \tau$. (ii) For any edge (u,v) that does not satisfy the condition of (a), we have

$$\delta(u,v) > 3\dot{\tau} \text{ and } \lambda(u,v) = (\delta(u,v) - \Delta(u,v))/3 \le \dot{\tau}.$$

Thus, we have

$$\Delta(u,v) \ge \delta(u,v) - 3\dot{\tau} = \delta(u,v) - 3\tau.$$

This means that there is no node u' satisfying that $u' \in ball(v,\delta(u,v) - 3\tau)$ and $u' \in N_<(u,v)$. Hence, (u,v) will appear in E_τ. Therefore, $\mathcal{E}_{\dot{\tau}} \subseteq E_\tau$. □

As the number of nodes traversed along a monotonic path in $\mathcal{G}_{\dot{\tau}}$ is a dominating factor in k-NN search, or specifically in 1-NN, we study the expected node degree in $\mathcal{G}_{\dot{\tau}}$, where the node degree of a node u in $\mathcal{G}$ is denoted as $\deg(u) = |\{v \mid (u,v) \in \mathcal{G}_{\dot{\tau}}\}|$.

Lemma 4.3. *Given a graph $\mathcal{G}_\tau = (\mathcal{V}_{\dot\tau}, \mathcal{E}_{\dot\tau}, \Lambda)$, for a given $\dot\tau$, over a dataset D in $\mathbb{R}^d$. The expected node degree, $\mathbb{E}[\deg(u)]$, is $O((\dot\tau + \dot\tau^d)\ln n)$, where $n = |\mathcal{V}_{\dot\tau}|$.*

Proof Sketch. For $\mathcal{G}_\tau$ when $\dot\tau = 0$, it is identical to τ-MG, G_τ, when $\tau = 0$. In such a case, it is proved in [17, 26] that the maximum node degree of the τ-MG graph G_0 is independent of n and is treated as a constant when we assume d is a constant. We omit the discussion. We discuss the $\dot\tau$ values when $\dot\tau > 0$ below. Note that every edge (u, v) in $\mathcal{G}_{\dot\tau}$ is with $\lambda(u, v) > 0$. There must exist $w \in lune(u, v)$ and $(u, w) \in \mathcal{G}_0$ by Definition 4.2, i.e., $u \in N_<(u, v)$. For $\lambda(u, v) \le \tau$, we have $\forall w \in N_<(u, v), \delta(v, w) \ge \delta(u, v) - 3\dot\tau$. Therefore, the probability of $\lambda(u, v) \le \tau$ is

$$Pr(u,v) = \frac{Vol(lune(u,v) \setminus ball(v, \delta(u,v) - 3\dot\tau))}{Vol(lune(u,v))},$$

due to the uniform distribution assumption used in the literature. Suppose w is the intersect point of line u–v and $ball(v, \delta(u, v) - 3\tau)$. As w may not be a point in the dataset D (or a node in $\mathcal{G}$), we have

$$Pr(u,v) < \frac{Vol(lune(u,v)) - Vol(lune(w,v))}{Vol(lune(u,v))} = 1 - \left(1 - \frac{3\dot\tau}{\delta(u,v)}\right)^d.$$

Next, we discuss $\delta(u, v)$ in $Pr(u, v)$. When $\delta(u, v) \le 3\dot\tau$, the corresponding edge (u, v) must be in G_ρ. Thus, $Pr(u, v) = 1$, and the expected node degree of u is $\gamma \cdot Vol(u, 3\dot\tau) = O(\gamma\dot\tau^d)$, where γ is the density. Otherwise, when $\delta(u, v) > 3\dot\tau$, by Bernoulli's inequality, we have

$$1 - \left(1 - \frac{3\dot\tau}{\delta(u,v)}\right)^d \le \frac{d3\dot\tau}{\delta(u,v)}.$$

Then, let $x = \delta(u, v)$, for x in the range of $[3\dot\tau, \delta_{max}]$ where δ_{max} is the max distance between any two points in the dataset D, we have the expected node degree of u,

$$\mathbb{E}[\deg(u)] \le 3d\dot\tau \int_{3\tau}^{\delta_{max}} \frac{1}{x} dx \le 3d\dot\tau(\ln\delta_{max} - \ln 3\dot\tau).$$

Next, based on the assumptions given in the preliminary used in the literature, δ_{max} can be replaced as follows. Here, since the points in the dataset D are uniformly distributed and the density of points is γ, we have $\gamma \cdot V_D = n$. As there exists ψ such that $\psi V_D \ge Vol(ball(\delta_{max}))$, we have

$$Vol(ball(\delta_{max})) \le \psi V_D = \psi\frac{n}{\gamma}.$$

Let $b = \frac{\pi^{d/2}}{\Gamma(1+d/2)}$, we have $\delta_{max} \le (\frac{\psi n}{\gamma b})^{1/d}$. As b, ψ, and d can all be treated as constants, by combining the results for $\delta(u, v) \le 3\dot\tau$, we have

$$\mathbb{E}(\deg(u)) \le O\left(\dot\tau^d\gamma + \dot\tau\ln\delta_{max}\right) = O\left(\left(\dot\tau + \dot\tau^d\right)\ln n\right). \qquad \square$$

We discuss ours, $\mathbb{E}[\deg(u)] = O((\dot\tau + \dot\tau^d)\ln n)$ for $\mathcal{G}_\tau$, and the one, $\mathbb{E}[\deg(u)] = O(\ln n)$, for τ-MG, G_τ given in [50]. For G_τ, it treats τ as a constant, because G_τ is constructed with a global τ value for any query. Therefore, the parameter τ does not need to appear in $\mathbb{E}[\deg(u)]$. If it takes τ as a parameter, it will be the same as the one we show in Lemma 4.3. This is because $\mathcal{G}_{\dot\tau}$ and G_τ are identical, as proven in Lemma 4.2. Both are the same in size. Here, G_τ needs to be sufficiently large to find the 1-NN $\bar p_1$ for any query q. Otherwise, it cannot, if $\delta(q, \bar p_1) > \tau$. For $\mathcal{G}_{\dot\tau}$, it is virtually constructed from LMG in 1-NN search for a query q, and the query-dependent τ used is the smallest one to ensure $\delta(q, \bar p) \le \tau$ to be found in $\mathcal{G}_{\dot\tau}$. Note that $|E(\mathcal{G}_{\dot\tau})| \le |E(G_\tau)|$ in finding 1-NN of q. As shown in Figure 2, the number of nodes traversed by LMG is smaller than that by τ-MG.

LMG Construction: We construct an LMG graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Lambda)$, for a dataset D by ConstructLMG (Algorithm 2). Here, $\mathcal{V} = D$. Based on Definition 4.1, ConstructLMG adds edges

ALGORITHM 2: ConstructLMG (D)

```
Input  : a dataset $D$
Output : an LMG graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Lambda)$
1  $\mathcal{V} = D, \mathcal{E}_0 \leftarrow \emptyset$;
2  for each $u \in \mathcal{V}$ do
3      for each $v \in \mathcal{V} \setminus \{u\}$ with ascending order of $\delta(u, v)$ do
4          if $\nexists (u, w) \in \mathcal{E}_0$ with $\delta(v, w) \leq \delta(u, v)$ then
5              $\mathcal{E}_0 \leftarrow \mathcal{E}_0 \cup \{(u, v)\}$ with $\lambda(u, v) = 0$;
6  $\mathcal{E} \leftarrow \mathcal{E}_0$;
7  for each edge $(u, w) \in \mathcal{E}_0$ do
8      for each $v \in \mathcal{V}$ do
9          if $(u, v) \notin \mathcal{E}$ then
10             $\mathcal{E} \leftarrow \mathcal{E} \cup \{(u, v)\}$;
11         if $\delta(u, v) > \delta(u, w)$ then
12             $\lambda(u, v) \leftarrow \max\{\lambda(u, v), (\delta(u, v) - \delta(v, w))/3\}$;
13 return $\mathcal{G}$
```

into $\mathcal{E}$ in two loops. In the first loop, it adds/labels edges in $\mathcal{E}_0$, and in the second loop, it adds/labels edges in $\mathcal{E}_{>0}$. It is worth mentioning that the graph $\mathcal{G}$ constructed by Algorithm 2 may be seen large. We can reduce $\mathcal{G}$ to the same size of τ-MG, G_τ, by deleting edges, (u, v), from $\mathcal{G}$, if $\lambda(u, v) > \tau$. With the LMG graph reduced, we can ensure that we can find 1-NN if BeamSearch can find 1-NN with G_τ.

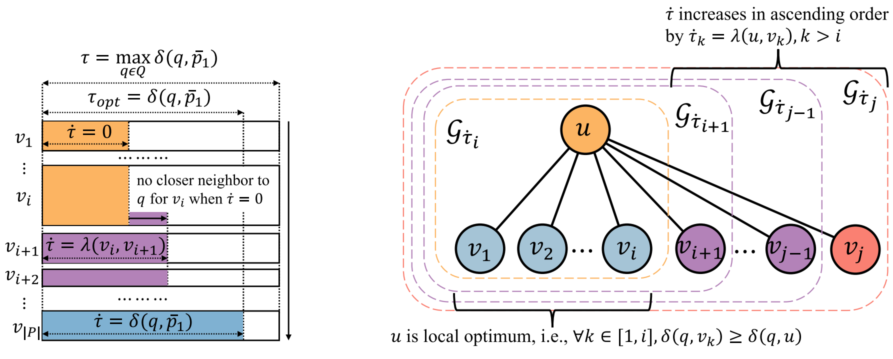
Lemma 4.4. *Given a dataset D with n points,* ConstructLMG *(Algorithm 2) constructs an LMG graph in $O(n^2 \log n)$.*

Proof Sketch. In ConstructLMG, it deals with edges in $\mathcal{E}_0$ in the first loop (lines 2-5). As we know, $\mathcal{E}_0$ is the edge set of G_τ when $\tau = 0$, hence, as proved in [17, 50], the maximum degree of nodes in G_τ is independent of n and is treated as a constant when we assume d is a constant. Then, at line 3, each v should be traversed with ascending order of $\delta(u, v)$, which needs $O(n \log n)$ to sort. Next, for each pair of points (u, v) (line 3) will check if $(u, w) \in \mathcal{E}_0$ satisfies $\delta(v, w) < \delta(u, v)$, which needs $O(deg(u))$, is treated as a constant. Hence, lines 2-5 needs $O(n^2 \log n)$ totally.

