Fast-Convergent Proximity Graphs for Approximate Nearest Neighbor Search

BINHONG LI, DSA Thrust, The Hong Kong University of Science and Technology (Guangzhou), China
XIAO YAN, Institute for Math & AI, Wuhan University, China
SHANGQI LU*, DSA Thrust, The Hong Kong University of Science and Technology (Guangzhou), China

Approximate nearest neighbor (ANN) search in high-dimensional metric spaces is a fundamental problem with many applications. Over the past decade, proximity graph (PG)-based indexes have demonstrated superior empirical performance over alternatives. However, these methods often lack theoretical guarantees regarding the quality of query results, especially in the worst-case scenarios. In this paper, we introduce the *α-convergent graph* (α-CG), a new PG structure that employs a new carefully designed *edge pruning rule*. If the distance between the query point q and its exact nearest neighbor v^* is at most τ for some constant $\tau > 0$, our α-CG finds the exact nearest neighbor in poly-logarithmic time, assuming bounded intrinsic dimensionality for the dataset; otherwise, it can find an ANN in the same time. To enhance scalability, we develop the *α-convergent neighborhood graph* (α-CNG), a practical variant that applies the pruning rule locally within each point's neighbors. We also introduce optimizations to reduce the index construction time. Experimental results show that our α-CNG outperforms existing PGs on real-world datasets. For most datasets, α-CNG can reduce the number of distance computations and search steps by over 15% and 45%, respectively, when compared with the best-performing baseline.

CCS Concepts: • **Theory of computation → Data structures design and analysis**; • **Information systems → Information retrieval query processing**.

Additional Key Words and Phrases: Proximity Graphs, Approximate Nearest Neighbor Search, Data Structures

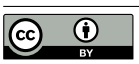
ACM Reference Format:
Binhong Li, Xiao Yan, and Shangqi Lu. 2026. Fast-Convergent Proximity Graphs for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 36 (February 2026), 24 pages. https://doi.org/10.1145/3786650

1 Introduction

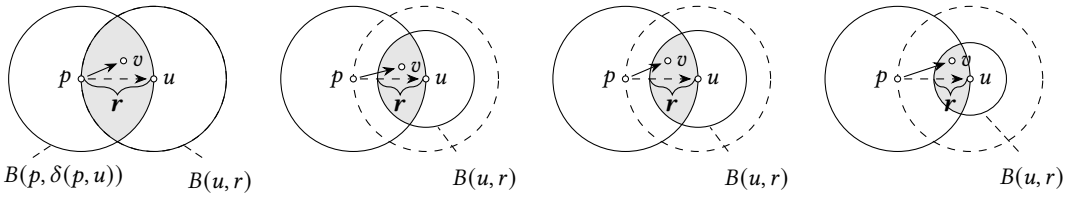
Approximate nearest neighbor search is a fundamental component in various applications such as image retrieval [42, 57], recommendation systems [11, 16], and data mining [32, 36]. Let P be a set of n data points from a metric space $(\mathcal{X}, \delta)$ where $\mathcal{X}$ is a set of points and $\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ is a distance function. We want to construct a data structure on P that supports the following *nearest neighbor* (NN) query: given a query point $q \in \mathcal{X}$, return the point $v^* \in P$ closest to q. Due to the "curse of dimensionality", all known space-efficient solutions for NN search have search times that grow exponentially with the dimensionality of the metric space. To enhance query efficiency, researchers often resort to c-approximate nearest neighbor (ANN) queries for some constant $c > 1$,

*Dr. Shangqi Lu is the corresponding author.

Authors' Contact Information: Binhong Li, DSA Thrust, The Hong Kong University of Science and Technology (Guangzhou), China, bli120@connect.hkust-gz.edu.cn; Xiao Yan, Institute for Math & AI, Wuhan University, China, yanxiaosunny@whu.edu.cn; Shangqi Lu, DSA Thrust, The Hong Kong University of Science and Technology (Guangzhou), China, shangqilu@hkust-gz.edu.cn.



ACM 2836-6573/2026/2-ART36
https://doi.org/10.1145/3786650



(a) MRNG **(b)** Vamana **(c)** τ-MG **(d)** α-CG

Fig. 1. The edge pruning rules of existing MRNG, τ-MG, Vamana, and α-CG. Given a data point p, a candidate u is pruned if p already has an out-neighbor v falling within the intersection of $B(p, \delta(p, u))$ and $B(u, r)$.