As analyzed previously, the number of edges in $\mathcal{E}_0$ traversed in line 7 is $|E(G_0)| = O(n)$, since the maximum degree of nodes in G_0 is treated as a constant when we assume d is a constant. Next, for each edge (line 7), we at most use it to deal with n edges (line 8), thus the time complexity of the second loop (lines 7-12) is $O(n^2)$. □

4.2 The Navigation Phase in k-NN Search

We discuss the navigation phase to find 1-NN $\bar{p}_1$ of a query point q, using $\mathcal{G}_\tau$ virtually constructed over LMG $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Lambda)$ in an iterative manner. In a similar way like τ-MG, we find 1-NN of q, $\bar{p}_1$, if $\delta(q, \bar{p}_1) \leq \tau$, in the navigation phase. In other words, the termination condition of navigation is $\delta(q, \bar{p}_1) \leq \tau$. As shown in Figure 7(a), the navigation starts from a randomly selected node v_1. Given such v_1, initially, a graph $\mathcal{G}_{\tau_0} = (\mathcal{V}_{\tau_0}, \mathcal{E}_{\tau_0}, \Lambda)$ is virtually constructed for $\tau_0 = 0$, as an induced graph over $\mathcal{V}_{\tau_0} = \{v_1\} \cup N_{v_1}$ where $N_{v_1} = \{u \mid (v_1, u) \in \mathcal{E} \wedge \lambda(v_1, u) \leq \tau_0\}$. In navigation, such an initial $\mathcal{G}_{\tau_0}$ will grow as shown in Figure 7(b). Suppose that, currently, it is $\mathcal{G}_{\tau_i}$ with which we navigate and try to decide if a node u is the nearest neighbor $\bar{p}_1$ of q. If it cannot terminate, we enlarge $\mathcal{G}_{\tau_i}$ in one of two ways. (a) We enlarge $\mathcal{G}_{\tau_i}$ while keeping the current τ_i unchanged. (b)


$\tau = \max_{q \in Q} \delta(q, \bar{p}_1)$
$\tau_{opt} = \delta(q, \bar{p}_1)$
v_1
$\dot{\tau} = 0$
v_i
no closer neighbor to q for v_i when $\dot{\tau} = 0$
v_{i+1}
$\dot{\tau} = \lambda(v_i, v_{i+1})$
v_{i+2}
$v_{|P|}$
$\dot{\tau} = \delta(q, \bar{p}_1)$
$\dot{\tau}$ increases in ascending order by $\dot{\tau}_k = \lambda(u, v_k), k > i$
$\mathcal{G}_{\dot{\tau}_i}$
$\mathcal{G}_{\dot{\tau}_{i+1}}$
$\mathcal{G}_{\dot{\tau}_{j-1}}$
$\mathcal{G}_{\dot{\tau}_j}$
u
v_1 v_2 v_i v_{i+1} v_{j-1} v_j
u is local optimum, i.e., $\forall k \in [1, i], \delta(q, v_k) \geq \delta(q, u)$

(a) τ-MG vs LMG in navigation (b) Dynamically adjust $\dot{\tau}$

Fig. 7. The navigation phase.

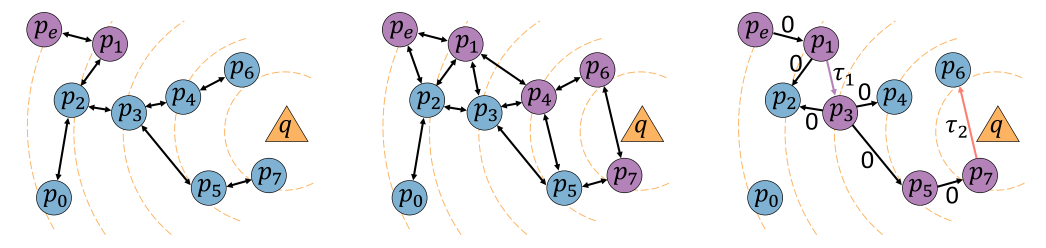
We enlarge $\mathcal{G}_{\dot{\tau}_i}$ to $\mathcal{G}_{\dot{\tau}_{i+1}}$ by enlarging $\dot{\tau}_i$ to $\dot{\tau}_{i+1}$, for $\dot{\tau}_i < \dot{\tau}_{i+1}$, as it reaches the local optimum in $\mathcal{G}_{\dot{\tau}_i}$ and cannot traverse further to reach the optimum in the same $\dot{\tau}$. The decision on whether (a) or (b) depends on a node w, which is closest to the query point q in N_u. If $\delta(w, q) < \delta(u, q)$, it is the case of (a), and we enlarge $G_{\dot{\tau}_i}$ to include the nodes of N_w and the edges from w to the nodes in N_w. If $\delta(w, q) \geq \delta(u, q)$, it is the case of (b), and it suggests that u is the local optimum, and it cannot go further in the graph $\mathcal{G}_{\tau_i}$. We try to find $\dot{\tau}_{i+1}$ which is greater than $\dot{\tau}_i$ but is smallest from those neighbor nodes of u that do not appear in $\mathcal{G}_{\tau_i}$. Then, we enlarge $\mathcal{G}_{\dot{\tau}_i}$ to be $\mathcal{G}_{\dot{\tau}_{i+1}}$ with new $N_u = \{v \mid (u, v) \in \mathcal{E} \wedge \lambda(u, v) \leq \dot{\tau}_{i+1}\}$, and the edges from u to those in N_u. Note that, as shown in Figure 7(b), for the current u, it may enlarge $\mathcal{G}_{\dot{\tau}}$ several times.

The key of our approach is in the elimination of the need for a predefined constant τ to construct a graph G_τ in [50]. Instead, we automatically adjust a query-dependent $\dot{\tau}$ from small to large to enlarge $\mathcal{G}_{\dot{\tau}}$ during the search process.

We give our adaptive algorithm, called AdaptiveSearch, by adjusting $\dot{\tau}$ from small to large in Algorithm 3. AdaptiveSearch takes input of $\mathcal{G}$ and q, and output 1-NN $\bar{p}_1$ of q. It is important to note that the graphs $\mathcal{G}_{\dot{\tau}_i}$ are virtually constructed for us to analyze. As can be seen below, we do not need to extract such $\mathcal{G}_{\dot{\tau}_i}$ entirely. Initially, let $\dot{\tau} = 0$, the navigation starts from a node u randomly selected (lines 1-2). It navigates to the nearest neighbor while the termination condition does not hold ($\delta(u, q) > \tau$) in lines 3-13. With u as the current closest node to q, it tries to find a node, w, which is closer to q in the set of neighbors of u, N_u, where edges from u to a node v in N_u is with label $\lambda(u, v) < \dot{\tau}$ (lines 4-5). If $\delta(w, q) \geq \delta(u, q)$ (line 6), it can be the local optimum or the optimum. To check if it is the optimum, it checks every neighbor v of u beyond N_u in ascending order of $\lambda(u, v)$ which is larger than the current $\dot{\tau}$ (lines 7-12). $\dot{\tau}$ is updated during the checking process (line 8), and when it finds such v closer to q, it routes to v and continues the search. During the process, if it finds $\delta(u, q) \leq \dot{\tau}$, then the current u is optimum to be returned. Otherwise, it lets u be the node which is currently most close to q (line 13), and continues the navigation.

Next, we can explain the issues of using G_τ, and our way of using $\mathcal{G}_{\tau_i}$, using examples.

Example 4.1. Figure 8(a) shows a τ-MG graph, G_{τ_0}, constructed for a given small τ_0 where $\tau_0 < \delta(q, \bar{p}_1)$. Note that, the edge p_e to p_2 is pruned because there exist edges from p_e to p_1 and from p_1 to p_2, with $\delta(p_e, p_1) < \delta(p_e, p_2)$ and $\delta(p_1, p_2) < \delta(p_e, p_2)$. Suppose that it starts from p_e to find 1-NN of q by greedy routing. When it finds p_1, it falls in the local optimum, following the termination condition, as there is no neighbor node of p_1 in G_{τ_0} that can be closer to q. Obviously, p_1 is not the exact 1-NN. This is because τ_0 used to construct G_{τ_0} is not sufficiently large for finding 1-NN.



(a) 1-NN search on G_τ with $\tau < \delta(q, \bar{p}_1)$

(b) 1-NN search on G_τ with $\tau \geq \delta(q, \bar{p}_1)$

(c) Adaptive search on $\mathcal{G}_\tau$ over G_τ of Fig. 8(b)

Fig. 8. τ-MG vs. LMG.

Then, suppose a τ-MG graph, G_τ, constructed for a given large τ ($> \tau_0$), as shown in Figure 8(b). Here, G_τ is larger than G_{τ_0} with more edges added, e.g., the edge from p_e to p_2. By 1-NN search, it starts from p_e to find 1-NN p_7 of q along the monotonic path of $[p_e, p_1, p_4, p_6, p_7]$.

Next, suppose the LMG graph, $\mathcal{G}$, is the same of G_τ. We show the navigation using Figure 8(c), the value of λ is marked near the edges. We start navigation from the same p_e with $\tau = 0$. The initial graph $\mathbb{G} = (\mathbb{V}_0, \mathbb{E}_0)$, where $\mathbb{V}_0 = \{p_e, p_0, p_1\}$ and $\mathbb{E}_0 = \{(p_e, p_0), (p_e, p_1)\}$ under the condition that both $\lambda(p_e, p_0)$ and $\lambda(p_e, p_1)$ are equal to 0. As p_1 is closer to q, it enlarges $\mathbb{G}$ with the node p_2 and the edge (p_1, p_2) while keeping $\tau = 0$ unchanged. In this stage, as p_1 is the local optimum (line 6 in Algorithm 3), it enlarges $\tau = 0$ to $\tau = \tau_1$ ($\lambda(p_1, p_3) = \tau_1$) with the node p_3 and the edge of (p_1, p_3) are added to $\mathbb{G}$ (line 8 in Algorithm 3). Here, we find that p_3 is closer to q (lines 9-10 in Algorithm 3). It enlarges $\mathbb{G}$ with the neighbor nodes of p_3, namely, $\{p_4, p_5\}$, and the edges $\{(p_3, p_2), (p_3, p_4), (p_3, p_5)\}$, since the values of λ of these edges are all $\leq \dot{\tau} = \tau_1$. Then, p_5 is closer to q, it enlarges $\mathbb{G}$ with the node p_7 and the edge (p_5, p_7), then routes to p_7. As $\tau = \tau_1 < \delta(q, p_7)$, it is unclear whether p_7 is the local optimum or the optimum, it needs to enlarge $\tau = \tau_1$ to $\tau = \tau_2$ with which we have $\lambda(p_6, p_7) = \tau_2$ to include the node p_6 and the edge (p_7, p_6), for $\tau_2 > \tau_1$. With $\tau_2 \geq \delta(q, v_7)$, we can find that p_7 is the optimum as it meets the termination condition (lines 11-12 in Algorithm 3).

It is worth mentioning, by 1-NN search on τ-MG, it needs to traverse almost all the nodes in G_τ, in Figure 8(b). We need to traverse a small number of nodes as shown in Figure 8(c).

We further show the difference between the search process on τ-MG and AdaptiveSearch (Algorithm 3) in Figure 7(a). On τ-MG, it needs to navigate over G_τ, where τ needs to be $\max_{q \in Q} \delta(q, \bar{p}_1)$ for any query point $q \in Q$. By AdaptiveSearch, we start with a small τ and enlarge $\mathcal{G}_\tau$, and the τ we used to find the nearest point $\bar{p}_1$ of q can be less than τ, and is query-dependent, i.e., the final value of $\tau \leq \delta(q, \bar{p}_1) \leq \tau$. The correctness of AdaptiveSearch (Algorithm 3) is proved below.

Theorem 4.1. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Lambda)$ be an LMG constructed for a dataset D.* AdaptiveSearch *(Algorithm 3) finds the exact nearest point $\bar{p}_1$ of a query point q (i.e., $\bar{p}_1 = \arg\min_{v \in \mathcal{V}} \delta(q, v)$).*

Proof Sketch. AdaptiveSearch navigates with $\tau < \delta(q, \bar{p}_1)$ first. It may be true at line 6, which suggests that there is no closer neighbor of u in G_τ to q. In such a case, we have $\tau < \delta(q, \bar{p}_1) \leq \delta(u, q)$. Then, there are two cases, either (i) AdaptiveSearch increases τ to continue (lines 9-10), or (ii) for τ updated, it finds the exact nearest neighbor based on the termination condition of $\delta(q, \bar{p}_1) \leq \delta(q, u) \leq \tau$ (lines 11-12). In case (i), it may repeat until τ is sufficiently large to continue to find a node which is closer to q or terminate it. The algorithm terminates when it meets the termination condition (Lemma 2.1). □

ALGORITHM 3: AdaptiveSearch ($\mathcal{G}, q$)

```
Input  : A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Lambda)$ and a query point $q$
Output : the nearest neighbor $\bar{p}_1$ of $q$
1  $\tau \leftarrow 0$;
2  Randomly select a node $u \in \mathcal{V}$;
3  while $\delta(u, q) > \tau$ do
4      $N_u \leftarrow \{v \mid (u, v) \in \mathcal{E} \wedge \lambda(u, v) \leq \tau\}$;
5      $w \leftarrow \arg\min_{v \in N_u} \delta(v, q)$;
6      if $\delta(w, q) \geq \delta(u, q)$ then
7          for each $v \in \{u' \mid (u, u') \in \mathcal{E}\} \setminus N_u$ in ascending order of $\lambda(u, v)$ do
8              $\tau \leftarrow \lambda(u, v)$;
9              if $\delta(v, q) < \delta(u, q)$ then
10                 $w \leftarrow v$; break;
11             if $\delta(u, q) \leq \tau$ then
12                 return $u$
13     $u \leftarrow w$;
14 return $u$
```

It is important to note that τ used in AdaptiveSearch can be less than $\delta(q, \bar{p})$ in navigation before reaching the 1-NN. Note that, when AdaptiveSearch terminates, to confirm the termination condition, the last τ may be greater than $\delta(q, \bar{p})$. We prove it below.

Lemma 4.5. *Given LMG, $\mathcal{G}$, the τ value used in an iteration with a focus on u in* AdaptiveSearch *to find 1-NN of a query point q is less than or equal to $\delta(q, \bar{p}_1)$ when $u \neq \bar{p}_1$.*

Proof Sketch. As Lemma 2.1 shows that G_τ has a monotonic path from any node to $\bar{p}_1$ when $\tau = \delta(q, \bar{p}_1)$. In AdaptiveSearch, at each node u ($\neq \bar{p}_1$) in navigation, $\tau = \max_{(u,v) \in \mathcal{G}_{\delta(q, \bar{p}_1)}} \lambda(u, v)$. This means that $\tau \leq \delta(q, \bar{p}_1)$ always holds. □

Based on Lemma 4.5, the node degree of finding 1-NN by AdaptiveSearch is smaller than the search on τ-MG. We also give the monotonic path length by AdaptiveSearch in the worst-case.

Lemma 4.6. *The expected length of a monotonic path by* AdaptiveSearch *(Algorithm 3) is at most $O(n^{2/d} \ln n)$.*

Proof Sketch. Let the monotonic path be $P = [v_1, v_2, \ldots, v_{|P|}]$ by AdaptiveSearch for a query point q in $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Lambda)$. For $i < |P| - 1$, we have $(v_i, \bar{p}_1) \notin \mathcal{E}$. This means that

$$\exists u, u \in ball(\bar{p}_1, \delta(v_i, \bar{p}_1 - 3\tau)) \text{ where } \tau \geq 0, \text{ i.e., } \delta(u, \bar{p}_1) \leq \delta(v_i, \bar{p}_1) - 3\tau.$$

Thus,

$$\begin{aligned}\delta(q, v_i) - \delta(q, v_{i+1}) &\geq (\delta(v_i, \bar{p}_1) - \delta(q, \bar{p}_1)) - (\delta(q, \bar{p}_1) + \delta(v_{i+1}, \bar{p}_1)) \\ &= (\delta(v_i, \bar{p}_1) - \delta(v_{i+1}, \bar{p}_1)) - 2\delta(q, \bar{p}_1) \\ &= 3\tau - 2\delta(q, \bar{p}_1), \qquad \text{for } i < |P| - 1.\end{aligned}$$

Let

$$\Delta_q = \max\{3\tau - 2\delta(q, \bar{p}_1), \min_{u, v \in \mathcal{V}} |\delta(q, u) - \delta(q, v)|\}.$$

We know the monotonic path gets closer to q at least Δ_q. This means that the distance that gets closer to q at each step is greater than or equal to the distance that gets closer to q as shown in [17]. Therefore, the expected length of a monotonic path by AdaptiveSearch is at most $O(n^{2/d} \ln n)$. □

ALGORITHM 4: RefineSearch ($\mathcal{G}, q, k$)

```
    Input  : LMG $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \Lambda)$, a query point $q$ and $k$ for $k$-NN
    Output : $k$-NN of $q$
1   $S \leftarrow$ AdaptiveSearch($\mathcal{G}, q$) for $S = \{s_1, s_2, \ldots, s_k\}$ where $\delta(s_1, q) \leq \ldots \leq \delta(s_k, q)$;
2   return $S$ if $k = 1$;
3   $\bar{p}_1 = s_1$;
4   $\varepsilon \leftarrow \delta(s_k, q) - \delta(s_1, q)$;
5   for each $v \in \{w \mid (\bar{p}_1, w) \in \mathcal{E}\} \setminus S$ in ascending order of $\delta(v, \bar{p}_1)$ do
6       if $\delta(v, \bar{p}_1) \geq 2\delta(q, \bar{p}_1) + \varepsilon$ then break;
7       if $\delta(q, v) < \delta(q, \bar{p}_1) + \varepsilon$ then
8           $S \leftarrow S \cup \{v\}$;
9           Remove $u$ with the largest $\delta(q, u)$ from $S$;
10          $\varepsilon \leftarrow \delta(q, s_k) - \delta(q, \bar{p}_1)$;
11  return $S$
```

There are three factors in the navigation to find 1-NN, namely, the τ used, the expected length of the monotonic path, and the expected node degree. First, the τ we use is the smallest (Lemma 4.5). Second, regarding the expected length of the monotonic path, from the viewpoint of complexity in theory, ours ($O(n^{2/d} \ln n)$) is greater than the one ($O(n^{1/d} \ln^2 n)$) by τ-MG [50]. In practice, it is observed that two nodes in a large graph can be connected by a short-length path [45]. And the lengths of monotonic paths are rather short [37]. Third, regarding the expected node degree, ours is the best as proved (Lemma 4.3). We also show the impacts of node degree on the number of traversed nodes in Figure 2. More experimental results will be shown in our experimental studies.

4.3 The Refinement Phase in k-NN Searh

We discuss the refinement in k-NN search ($k > 1$), because there is no need to refine if it only needs to find 1-NN. The algorithm for k-NN is called RefineSearch, and is shown in Algorithm 4. RefineSearch takes three inputs: an LMG graph $\mathcal{G}$, a query point q, and k for k-NN, and outputs the k-NN of q. Here, the navigation is done by AdaptiveSearchS, which is the same as AdaptiveSearch with a minor difference. The difference is that AdaptiveSearch returns the nearest neighbor, whereas AdaptiveSearchS returns a set of S in size of k, which contains k nearest neighbors of q among the nodes traversed during navigating the monotonic path toward q. In brief, $S = \{s_1, s_2, \ldots, s_k\}$ with $\delta(s_i, q) \leq \delta(s_j, q)$ if $i < j$. The refinement is in lines 3-10 in RefineSearch. We discuss the refinement phase below. Note that $\bar{p}_1 = s_1$ is 1-NN. We use $\varepsilon = \delta(q, s_k) - \delta(q, \bar{p}_1)$ as an upper bound for $\delta(q, \bar{p}_k) - \delta(q, \bar{p}_1)$, where $\bar{p}_k$ denotes the top-k nearest neighbor of the query point q (line 4), and traverse the neighbor nodes of $\bar{p}_1$ in ascending order of $\delta(v, \bar{p}_1)$ (lines 5-10). The upper bound is updated repeatedly (line 10) when S is updated.

We then prove the correctness of Algorithm 3.

Lemma 4.7. RefineSearch *(Algorithm 4) can find k-NN of q for a query point q over an LMG graph $\mathcal{G}$.*

Proof Sketch. If the point u is one of k-NN of q, we have

$$\delta(\bar{p}_1, u) \leq \delta(\bar{p}_1, \bar{p}_k) \leq \delta(\bar{p}_1, q) + \delta(\bar{p}_k, q) = 2\delta(\bar{p}_1, q) + (\delta(\bar{p}_k, q) - \delta(\bar{p}_1, q)).$$

Since ε is the upper bound of $\delta(\bar{p}_k, q) - \delta(\bar{p}_1, q)$, we have $\varepsilon \geq \delta(\bar{p}_k, q) - \delta(\bar{p}_1, q)$. Hence $\delta(\bar{p}_1, u) \leq 2\delta(\bar{p}_1, q) + \varepsilon$. If the for-loop in line 5 does not early terminate at line 6, we can find k-NN of q since traverse all neighbors of $\bar{p}_1$. Otherwise, we have $\delta(v, \bar{p}_1) \geq 2\delta(\bar{p}_1, q) + \varepsilon$ when the algorithm terminates, as we traverse nodes in ascending order of $\delta(v,\bar{p}_1)$ and $\delta(\bar{p}_1,u) \le 2\delta(\bar{p}_1,q)+\varepsilon$ holds if u is one of the k-NN, the k-NN must have been traversed. □

Based on Lemma 4.7, since the k-NN of q is found, we can directly have the following corollary, that is, the upper bound ε tightens to $\delta(q,\bar{p}_k)-\delta(q,\bar{p}_1)$ when RefineSearch (Algorithm 3) terminates, which means we will do not traverse any additional node whose distance to $\bar{p}_1$ is larger than $\delta(q,\bar{p}_1)+\delta(q,\bar{p}_k)$.

Corollary 4.1. *The upper bound ε equals to $\delta(q,\bar{p}_k)-\delta(q,\bar{p}_1)$ when* RefineSearch *(Algorithm 3) terminates.*

The following lemma gives the expected number of nodes that need to be traversed in Algorithm 4.