Table 1. Comparison of PG-based methods

		Accuracy Guarantees			Theoretical Properties		
Method	**Pruning radius**	$\delta(q, v^*) = 0$	$\delta(q, v^*) < \tau$	$\delta(q, v^*) > \tau$	Routing Length	Avg. Time	Worst-case Time
MRNG [22]	$d(p, u)$	✓	✗	✗	Linear	$O(n^{\frac{2}{m}} \ln n)$	$O(n)$
τ-MG [56]	$d(p, u) - 3\tau$	✓	✓	✗	Linear	$O(n^{\frac{1}{m}} (\ln n)^2)$	$O(n)$
Vamana [62]	$\frac{1}{\alpha} d(p, u)$	✓	☑	☑	$O(\log_\alpha \frac{\Delta}{(\alpha-1)\epsilon})$	–	$O\Big(\alpha^d \log \Delta \log \frac{\Delta}{(\alpha-1)\epsilon}\Big)$
α-CG (Ours)	$\frac{1}{\alpha} d(p, u) - (\alpha + 1)\tau$	✓	✓	☑	$O(\log_\alpha \Delta)^\dagger$	–	$O\big((\alpha\tau)^d \log \Delta \log_\alpha \Delta\big)^\dagger$

✓: exact guarantee; ☑: $\big(\frac{\alpha+1}{\alpha-1} + \epsilon\big)$-ANN; ✗: no guarantee; m: dimension of Euclidean space; d: doubling dimension; Δ: aspect ratio; † when $\delta(q, v^*) \le \tau$.

which balance query accuracy and search time by returning a point whose distance to q is within a c-factor of the exact NN.

In recent years, numerous studies [5, 22, 50, 56, 62, 68] have demonstrated that proximity graph-based solutions exhibit superior empirical performance on real-world data over other indexes (e.g., trees [44, 65] and inverted index [6, 39]). A *proximity graph* (PG) is a simple directed graph, where each vertex represents a data point $p \in P$ and edges are connected based on particular geometric properties (usually close neighbors in distance). Given a query point q, a simple *greedy routing* is performed to find an ANN of q: starting from a fixed or random entry vertex, at each step, it explores the nearest out-neighbor of the currently visited vertex to q. This process traverses a search path until no closer nodes can be identified, returning the closest visited point as the answer[1]. The running time of the algorithm is bounded by the total out-degree of the nodes on the search path. Therefore, to ensure efficient query performance, the goal is to construct a sparse graph (with a low maximum out-degree) that allows the search algorithm to converge to the ANNs of q quickly.

1.1 Existing PGs and Their Limitations

Most methods for constructing PGs follow a common framework. For each data point p, we first obtain a candidate set $\mathcal{V}$ of points from P that are close to p, through greedy search on a base graph (e.g., an approximate K-NN graph [22, 56]) or an intermediate graph over a subset of P [50, 62]. A small subset of $\mathcal{V}$ is then selected as the out-neighbors of p to maintain graph connectivity. Crucially, the resulting graph should have the "shortcutable" property: for any query point q, if data point p is not an ANN of q, p should have an out-neighbor that is closer to q than p. To achieve this,

[1]We can generalize the algorithm to find k ANNs by maintaining a sorted list, known as the *beam search* algorithm (see Section 2.2).

existing PG solutions introduce *edge pruning rules* to eliminate unnecessary candidates in $\mathcal{V}$ in the following procedure. For each point $u \in \mathcal{V}$ processed in ascending order of the distances to p:

- u is pruned if there already exists an out-neighbor v of p falling within the intersection of the two balls[2] $B(p, \delta(p, u))$ and $B(u, r)$, where $r > 0$ is the *pruning radius* depending on $\delta(p, u)$;
- otherwise, u is added as a new out-neighbor of p.