Lemma 4.8. *The expected number of nodes that need to be traversed in the refinement phase* RefineSearch *(Algorithm 4) is $O((\delta(q,\bar{p}_1)+\delta(q,\bar{p}_k))^d \ln n)$.*

Proof Sketch. The node u will be traversed if $\delta(u,\bar{p}_1) \le 2\delta(\bar{p}_1,q)+(\delta(\bar{p}_k,q)-\delta(\bar{p}_1,q))$. Therefore, the expected number of traversed nodes is

$$g \cdot Vol(u,\delta(q,\bar{p}_1)+\delta(q,\bar{p}_k)) = O(g\cdot(\delta(q,\bar{p}_1)+\delta(q,\bar{p}_k))^d) = O((\delta(q,\bar{p}_1)+\delta(q,\bar{p}_k))^d \ln n). \qquad \square$$

As shown in Figure 6 the difference between $\delta(q,\bar{p}_1)$ and $\delta(q,\bar{p}_k)$ is consistently small and can be seen as a constant. Therefore, the expected number of nodes traversed in the refinement of RefineSearch (Algorithm 4) can be simplified to $O(\ln n)$. Note that the expected degree of node u in G_τ is $\gamma \cdot Vol(u,3\tau)$ based on Lemma 4.3, for $\tau=\delta(q,\bar{p}_1)$ by AdaptiveSearch (Algorithm 3). The expected number of traversed nodes is $\gamma \cdot Vol(u,2\delta(q,\bar{p}_1)+(\delta(q,\bar{p}_k)-\delta(q,\bar{p}_1)))$. The difference lies in the values of $\delta(q,\bar{p}_1)$ and $\delta(q,\bar{p}_k)-\delta(q,\bar{p}_1)$. As shown in Figures 1 and 6, we observe that $\delta(q,\bar{p}_k)-\delta(q,\bar{p}_1)$ is significantly less than $\delta(q,\bar{p}_1)$. Hence, our two-phase RefineSearch (navigation and refinement) provides a similar running time compared to τ-MG [50], while giving a guaranteed result on k-NN.

5 k-ANN Search

We have discussed on k-NN search. In the literature, existing graph-based approaches (e.g., τ-MG) cannot construct an exact graph index due to the efficiency issue. Hence, they first construct a κNN graph, $\mathbf{G}$, and then construct an approximate PG, e.g., in τ-MG [50], it constructs an τ-MNG, $\tilde{G}_\tau$, from $\mathbf{G}$ with a rather small τ. The graph $\tilde{G}_\tau$ constructed may be large still. Then, it further deletes some edges from $\tilde{G}_\tau$ to ensure $\deg(u) \le \eta_d$ for any node $u \in \tilde{G}_\tau$ with a parameter η_d ($< \kappa$) tuned. We do face similar problems. In this article, we construct an approximate LMG graph, $\tilde{\mathcal{G}}=(\tilde{\mathcal{V}},\tilde{\mathcal{E}},\Lambda)$, from κNN graph, where node degree u in $\tilde{\mathcal{V}}$ is $\deg(u)\le\eta$ for $\eta=\eta_d+\Delta$ (e.g., $\Delta=10$) for $\eta<\kappa$ (details regarding the parameter settings of η,η_d, and Δ will be discussed in Section 6). In brief, let τ-MG G_0 be a G_τ graph with $\tau=0$ (Definition 2.4). An approximate LMG graph, $\tilde{\mathcal{G}}$, constructed contains an approximate $\tilde{G}_0$ of τ-MG G_0 ($\tilde{G}_0 \subseteq \tilde{\mathcal{G}}$). For an edge, (u,v), of $\tilde{\mathcal{G}}$, that appears in $\tilde{G}_0$, we label it with $\lambda(u,v)=0$. For any other edge, (u,v), we label it following Definition 4.1.

Construction of an approximate LMG: We give an algorithm, ConstructALMG, in Algorithm 5, to construct an approximate LMG graph, $\tilde{\mathcal{G}}=(\tilde{\mathcal{V}},\tilde{\mathcal{E}},\Lambda)$. ConstructALMG takes four inputs, a dataset D, a κ value used to construct κNN graph (line 1), the beam width w, and the parameter η. The graph $\tilde{\mathcal{G}}$ constructed is used in k-ANN search starting for a specific node, p_e, which is the approximate centroid of D (lines 2-3). We initialize $\tilde{\mathcal{G}}$ in line 4, and construct the first approximate LMG $\tilde{\mathcal{G}}$ with edges labeled in lines 3-10 using BeamSearch. That is, it labels an edge, (u,v) with

ALGORITHM 5: ConstructALMG (D, κ, w, η)

```
   Input  : a dataset $D$ with $n$ points, node degree of $\kappa$NN graph $\kappa$, beam width $w$ and a parameter $\eta$
   Output : an approximate LMG graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \Lambda)$
 1 Construct a $\kappa$NN graph $\mathbf{G}$ with node degree $\kappa$ on $D$ by [13];
 2 let $c$ be the centroid of $D$;
 3 $p_e \leftarrow$ BeamSearch$(\mathbf{G}, c, 1, w)$;
 4 Initialize $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \Lambda)$ with $\tilde{\mathcal{V}} = D$ and $\tilde{\mathcal{E}} = \emptyset$;
   /* Construct approximate LMG $\tilde{\mathcal{G}}$ */
 5 for each $u \in \tilde{\mathcal{V}}$ do
 6     $V' \leftarrow$ BeamSearch$(G, u, \eta, w)$ starting from $p_e$;
 7     $\tilde{\mathcal{E}} \leftarrow \tilde{\mathcal{E}} \cup \{(u, v)\}$ for every $v \in V'$ with $\lambda(u, v) = -\infty$;
 8     for each $v \in V'$ with ascending order of $\delta(u, v)$ do
 9         if $\nexists (u, u') \in \tilde{\mathcal{E}}$ with $\delta(v, u') \le \delta(u, v)$ then
10             $\tilde{\mathcal{E}} \leftarrow \tilde{\mathcal{E}} \cup \{(u, v)\}$ with $\lambda(u, v) \leftarrow 0$;
   /* Ensure the connectivity in $\tilde{\mathcal{G}}$ by adding edges */
11 while $\exists u \in \tilde{\mathcal{V}}$ that cannot be reached by $p_e$ in $\tilde{\mathcal{G}}$ do
12     $v \leftarrow$ BeamSearch$(\tilde{\mathcal{G}}, u, 1, w)$ staring from $p_e$;
13     $\tilde{\mathcal{E}} \leftarrow \tilde{\mathcal{E}} \cup \{(u, v)\}$ with $\lambda(u, v) \leftarrow 0$;
   /* Label $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \Lambda)$ by Definition 4.1 */
14 for each edge $(u, v) \in \tilde{\mathcal{E}}$ with $\lambda(u, v) = 0$ do
15     for each $u' \in \tilde{\mathcal{V}}$ with $(u, u') \in \tilde{\mathcal{E}}$ do
16         if $\lambda(u, u') \ne 0$ and $\delta(u, u') > \delta(u, v)$ then
17             $\lambda(u, u') \leftarrow \max\{\lambda(u, u'), (\delta(u, u') - \delta(u', v))/3\}$;
18 return $\tilde{\mathcal{G}}$
```

$\lambda(u, v) = 0$, if v is not pruned in line 9 and label other (u, v) with $\lambda(u, v) = -\infty$. Then, it ensures the connectivity of $\tilde{\mathcal{G}}$ by adding edges (u, v) if u cannot be reached from a starting node p_e in the first approximate LMG constructed. Here, v is 1-ANN returned by BeamSearch with u as a query point from p_e (lines 11-13). Finally, it labels every edge of $(u, v) \in aE$ if $\lambda(u, v) \ne 0$ in lines 14-17.

Lemma 5.1. *Given a κNN graph constructed for a dataset D with n points, the time complexity of Algorithm 5 is $O(n^{1.14} + \kappa n^{1+2/d} \ln n)$.*

Proof Sketch. The time complexity of constructing the κNN graph using the nn-descent algorithm (line 1) is $O(n^{1.14})$ [13]. To generate edges of an approximate LMG, we need to apply k-ANN search on the κNN graph. Based on Lemma 4.6 and the degree of nodes on κNN graph is κ, the time complexity is $O(n \cdot \kappa \cdot n^{2/d} \ln n)$. □

k-ANN Search on the Approximate LMG: We discuss AdaptiveSearch (Algorithm 3) to find 1-NN over an LMG graph, $\mathcal{G}$, by varying τ from 0 to a value with which we can find 1-NN, and we discuss RefineSearch (Algorithm 4) to find k-NN in which it first finds 1-NN by AdaptiveSearch in the navigation phase, and then refines it in the refinement phase. As shown in AdaptiveSearch, there is no input for the beam width, w, as used in BeamSearch (Algorithm 1). In other words, in AdaptiveSearch, we treat $w = 1$. However, over an approximate LMG, $\tilde{\mathcal{G}}$, constructed, there is no guarantee we can continue to navigate by varying τ with $w = 1$, if it falls in the local optimum.

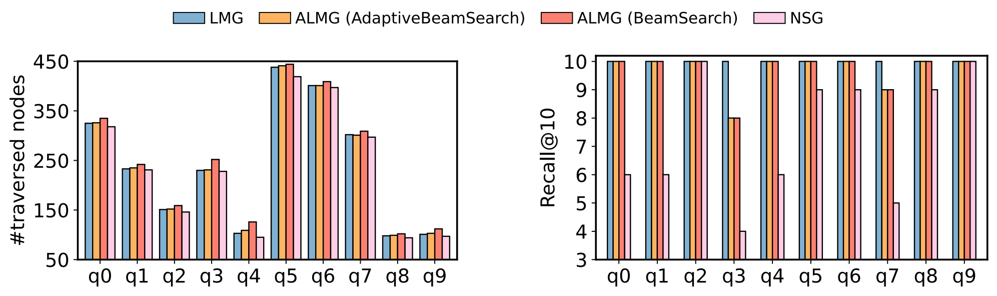
ALGORITHM 6: AdaptiveBeamSearch $(\tilde{\mathcal{G}}, q, w)$

```
Input   : A graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \Lambda)$, a query point $q$ and beam width $w$
Output  : the 1-ANN $\bar{p}_1$ of $q$
1  $\tau \leftarrow 0$; $S \leftarrow \{p_e\}$ where $p_e$ is the entry node in $\tilde{\mathcal{G}}$;
2  while $\min_{u \in S} \delta(u, q) > \tau$ do
3      while there is no unvisited nodes in $S$ do
4          $f \leftarrow$ false;
5          for each edge $(u, v) \in \{(x, y) \mid x \in S \wedge (x, y) \in \tilde{\mathcal{E}} \wedge \lambda(x, y) > \tau\}$ in ascending order of $\lambda(u, v)$ do
6              $\tau \leftarrow \lambda(u, v)$;
7              if $v \notin S$ and $(\delta(v, q) < \max_{u' \in S} \delta(u', q)$ or $|S| < w)$ then
8                  Insert $v$ into $S$;
9                  if $|S| > w$ then
10                     Delete the farthest nodes to $q$ from $S$;
11                 $f \leftarrow$ true; break;
12         if $f$ = false then return $\arg\min_{u \in S} \delta(q, u)$;
13     $u \leftarrow$ the nearest unvisited node to $q$ in $S$;
14     Mark $u$ visited;
15     $N_u \leftarrow \{v \mid (u, v) \in \tilde{\mathcal{E}} \wedge \lambda(u, v) \le \tau\}$;
16     Insert all node $v \in N_u \setminus S$ into $S$;
17     while $w < |S|$ do
18         Remove the farthest nodes to $q$ from $S$;
19 return $\arg\min_{u \in S} \delta(u, q)$
```

To address it, in the navigation phase, we need to use the beam width w while adjusting τ. We show the algorithm called AdaptiveBeamSearch in Algorithm 6. It takes three inputs, an approximate LMG, $\tilde{\mathcal{G}}$, a query point q, and the beam width w, and output 1-ANN. Initially, $\tau = 0$ and $S = \{p_e\}$ where p_e is the entry node of the graph index $\tilde{\mathcal{G}}$ in ConstructALMG (Algorithm 5). In every iteration (lines 2-18), it enlarges τ and updates S, for $|S| \le w$, if u is the local optimum (line 2). By local optimum, it means that $u \in S$ is the closest to q, but it cannot continue navigation as all nodes in S have been visited and the termination condition does not hold ($\delta(u, q) > \tau$). It enlarges τ and updates S in lines 5-11. As it may not be able to continue always in an approximate LMG, a flag f is used. Initially, $f = false$ (line 4). If f is still $false$ (line 12) after attempting to update S, it means that it cannot update S anymore, and needs to terminate with the best 1 ANN found to be returned. Note that AdaptiveBeamSearch$(\mathcal{G}, q, 1)$ is the same as AdaptiveSearch$(\mathcal{G}, q)$. We omit the discussion on the details. In the refinement phase, we use RefineSearch$(\tilde{\mathcal{G}}, q, k)$ (Algorithm 4).

LMG vs. ALMG: In theory, ALMG can provide the same guarantees for both $1/k$-ANN search as LMG does when (i) the κ value of the initially constructed κNN graph is sufficiently large, and (ii) the κNN graph is exact, i.e., each point in the κNN graph connects to its κ-NN. However, satisfying both conditions in practical scenarios poses challenges since highly efficient graph index construction is needed. Hence, in practice, ALMG loses the theoretical guarantees on both $1/k$-ANN searches. In addition, due to the construction of the initial κNN graph is approximate, ALMG uses a beam width w in 1-ANN search while adjusting τ, which implies that ALMG does not tackle the w-issue as LMG does.

As follows, we conduct an experimental study to find how the search performance of ALMG would be affected if we first construct an exact κNN graph before building the ALMG. We show



(a) The # of traversed nodes (b) Recall@10

Fig. 9. k-NN on LMG vs. k-ANN on ALMG over SIFT10K.

the performance of k-NN over LMG, $\mathcal{G}$, vs. k-ANN over approximate LMG (ALMG with Adaptive-BeamSearch), $\tilde{\mathcal{G}}$, on SIFT10K dataset in Figure 9. ALMG is constructed based on an exact κ-NN graph with $\kappa = \eta = 30$. In Figure 9, we conduct k-NN on LMG by Algorithm 4, and k-ANN on ALMG by Algorithm 6 for $k = 10$. As Figure 9(a) shown, we set $w = 5$ of Algorithm 6 to make the numbers of nodes traversed for k-NN/k-ANN over $\mathcal{G}/\tilde{\mathcal{G}}$ are similar. Figure 9(b) shows the accuracy (Recall@10). In many cases, k-ANN achieves the same accuracy of k-NN, and in some cases, the recall of k-ANN is lower than that of k-NN as expected. The results indicate that our practical approximate index construction does not lose much accuracy compared to the algorithms in theory when the accuracy of κNN graph is high.

Additionally, as shown in Figure 9, we conduct a comparison between two search algorithms on ALMG: AdaptiveBeamSearch (Algorithm 6) and BeamSearch (Algorithm 1). Both algorithms have w set to 5. It is evident that BeamSearch significantly increases the number of traversed nodes but shows similar search performance compared to AdaptiveBeamSearch. This is because, for every explored point, BeamSearch disregards the λ values on the edges and explores all edges from that point. Furthermore, we compare these approaches with NSG, i.e., G_τ with $\tau = 0$, which is also constructed based on an exact κNN graph with $\kappa = \eta_d = 30$. While the number of traversed nodes is slightly lower than that of ALMG, the accuracy (Recall@10) of NSG is notably inferior to that of ALMG.

6 Experiments

We conduct extensive experiments on real-world datasets and report our findings.

Datasets: We employ 7 real-world datasets with different numbers of dimensions/points, from diverse applications including image (SIFT1M [1], DEEP [56], GIST [1]), audio (Msong [6]) and text (GloVe [51], Crawl [2]). The summary can be found in Table 2, with the number of dimensions (dim.), the number of data points (# of points), the number of query points (# of queries), and the **local intrinsic dimensionality** (**LID**) [3], where a larger LID indicates increased difficulty in finding 1-NN in the dataset. To assess the scalability of our algorithms concerning dataset size, we also employ SIFT1B [1] in the experiment.

Algorithms: We compare our search method (Algorithms 6 and 4), denoted as `ALMG`, with four state-of-the-art PG-based approaches for k-ANN search: `NSG` [17], `HNSW` [43], `Vamana` [30] and τ-`MNG` [50]. Among them, `NSG` and `Vamana` are the up-to-date RNG-based methods, whereas `HNSW` is the up-to-date NSWG-based method. Notably, τ-`MNG` is also an RNG-based method and serves as the precursor to our approach in optimizing the graph index based on the condition of $\delta(q, \bar{p}_1) < \tau$. We obtained the source codes for `NSG`, `HNSW`, `Vamana` and τ-`MNG` online, and the implementation of τ-`MNG` is built upon the codebase of `NSG`.

Table 2. Statistics of Datasets

Dataset	dim.	# of points	# of queries	LID	Type
Msong	420	992,272	200	9.5	Audio
SIFT1M	128	1,000,000	10,000	9.3	Image
DEEP	256	1,000,000	1,000	12.1	Image
GIST	960	1,000,000	1,000	18.9	Image
GloVe	100	1,183,514	10,000	20.0	Text
Crawl	300	1,989,995	10,000	15.7	Text
SIFT1B	128	1,000,000,000	10,000	9.3	Image

Parameters: We determine the parameters for constructing the graph index of NSG, HNSW, Vamana, τ-MNG, and our approach based on prior studies [17, 43, 50, 56, 62], followed by fine-tuning via grid search within the parameter space to optimize search performance. To ensure fairness in the comparison and maintain consistency across all approaches, we exclude all pruning techniques used for distance computation (e.g., the PDP technique in [50]) and set the same degree upper bound for different approaches, i.e., set 32 for all datasets. Instead, we employ the same method based on SIMD for distance computation in all experiments. We test k-ANN queries with a default $k = 100$ for all approaches.

Performance Metrics: We focus on two metrics to measure the accuracy of search results: the recall at k (recall@k) and the **relative distance error** (**RDE**). Recall at k is the ratio between the number of successfully retrieved ground truth k-NN and k-ANN. For a query point q, the relative distance error is the average of the distance ratios of the retrieved k vectors wrt the ground truth k-NNs. This equals to $avg_{i=0}^{k-1}(\frac{\delta(q,a_i)}{\delta(q,\bar{p}_i)} - 1)$, where a_i and $\bar{p}_i$ are the approximate ith NN returned from the algorithms and exact ith NN, respectively.

We adopt the **query-per-second** (**QPS**), latency and the number of **traversed nodes** (**#TN**) to measure efficiency. QPS is the number of handled queries per second, latency is the average time taken for each query and #TN is the total number of traversed nodes during the search process.

Experimental Environments: The experiments are conducted on a Linux server with an AMD EPYC 7443 24-Core Processor and 1024G memory. All algorithms are implemented in C++. The code is compiled with g++ 8.5 under O3 optimization.

For index construction, we report the graph index time/size on one same single machine by utilizing multiple threads. For k-ANN search, we evaluate the performance of the search algorithms using a single thread following the previous works [17, 37, 50], and incrementally adjust parameters within the search algorithm (i.e., the beam width) to achieve the desired recall in the experiments.

Exp 1. *k*-NN Search Performance. We first compare our LMG with the existing theoretical guaranteed approach, τ-MG [50], for arbitrary 1-NN queries. Given the substantial construction time of both methods, we implement the experiments on the SIFT10K, which randomly extracts $10,000$ points from the SIFT1M dataset. Figure 2 presents the number of traversed nodes for finding 1-NN. In the comparison, we implement τ-MG with two different values of τ, $\tau = \max_{q\in Q} \delta(q,\bar{p}_1)$ for all queries and $\tau = \delta(q,\bar{p}_1)$ for each individual query q. All methods can find the exact 1-NN, with LMG consistently traversing the fewest nodes to find 1-NN.

We further compare the two methods for finding 10-NN. For τ-MG, we construct 10 indexes with setting $\tau = \delta(q,\bar{p}_1)$ for each query q. In the search process on τ-MG, we set the beam width to 1 and 10 respectively. As illustrated in Figure 10, LMG consistently traverses the fewest number of

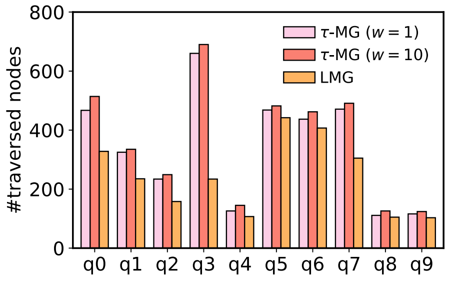
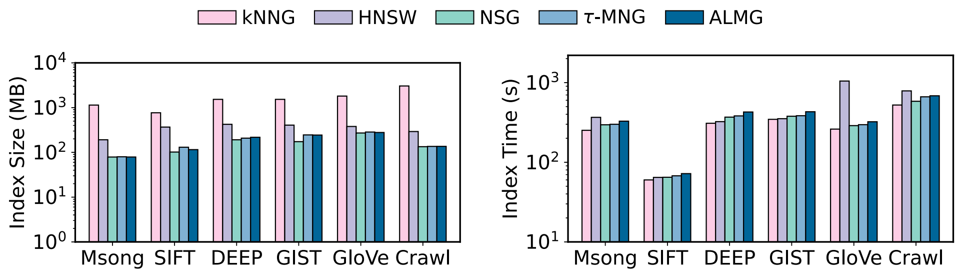


Fig. 10. The # of traversed nodes in SIFT10K for 10-NN.



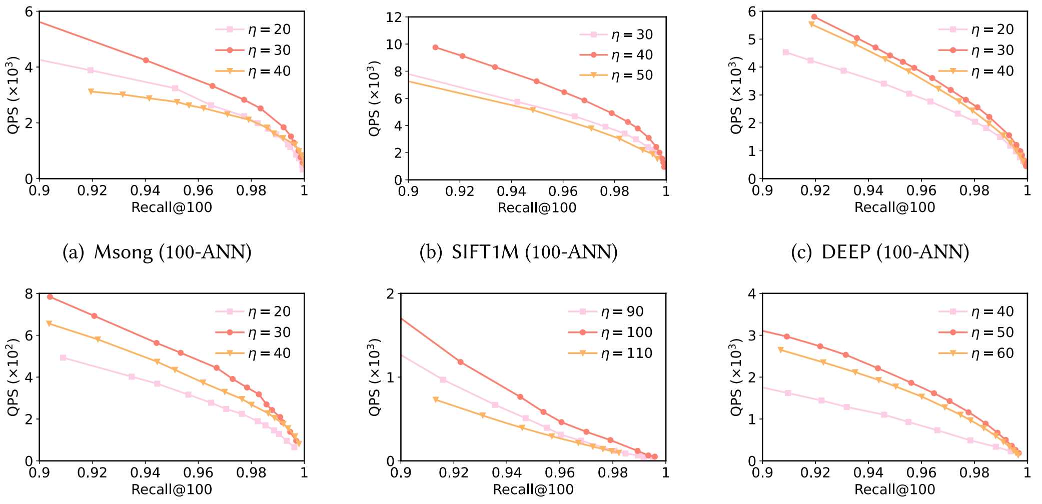
(a) Size of graph topology

(b) Index construction time

Fig. 11. Index construction performance.

nodes. The τ-MG with beam widths of 1 and 10 can achieve recall@10 at 0.91 and 0.98 respectively, while our LMG can reach recall@10 at 1.