Different choices of the pruning radius r lead to distinct properties and search guarantees of PGs. In early PG methods, such as HNSW [50] and NSG [22], r is set directly to $\delta(p, u)$, as illustrated in Fig. 1a. When $\mathcal{V} = P \setminus \{p\}$ for every $p \in P$ (the candidate set is the entire dataset), the constructed PG is the monotonic relative neighborhood graph (MRNG) and has the following property [2, 22]: if query q is a data point in P, a greedy routing will always terminate at q. The query time is $O(n^{2/m} \ln n)$ when the points in P are from a uniform distribution in the m-dimensional Euclidean space [22]. However, the search path length can be as large as $O(n)$ in the worst case, and when $q \notin P$, MRNG does not guarantee finding even an ANN.

The Vamana graph in the DiskANN method [62] employs a more relaxed edge pruning rule by setting $r = \delta(p, u)/\alpha$ (see Figure 1b) where $\alpha > 1$ is a parameter. When $\mathcal{V} = P \setminus \{p\}$ for each $p \in P$, we refer to the constructed PG as the *slow preprocessing version* of Vamana. It can be verified that the distance of every visited node on the search path to the query point decreases by an α-multiplicative factor when $q \in P$, but this does not hold when $\delta(q, v^*) > 0$. Recently, Indyk and Xu [35] studied the worst-case performance of popular PGs (such as HNSW, NSG, and Vamana) when $\delta(q, v^*)$ can be arbitrarily large. They proved that only (the slow preprocessing version of) Vamana provides approximation guarantees: a greedy routing finds an $(\frac{\alpha+1}{\alpha-1} + \epsilon)$-ANN of q in $O(\alpha^d \cdot \log \Delta \cdot \log_\alpha \frac{\Delta}{(\alpha-1)\epsilon})$ time. Here, Δ is the *aspect ratio*[3] of P, and d is the *doubling dimension* of the dataset (an intrinsic dimensionality, see Section 2.3). However, Vamana still cannot find the exact NN when $q \notin P$.

A more practical scenario between $\delta(q, v^*) = 0$ and $\delta(q, v^*) = \infty$ is the case when $\delta(q, v^*)$ is bounded by a small constant $\tau > 0$, as observed by [31, 56] in real-world datasets. Recently, Peng et al. [56] proposed the τ-monotonic graph (τ-MG) such that after each step in a greedy routing, the distance between the next visited node and query point q is reduced by a τ-additive factor. τ-MG is constructed with $\mathcal{V} = P \setminus \{p\}$ for each $p \in P$ and another edge pruning rule by setting $r = \delta(p, u) - 3\tau$ (see Figure 1c). The average search time of τ-MG is $O(n^{1/m}(\ln n)^2)$ when P is from a uniform distribution. But as the starting point may have a very large distance to q, the worst-case search time is still $O(n)$.

Motivation. As summarized in Table 1, no existing algorithm can find the exact NN in poly-logarithmic time under the assumption that $\delta(q, v^*) \leq \tau$, even when the dataset has a constant doubling dimension. Solving this problem will address an open question in the theoretical foundation of proximity graphs. Another question is whether we can combine the advantages of Vamana with τ-MG and propose a unified framework that, when $\delta(q, v^*) \leq \tau$, finds the exact NN, and when $\delta(q, v^*) > \tau$, returns an ANN, with search times consistently poly-logarithmic. In this paper, we will design a new PG framework that addresses these questions affirmatively.

From a practical standpoint, our goal is to enhance both the accuracy and efficiency of existing PG algorithms. We seek to develop practical PGs that provide robust query performance on real datasets, regardless of whether the condition $\delta(q, v^*) \leq \tau$ holds.

[2]Given any point $p \in \mathcal{X}$ and $r > 0$, the *ball* $B(p, r)$ with radius r represents the set containing all the points in $\mathcal{X}$ whose distance to p is at most r.

[3]The ratio between the maximum and minimum pairwise distances of P.

1.2 Our Contributions

A new proximity graph with enhanced guarantees. We propose a new PG, the *α-convergent graph* (α-CG), in which we set $\mathcal{V} = P \setminus \{p\}$ for each $p \in P$, and then utilize a carefully designed edge pruning rule that incorporates both parameters τ and α to eliminate unnecessary candidates. Specifically, as illustrated in Figure 1d, we set $r = \frac{1}{\alpha}(\delta(p,u) - (\alpha+1)\tau)$, which reduces the radius of the ball $B(u, \delta(p,u))$ by first subtracting $(\alpha+1)\tau$ from $\delta(p,u)$ and then scaling a multiplicative factor of α. The intersection area is strictly smaller than that of the MRNG and Vamana.

With this new edge pruning rule, we prove that the α-CG admits non-trivial theoretical guarantees. When $\delta(q, v^*) \leq \tau$, our α-CG can find the exact NN of q in $O((\alpha\tau)^d \log \Delta \log_\alpha \Delta)$ time, thereby achieving the first algorithm that can find the exact NN in poly-logarithmic time when the doubling dimension is constant (see Table 1). A critical property of our PG is that after each hop, the distance from the next visited node to q is reduced by an α-multiplicative factor, and the search algorithm terminates in $O(\log_\alpha \Delta)$ steps. While τ-MG and the slow preprocessing version of Vamana terminates in $O(n)$ and $O(\log_\alpha \frac{\Delta}{(\alpha-1)\epsilon})$ steps, respectively. When $\delta(q, v^*) > \tau$, a greedy routing on α-CG can find an $(\frac{\alpha+1}{\alpha-1} + \epsilon)$-ANN of q for any $\epsilon > 0$, with the same query accuracy as that of Vamana. The space, query time, and construction time of α-CG match those of Vamana [35], up to an $O(\tau^d)$-factor (see Theorem 3).

A practical variant with efficient construction. To reduce the index construction time, similar to existing approaches, we propose a practical variant of our α-CG, the *α-convergent neighborhood graph* (α-CNG). The graph is constructed by generating a *local-neighbor set* $\mathcal{V}$ for each data point p (much smaller than $P \setminus \{p\}$), and subsequently applying our edge pruning rule to select a small subset of $\mathcal{V}$ as the out-neighbors of p.

Empirical analysis shows that the parameter α greatly affects the performance of our graph. Increasing α reduces the number of search steps but results in higher out-degrees in the PG. This trade-off complicates the selection of an optimal α, a problem that subsequent works [26, 37, 61, 62] on Vamana have overlooked. We propose an *adaptive local pruning* strategy that adjusts α for each node locally during graph construction. Starting from a small value, we gradually increase α and prune candidates until the node's out-degree reaches a predefined threshold, preserving long-distance shortcut edges and maintaining graph connectivity. To construct our graph efficiently, we implement a *distance-reusing* mechanism that stores and reuses intermediate computation results to accelerate the adaptive pruning process, along with a *lazy pruning* strategy that significantly reduces the number of pruning operations.

Experiments. We compared our α-CNG with 4 state-of-the-art PG indexes on 8 real-world datasets. At the same accuracy levels, α-CNG usually reduced distance computations by at least 15% and search steps by over 45% when compared with the best-performing baseline, while the maximum speedups in distance computations and search steps can be 2.28x and 2.88x, respectively. These improvements indidate the faster convergence of our method, making α-CNG particularly suitable for disk-based or distributed deployments, where I/O operations scale proportionally with search steps. Besides, both the index sizes and construction times of α-CNG are comparable to existing PGs (e.g., HNSW). We also validated our edge pruning rule by integrating it into HNSW and Vamana, and the results show that it improved efficiency for most datasets, suggesting the effectiveness of our pruning rule.

2 Preliminaries

2.1 Problem Setting and Basic Notations

A *metric space* $(\mathcal{X}, \delta)$ consists of a set $\mathcal{X}$ of points and a distance function $\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ satisfying the *triangle inequality* $\delta(p_1, p_2) + \delta(p_2, p_3) \geq \delta(p_1, p_3),\ \forall p_1, p_2, p_3 \in \mathcal{X}$. Let $P \subseteq \mathcal{X}$ be a

Algorithm 1 beam-search(G, q, s, L, k)

Input: graph G, query point q, entry point s, queue size L
Output: k ANN of q

```
candidate queue Q ← {s}
explored set E = ∅
while Q \ E ≠ ∅ do
    u* ← arg min{δ(x, q) | x ∈ Q \ E}
    for each out-neighbor v of u* do
        Q ← Q ∪ {v}
    end for
    keep the L entries in Q that are closest to q
    E ← E ∪ {u*}
end while
return k points in Q closest to q
```

set of n data points from $\mathcal{X}$. Given a query point q, a point $v^* \in P$ is a *nearest neighbor* (NN) of q if $\delta(v^*, q) \leq \delta(p, q)$ for all $p \in P$; while a point $p' \in P$ is a *c-approximate nearest neighbor* (c-ANN) of q for some constant $c > 1$ if $\delta(p', q) \leq c \cdot \delta(p, q)$ for all $p \in P$. We want to preprocess P into a data structure with a small space that can answer exact NN or ANN queries efficiently.