Exp 2. Graph Index Construction. As the construction of ALMG involves building an approximate 0-MG (i.e., NSG) initially, we adopt the same values for parameters κ and w as used in [17] ($\kappa = 100$, $w = 150$). We then focus on studying the effect of parameter η in the construction of an ALMG using ConstructALMG (Algorithm 5). We present our findings based on the results obtained from AdaptiveBeamSearch (Algorithm 6). We present the results of all datasets in Figure 12 under different values of η. The parameters of the pink line are set in the same way as the parameters of NSG by grid search within the parameter space, i.e., the value of η corresponds to their node degree limit. It can be observed that the performance of ALMG initially improves and then deteriorates while η increases across the datasets. The reason behind this lies in the dependency of search efficiency on the number of points traversed during the search process. As η increases initially, ALMG facilitates easier exploration of the neighbor points in proximity to the query point by introducing additional edges (i.e., support search on G_τ with a larger value of τ). This leads to an enhancement in performance. However, when the value of η becomes excessively large, the number of traversed neighbors during the search increases, but the quality does not improve significantly due to the limitations of the approximate κNN graph used. As a result, the search time increases significantly while the improvement in recall or precision remains the same. Furthermore, it can be noted that the disparity between different values of η diminishes when the recall or precision reaches relatively high values (i.e., recall close to 1). This is attributed to the fact that when recall or precision is relatively small it is easier to explore a greater number of nearby neighbors of a query point through the additional edges in the κNN graph, which are not included in PGs.



(a) Msong (100-ANN) (b) SIFT1M (100-ANN) (c) DEEP (100-ANN)

(d) GIST (100-ANN) (e) GloVe (100-ANN) (f) Crawl (100-ANN)

Fig. 12. Study of parameter η for 100-ANN.

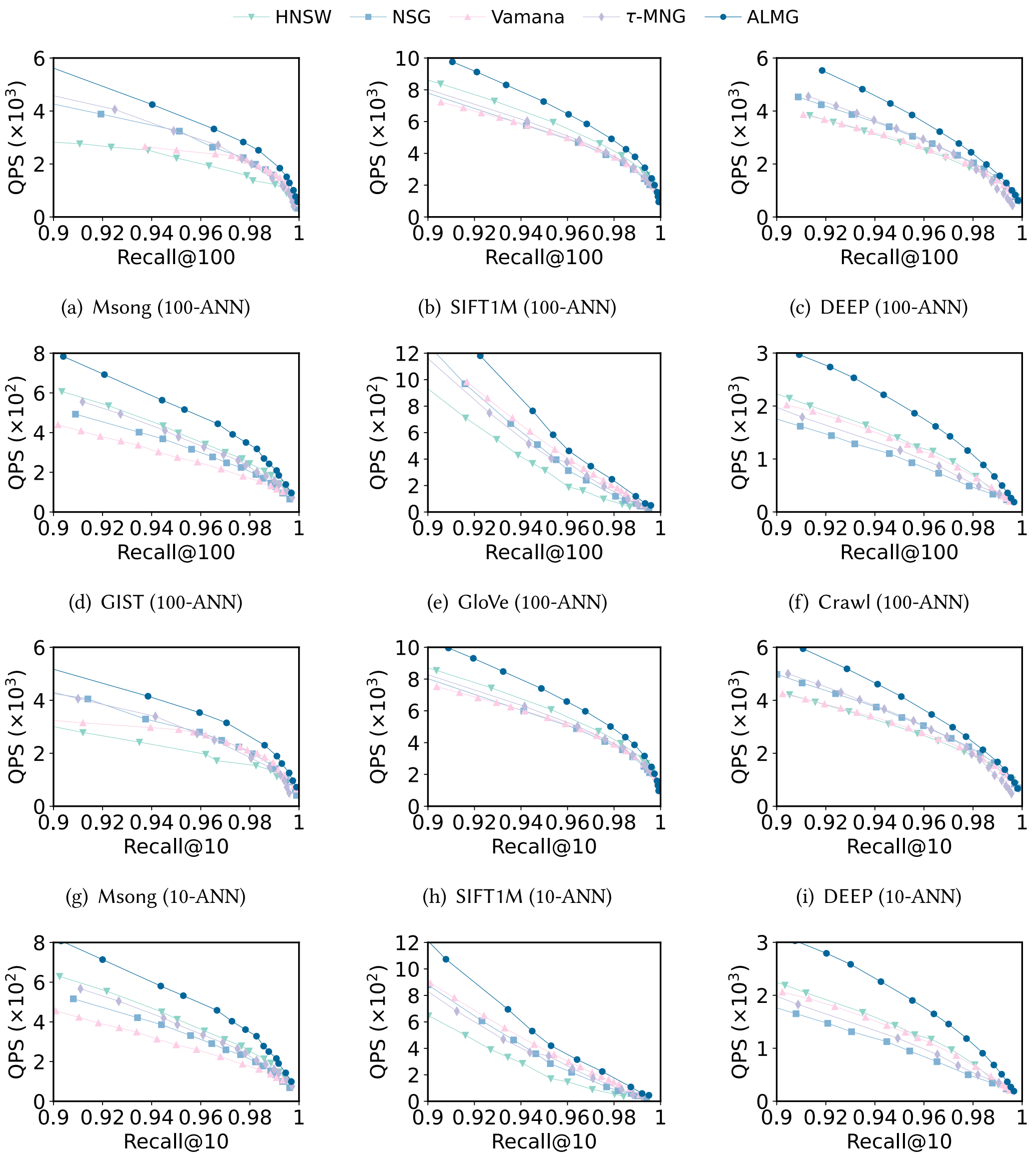
In our later experiments, we set the value of η as the node degree limit of NSG plus 10 by default, while the node degree limit of NSG is set by grid search within the parameter space. The values of κ and w are set the same as the value of NSG in [17, 56].

We then examine the index size and index construction time. All NSG, τ-MNG, and ALMG are constructed based on κNN index. As depicted in Figure 11(a), the index size of ALMG is almost similar to that of NSG and τ-MNG by vary the parameters for fairness in later comparisons, and significantly smaller than κNN and HNSW by more than one order of magnitude. Both NSG and τ-MNG construction algorithms necessitate the construction of κNN prior to their own construction. However, similar to NSG and τ-MNG, since our index size is much smaller than that of κNN, it supports that our algorithms do not require additional memory during the index construction phase, compared to the graph indexes of NSG and τ-MNG.

Figure 11(b) presents the index construction time of five different graph indexes. The results show that ALMG construction time is similar to that of NSG and τ-MNG, with a slightly higher value. This slight increase can be attributed to the additional distance computations required for the extra edges included in our graph index.

Exp 3. $1/k$-ANN Search Performance. We plot the QPS-recall curves in Figure 13 (upper-right indicates better performance), the latency-recall curves in Figure 14 and the #TN-RDE curves in Figure 16 (lower-left indicates better performance). Both figures compare our ALMG (Algorithm 6) with state-of-the-art methods across six datasets. We focus only on the region where the recall is at least 90% and the RDE is less than 0.004 (0.01 for GloVe), based on practical scenarios. According to the results illustrated in Figures 13, 14, and 16, it is evident that our approach, ALMG, outperforms all baseline methods across different recall values. Notably, Figure 13 demonstrates the enhancements in performance at lower recall thresholds, e.g., from 0.9 to 0.98, while for the high recall values, e.g., surpassing 0.99, Figure 14 clearly shows that ALMG consistently outperforms other existing methods on most datasets.

In particular, on Msong, SIFT1M, DEEP, GIST, GloVe, and Crawl, when recall@100 is 0.90, ALMG is at least 1.31, 1.13, 1.21, 1.29, 1.35, 1.39 times faster, and up to 1.99, 1.25, 1.44, 1.58, 1.83, 1.77 times faster than the three state-of-the-art methods, respectively. It represents a substantial improvement



(a) Msong (100-ANN) (b) SIFT1M (100-ANN) (c) DEEP (100-ANN)

(d) GIST (100-ANN) (e) GloVe (100-ANN) (f) Crawl (100-ANN)

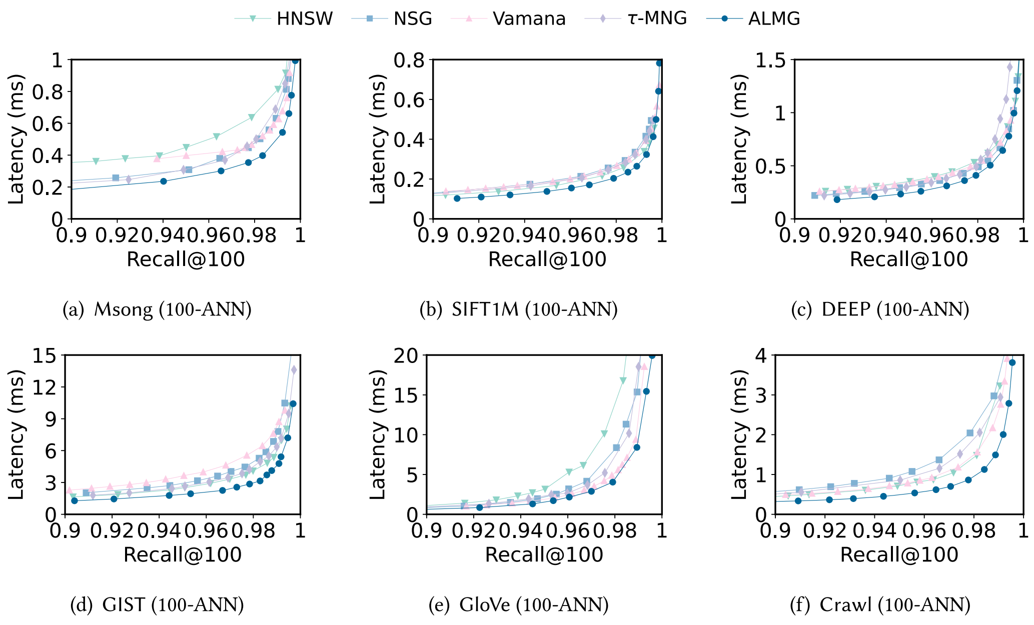
(g) Msong (10-ANN) (h) SIFT1M (10-ANN) (i) DEEP (10-ANN)

(j) GIST (10-ANN) (k) GloVe (10-ANN) (l) Crawl (10-ANN)

Fig. 13. Comparisons with current SOTA k-ANN methods (QPS vs. Recall).

in the performance of k-ANN search compared to previous works. Moreover, our method exhibits a larger improvement in 1-ANN as shown in Figure 15, confirming the effectiveness of our adaptive method. Furthermore, our methods consistently yield more accurate results, as evidenced by the significantly smaller relative distance error between our results and the ground truth.