For any set $S \subseteq \mathcal{X}$, the diameter of S — denoted as $diam(S)$ — is the maximum distance of two points in S, while the *aspect ratio of* S is the ratio between $diam(S)$ and the minimum pairwise distance in S. We will use Δ to denote the aspect ratio of the input set P. In this paper, we will assume that the minimum pairwise distance in P is exactly 1, as can be achieved by scaling the distance function δ appropriately. Hence, we have $\Delta = diam(P)$.

This paper studies a practical scenario where the distance from the query point q to its NN v^* is bounded by a small constant $\tau \in (0, \Delta]$. Our objective is to develop a PG that can find the exact NN efficiently. For simplicity in the analysis, we assume $\tau \geq 1$. The scenario where $\tau < 1$ can be handled by using a solution designed for $\tau \geq 1$. We will explore how to support ANN queries efficiently when the condition $\mathrm{d}(q, v^*) \leq \tau$ does not hold.

Many real-world applications often require retrieving the top-k nearest neighbors. This leads to the *approximate k-nearest neighbor search* problem, where each query returns a set of k data points whose distances to q are no further than the other data points by at most a constant factor. To evaluate the accuracy of a method, empirical studies often rely on ranking-based metrics that compare the returned set with the true top-k results. A widely used metric is *recall*, which measures the average fraction of the true k nearest neighbors returned by the data structure.

2.2 Proximity Graphs

A *proximity graph* (PG) on P is a simple directed graph G whose vertices are precisely the points of P. We denote directed edges as (u, v), representing arcs from vertex u to vertex v, and define $N^+(u)$ as the set of out-neighbors of u in G.

Although various PG methods may use different strategies for connecting the edges in G, a simple greedy algorithm is commonly used to answer k-ANN queries. As shown in Algorithm 1, given a query point q, an entry (starting) point $s \in P$, and a queue size L, the *beam search* algorithm maintains a queue Q containing up to L of the closest points visited during search. We say a point u in Q is *explored* if the distances of its out-neighbors to q are computed. Initially, Q contains only the entry point s. At each step, the algorithm selects a point $u^* \in Q$ that is closest to q and has not

been explored. It then visits the out-neighbors of u^*, attempts to insert them into Q, and marks u^* as explored. We refer to u^* as a *hop vertex*. When all the points in Q are visited, the algorithm terminates by returning the k points in Q with the smallest distances to q. A special case is when $L = 1$, the algorithm — referred to as *greedy grouting* — traverses a sequence of hop vertices having descending distances to q, returning the last hop vertex as the answer.

Following the convention of previous works [22, 35, 56], this paper will assume that each distance computation consumes constant time. The query time of the beam search algorithm is asymptotically bounded by the total number of distance computations, which is the total out-degree of the visited hop vertices.

2.3 Doubling Dimension

Given a point p and a real value $r > 0$, we will use $B(p, r)$ to represent the set containing all the points in $\mathcal{X}$ whose distance to p is at most r; we refer to $B(p, r)$ as *a ball* centered at p with radius r. To analyze the performance of PG-based algorithms, we introduce the notion of doubling dimension, which is often used to measure the "intrinsic dimensionality" of high-dimensional point sets [10, 29, 41]:

Definition 1 (Doubling Dimension). *Let $(\mathcal{X}, \delta)$ be a metric space. A finite dataset $D \subseteq \mathcal{X}$ is said to have* doubling constant λ *if, for any point $p \in D$ and radius $r > 0$, the set $D \cap B(p, r)$ can be covered by at most λ balls of radius $r/2$, and λ is the smallest number with this property. The doubling dimension of D is defined as $\log_2 \lambda$.*