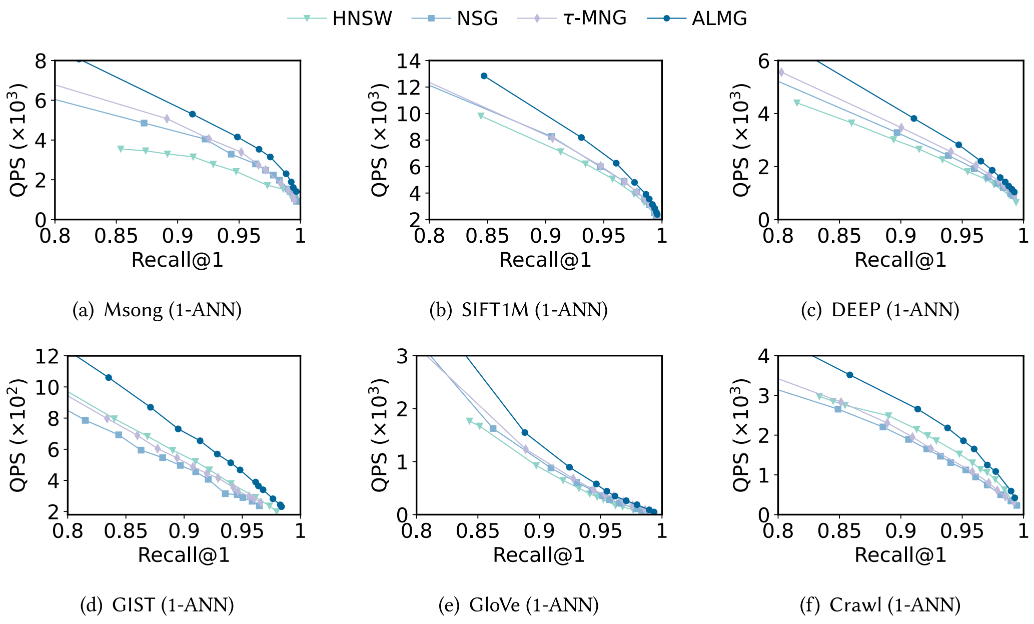
Furthermore, the performance gap between the search on ALMG and the other three graph indexes widens when the recall is low or the RDE is high. This is because there is a greater potential for additional edges that are not present in edges with label 0 (i.e., focusing solely on the PG index) to explore points closer to the query point. Additionally, it is worth noting that NSG does not



(a) Msong (100-ANN) (b) SIFT1M (100-ANN) (c) DEEP (100-ANN)

(d) GIST (100-ANN) (e) GloVe (100-ANN) (f) Crawl (100-ANN)

Fig. 14. Comparisons with current SOTA k-ANN methods (Latency vs. Recall).



(a) Msong (1-ANN) (b) SIFT1M (1-ANN) (c) DEEP (1-ANN)

(d) GIST (1-ANN) (e) GloVe (1-ANN) (f) Crawl (1-ANN)

Fig. 15. Comparisons with current SOTA methods on 1-ANN (QPS vs. Recall).

consistently outperform HNSW, and τ-MNG consistently performs similarly to NSG, with potential advantages on certain datasets within specific recall ranges, but τ-MNG has more advantage on the performance of the 1-ANN. In contrast, our ALMG consistently outperforms all three methods across all six datasets.

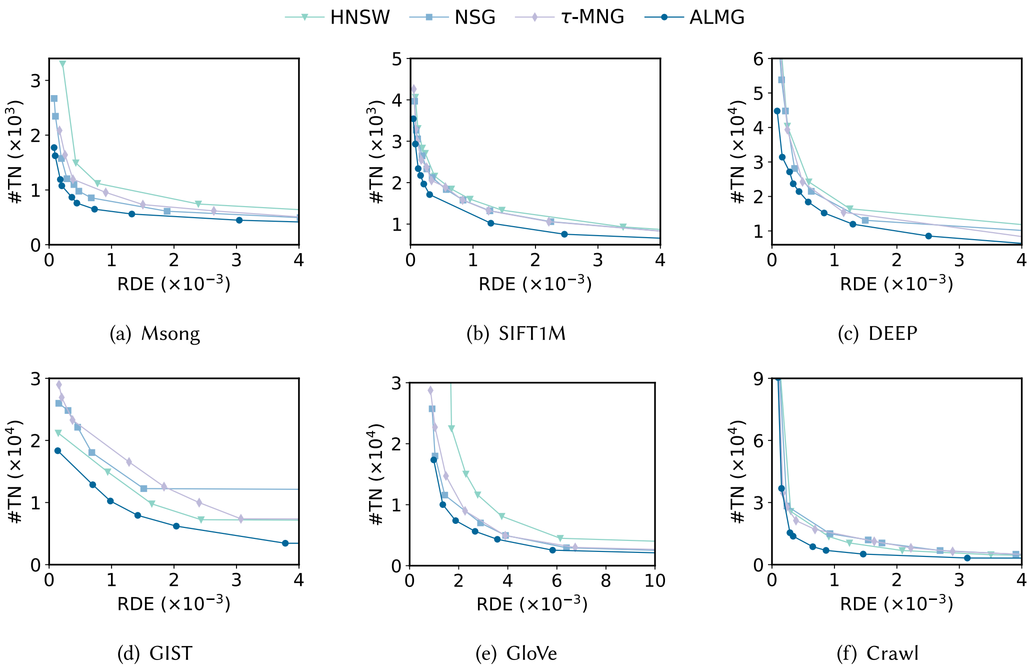


Fig. 16. Comparisons with current SOTA k-ANN methods (#TN vs. RDE).

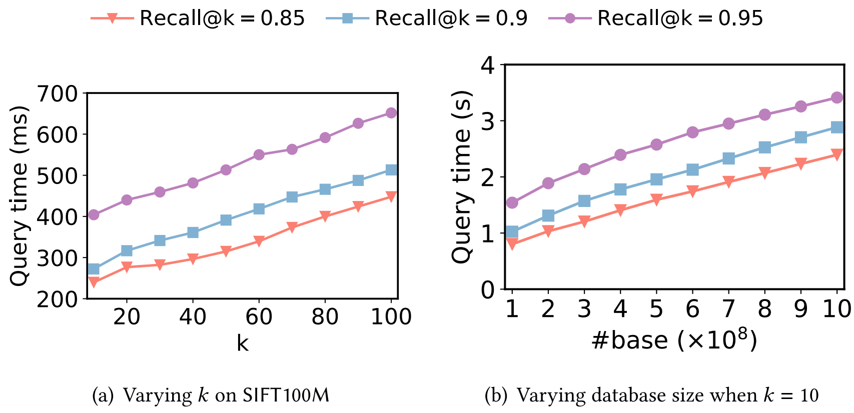


Fig. 17. Scalability performance.

Exp 4. Scalability. To evaluate scalability, we vary the value of k on the SIFT100M dataset and perform k-ANN search. Figure 17(a) illustrates the average query processing time for different values of k when the recalls are 0.85, 0.9, and 0.95, respectively. The average query time exhibits a linear increase while k increases. Moreover, the difference between the plots for recall at 0.85 and 0.9 is smaller than the difference between the plots for recall at 0.9 and 0.95. This finding indicates that the growth in running time surpasses a linear relationship with the growth in recall.

We also evaluate the performance of ALMG against the dataset size using the SIFT1B dataset. We vary the datasize from 100 million to 1 billion by randomly selecting points from the SIFT1B dataset. Figure 17(b) illustrates the relationship between the average query processing time and the dataset size. The average query time increases almost linearly as the size of dataset (#base) increases. Furthermore, we can find that the average query processing time rises at a much slower rate compared to the growth in the database size.

7 Related Work

We review the related works on k-NN/k-ANN search problems with/without theoretical guarantees.

Theoretical Guaranteed Methods: We discuss the graph-based approaches and non-graph-based methods for classifying theoretical guaranteed methods.

Graph-based methods: The DG is the dual graph of the Voronoi diagram [5]. DG guarantees finding the nearest neighbor through greedy routing of any given query point q [34]. However, when the dimensionality is large, DG becomes a complete graph [22], resulting in an intolerable time cost for greedy routing. To ensure the efficiency of the search process, it is necessary to limit the node degree of the graph index. Fu et al. [17] propose the monotonic relative neighborhood graph (MRNG), which provides a limited node degree guarantee. MRNG ensures finding the nearest neighbor of a query point q if q is in the dataset ($q \in D$). However, it does not provide any guarantee when $q \notin D$. Harwood et al. [22] propose FANNG, which ensures finding the nearest neighbor $\bar{p}$ of any given query point q if $\delta(q, \bar{p}) < \tau$. However, it does not provide a theoretical analysis of the node degree and time complexity of the greedy routing. To address this issue, Peng et al. [50] propose τ-MG, which guarantees finding the nearest neighbor $\bar{p}$ of any given query point q if $\delta(q, \bar{p}) < \tau$, while also limiting the node degree. However, as we analyze in the introduction and the previous section, determining the value of τ is challenging since it requires prior knowledge. Yet, the index and search performance of τ-MG are highly dependent on the value of τ. For k-ANN, to the best of our knowledge, there are no existing graph-based methods can efficiently provide a theoretical guarantee. The only finding is in [26], which shows that the time required of existing approaches (e.g., HNSW, NSG) to achieve reasonably accurate results in a k-ANN query is linear in the dataset size by experimental.

Non-graph-based methods: Theoretical guaranteed non-graph-based methods for ANN search problems include tree-based methods (e.g., [10, 11, 28, 63]) and hashing-based methods (e.g., [18, 24, 25, 53, 64]). However, recent studies [4, 36, 37, 47, 56] demonstrate that these methods are outperformed by graph-based methods, since non-graph-based methods tend to check much more points than the graph-based methods to achieve the same accuracy.

Non-guaranteed Methods: As mentioned above, graph-based ANN algorithms have become the dominant paradigm in this field. Hence, in this subsection, we will not cover non-graph-based methods and instead recommend interested readers to refer to recent surveys (e.g., [27, 36, 54, 57]) for more comprehensive details.

To reduce the node degree of DG in high-dimensional spaces, κNN graph is proposed as an approximation of DG, i.e., each node in κNN graph is only connected to its top-κ nearest points [21, 32]. However, constructing the exact κNN graph requires $O(n^2)$ time, thus several studies focus on constructing the approximate κNN graphs to expedite the construction process [13, 15, 37].

The Milgram's social experiment revealed that two nodes in a large graph can be connected by a short-length path, and can be discovered through a greedy routing [45]. Based on this, numerous graph-based methods are proposed, however, many of them are designed for low-dimensional spaces [33, 58], while others suffer from high node degrees in high-dimensional spaces [42, 44]. Recently, Malkov et al. [43] introduce the hierarchical navigable small world graph, which supports efficient routing in polylogarithmic time, but it does not provide any error guarantees on search results.

Several studies apply the combination of graph-based methods with other techniques, e.g., [8, 14, 48] use clustering or partitioning to divide the points in the dataset into multiple subgroups, then graph-based methods are applied within each subgroup. In addition, numerous works have focused on expanding the scope of ANN search to various scenarios, e.g., search on billion-scale datasets [9, 30, 49], hybrid search [7, 20, 60, 61, 65], out-of-distribution queries [29], privacy-preserving search [41]. Furthermore, a recent study [62] has identified certain issues with the NSG and HNSW constructions. To address these issues, it first introduces a new framework for NSG construction that aims at accelerating the index construction without compromising search performance. In the case of HNSW, this work advocates for integrating the NSG construction into each layer of HNSW to optimize both search performance and index building efficiency.

8 Conclusion

In this article, we propose a new graph-based approach to support k-NN/ANN search. Our approach has a theoretical guarantee to support both 1-NN and k-NN ($k > 1$) for any query point q without constraints. Based on our approach for $1/k$-NN in theory, we give new algorithms to deal with $1/k$-ANN in practice. Our extensive experiments demonstrate that our proposed approach surpasses all the state-of-the-art baselines on $1/k$-ANN search. Our approach can achieve an average improvement of 1.51x compared to the baselines when recall@10=0.85, and such accuracy/-efficiency is achieved with a similar index size and index construction time in comparison to the state-of-the-art approaches.

References

[1] 2010. Datasets for approximate nearest neighbor search. Retrieved May 30, 2024 from http://corpus-texmex.irisa.fr/

[2] 2023. Common Crawl. Retrieved May 30, 2024 from https://commoncrawl.org/

[3] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E. Houle, Ken-ichi Kawarabayashi, and Michael Nett. 2015. Estimating local intrinsic dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 29–38.

[4] Martin Aumüller, Erik Bernhardsson, and Alexander John Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Inf. Syst.* 87, C (2020), 101374.

[5] Franz Aurenhammer. 1991. Voronoi diagrams - A survey of a fundamental geometric data structure. *ACM Computing Surveys* 23, 3 (1991), 345–405.

[6] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. 2011. The million song dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*. University of Miami, 591–596.

[7] Yuzheng Cai, Jiayang Shi, Yizhuo Chen, and Weiguo Zheng. 2024. Navigating labels and vectors: A unified approach to filtered approximate nearest neighbor search. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 246:1–246:27.

[8] 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*. Retrieved from https://github.com/Microsoft/SPTAG

[9] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. SPANN: Highly-efficient billion-scale approximate nearest neighbor search. In *Proceedings of the 35th Conference on Neural Information Processing Systems*.

[10] Bin Cui, Beng Chin Ooi, Jianwen Su, and Kian-Lee Tan. 2003. Contorting high dimensional data for efficient main memory processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. ACM, 479–490.

[11] Bin Cui, Beng Chin Ooi, Jianwen Su, and Kian-Lee Tan. 2005. Indexing high-dimensional data for efficient in-memory similarity search. *IEEE Transactions on Knowledge and Data Engineering* 17, 3 (2005), 339–353.

[12] D. W. Dearholt, N. Gonzales, and G. Kurup. 1988. Monotonic search networks for computer vision databases. In *Proceedings of the 22nd Asilomar Conference on Signals, Systems and Computers*. IEEE, 548–553.

[13] Wei Dong, Moses Charikar, 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*. ACM, 577–586.

[14] Carlos Eiras-Franco, David Martínez-Rego, Leslie Kanthan, César Piñeiro, Antonio Bahamonde, Bertha Guijarro-Berdiñas, and Amparo Alonso-Betanzos. 2020. Fast distributed kNN graph construction using auto-tuned locality-sensitive hashing. *ACM Transactions on Intelligent Systems and Technology* 11, 6 (2020), 71:1–71:18.

[15] Cong Fu and Deng Cai. 2016. EFANNA: An extremely fast approximate nearest neighbor search algorithm based on kNN graph. arXiv:1609.07228. Retrieved from https://arxiv.org/abs/1609.07228

[16] Cong Fu, Changxu Wang, and Deng Cai. 2022. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2022), 4139–4150.

[17] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment* 12, 5 (2019), 461–474.

[18] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 541–552.

[19] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 27 pages.

[20] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, et al.. 2023. Filtered-DiskANN: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference 2023*. ACM, 3406–3416.

[21] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. 2011. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. IJCAI/AAAI, 1312–1317.

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

[23] Jae-Pil Heo, Zhe L. Lin, and Sung-Eui Yoon. 2019. Distance encoded product quantization for approximate K-Nearest neighbor search in high-dimensional space. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41, 9 (2019), 2084–2097.

[24] 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.

[25] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 13th Annual ACM Symposium on the Theory of Computing*. ACM, 604–613.

[26] Piotr Indyk and Haike Xu. 2023. Worst-case performance of popular approximate nearest neighbor search implementations: Guarantees and limitations. In *Proceedings of the NeurIPS 2023,*.

[27] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. 2021. A survey on locality sensitive hashing algorithms and their applications. arXiv:2102.08942. Retrieved from https://arxiv.org/abs/2102.08942

[28] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. 2005. iDistance: An adaptive B$^+$-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems* 30, 2 (2005), 364–397.

[29] Shikhar Jaiswal, Ravishankar Krishnaswamy, Ankit Garg, Harsha Vardhan Simhadri, and Sheshansh Agrawal. 2022. OOD-DiskANN: Efficient and scalable graph ANNS for out-of-distribution queries. arXiv:2211.12850. Retrieved from https://arxiv.org/abs/2211.12850

[30] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast accurate billion-point nearest neighbor search on a single node. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, Inc.

[31] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128.

[32] 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.

[33] Jon M. Kleinberg. 2000. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*. ACM, 163–170.

[34] Govinda D. Kurup. 1992. *Database Organized on the Basis of Similarities with Applications in Computer Vision*. Ph.D. Dissertation.

[35] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*.

[36] 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 International Conference on Management of Data, SIGMOD Conference 2020*. ACM, 2539–2554.

[37] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2020), 1475–1488.

[38] Yiran Li, Renchi Yang, and Jieming Shi. 2023. Efficient and effective attributed hypergraph clustering via K-Nearest neighbor augmentation. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 116:1–116:23.

[39] Yingfan Liu, Hong Cheng, and Jiangtao Cui. 2017. PQBF: I/O-Efficient approximate nearest neighbor search by product quantization. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 667–676.

[40] Ying Liu, Dengsheng Zhang, Guojun Lu, and Wei-Ying Ma. 2007. A survey of content-based image retrieval with high-level semantics. *Pattern Recognition* 40, 1 (2007), 262–282.

[41] Yingfan Liu, Yandi Zhang, Jiadong Xie, Hui Li, Jeffrey Xu Yu, and Jiangtao Cui. 2025. Privacy-preserving approximate nearest neighbor search on high-dimensional data. In *Proceedings of the ICDE*. IEEE Computer Society, 3017–3029.

[42] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836.

[43] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836.

[44] Charles U. Martel and Van Nguyen. 2004. Analyzing Kleinberg's (and other) small-world models. In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing*. ACM, 179–188.

[45] Stanley Milgram. 1967. The small world problem. *Psychology Today* 2, 1 (1967), 60–67.

[46] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Ali Mousavi, Ihab F. Ilyas, Umar Farooq Minhas, Jeffrey Pound, and Theodoros Rekatsinas. 2023. High-throughput vector similarity search in knowledge graphs. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 197:1–197:25.

[47] Stanislav Morozov and Artem Babenko. 2018. Non-metric similarity graphs for maximum inner product search. In *Proceedings of the Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018*. 4726–4735.

[48] Javier Alvaro Vargas Muñoz, Marcos André Gonçalves, Zanoni Dias, and Ricardo da Silva Torres. 2019. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognit.* 96, C (2019), 106970.

[49] Jiongkang Ni, Xiaoliang Xu, Yuxiang Wang, Can Li, Jiajie Yao, Shihai Xiao, and Xuecang Zhang. 2023. DiskANN++: Efficient page-based search over isomorphic mapped graph index using query-sensitivity entry vertex. arXiv:2310.00402. Retrieved from https://arxiv.org/abs/2310.00402

[50] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient approximate nearest neighbor search in multi-dimensional databases. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 54:1–54:27.

[51] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 1532–1543.

[52] Liudmila Prokhorenkova and Aleksandr Shekhovtsov. 2020. Graph-based nearest neighbor search: From practice to theory. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020 (Proceedings of Machine Learning Research)*. PMLR, 7803–7813.

[53] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: Solving c-Approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *Proceedings of the VLDB Endowment* 8, 1 (2014), 1–12.

[54] Yao Tian, Ziyang Yue, Ruiyuan Zhang, Xi Zhao, Bolong Zheng, and Xiaofang Zhou. 2023. Approximate nearest neighbor search in high dimensional vector databases: Current research and future directions. *IEEE Data Eng. Bull.* 46, 3 (2023), 39–54.

[55] Hongya Wang, Zhizheng Wang, Wei Wang, Yingyuan Xiao, Zeng Zhao, and Kaixiang Yang. 2020. A note on graph-based nearest neighbor search. arXiv:2012.11083. Retrieved from https://arxiv.org/abs/2012.11083

[56] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 14, 11 (2021), 1964–1978.

[57] Zeyu Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Graph-and tree-based indexes for high-dimensional vector similarity search: Analyses, comparisons, and future directions. *IEEE Data Eng. Bull.* 46, 3 (2023), 3–21.

[58] Duncan J. Watts and Steven H. Strogatz. 1998. Collective dynamics of 'small-world'networks. *Nature* 393, 6684 (1998), 440–442.

[59] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N. Holtmann-Rice, David Simcha, and Felix X. Yu. 2017. Multiscale quantization for fast similarity search. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*. 5745–5755.

[60] Jiadong Xie, Jeffrey Xu Yu, and Yingfan Liu. 2025. Fast approximate similarity join in vector databases. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 158:1–158:26.

[61] Yuexuan Xu, Jianyang Gao, Yutong Gou, Cheng Long, and Christian S. Jensen. 2024. iRangeGraph: Improvising range-dedicated graphs for range-filtering nearest neighbor search. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 239:1–239:26.

[62] Shuo Yang, Jiadong Xie, Yingfan Liu, Jeffrey Xu Yu, Xiyue Gao, Qianru Wang, Yanguo Peng, and Jiangtao Cui. 2025. Revisiting the index construction of proximity graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 18, 6 (2025), 1825–1838.

[63] Cui Yu, Beng Chin Ooi, Kian-Lee Tan, and H. V. Jagadish. 2001. Indexing the distance: An efficient method to KNN processing. In *Proceedings of the VLDB*. Morgan Kaufmann, 421–430.

[64] Jiaqi Zhai, Yin Lou, and Johannes Gehrke. 2011. ATLAS: A probabilistic algorithm for high dimensional similarity search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 997–1008.

[65] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, Mao Yang, and Lidong Zhou. 2023. VBASE: Unifying online vector similarity search and relational queries via relaxed monotonicity. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, 377–395.

[66] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. *Proceedings of the VLDB Endowment* 16, 8 (2023), 1979–1991.

Received 15 June 2024; revised 25 February 2025; accepted 1 May 2025