The doubling dimension generalizes the Euclidean dimension. For any set $D \subseteq \mathbb{R}^m$, the doubling dimension of D is $O(m)$.[4] For example, when $m = 2$, any ball of radius r can be covered by 7 balls with radius $r/2$, meaning that the doubling constant of D is at most $\log_2 7$. Moreover, empirical studies [19, 35] showed that the doubling dimension of real data sets is often smaller than their ambient dimension, e.g., dimension d of the Euclidean space. Following the previous studies on doubling dimension [15, 30, 41], this paper assumes that the doubling dimension of input P — denoted as d — is $O(1)$, namely, P has a low doubling dimension. Based on the definition of doubling dimension, we can derive [35]:

Lemma 1. *Consider any set P of points with doubling dimension d. For any point $p \in P$ and real values $R \geq r > 0$ satisfying $R/r = O(1)$, the set $P \cap B(p, R)$ can be covered by $O((R/r)^d)$ balls of radius r. Formally, there exist $p_1, \ldots, p_s \in P$ such that:*

$$P \cap B(p, R) \subseteq \bigcup_{i=1}^{s} B(p_i, r) \text{ and } s = O((R/r)^d).$$

3 Alpha-Convergent Graph

This section presents a new PG, called the *α-convergent graph (α-CG)*, which offers stronger theoretical guarantees on query time and accuracy. Similar to Vamana and τ-MG, its construction time is $\Omega(n^2)$. We will provide a practical variant in Section 4 to reduce the construction time.

3.1 A New Pruning Strategy

Recall that in PG construction, for each point $p \in P$, we need to select an appropriate subset from a candidate $\mathcal{V}$ as the out-neighbors of p. We consider the following sub-problem:

Definition 2 (Neighbor Selection [5]). *For each point $p \in P$ and a candidate set $\mathcal{V} \subseteq P \setminus \{p\}$ of points, select a subset of $\mathcal{V}$ as the out-neighbors of p, while maintaining graph connectivity.*

[4] https://en.wikipedia.org/wiki/Doubling_space

Algorithm 2 pruning(p, $\mathcal{V}$, α)

Input: point p, candidate set $\mathcal{V}$, parameter α
Output: a shortcut set S of p on $\mathcal{V}$

```
sort V in the ascending order of the distances to p
shortcut set S = ∅
for each u ∈ V (sorted) do
    if there exists a point v ∈ S satisfying inequality (1)
        then continue
    else S ← S ∪ {u}
end for
return S
```

An algorithm for this subproblem serves as a critical component in PG construction. The goal is to choose a "shortcut" subset from $\mathcal{V}$ such that if p is not the exact NN of q, then p connects to an out-neighbor v much closer to q than p. To achieve the purpose, we introduce the following edge pruning rule:

Definition 3 (Edge pruning rule of α-CG). *Given any point p and its candidate set $\mathcal{V}$, a point $u \in \mathcal{V}$ is pruned if there exists an out-neighbor v of p satisfying the following condition:*

$$\delta(p,u) > \alpha \cdot \delta(u,v) + (\alpha+1)\cdot\tau \tag{1}$$

where $\alpha > 1$ is a parameter.

It can be verified (1) is equivalent to $v \in B(p,\delta(p,u)) \cap B(u,r)$ where $r = \frac{1}{\alpha}(\delta(p,u) - (\alpha+1)\tau)$, as shown in Figure 1d. Given this pruning rule, we define a *pruning* procedure (Algorithm 2) to solve the problem in Definition 2. Specifically, we sort the points in $\mathcal{V}$ in ascending order according to their distances to p. Then, we iteratively select a subset S of $\mathcal{V}$ in the following manner. Initially, set $S = \emptyset$. For every point u in this sorted sequence, check if there exists a point $v \in S$ satisfying the condition (1). If not, add u into S; otherwise, skip u and proceed to check the next point. When all points in $\mathcal{V}$ are checked, the procedure returns S, which will be referred to as the *shortcut set of p* on $\mathcal{V}$. The next subsection will show that when $\mathcal{V} = P \setminus \{p\}$, the edges from p to nodes in S satisfy the "shortcutable" property with a fast convergence rate.

3.2 The α-Convergent Graph

Based on our new edge pruning rule, we define the α-CG G of P as follows:

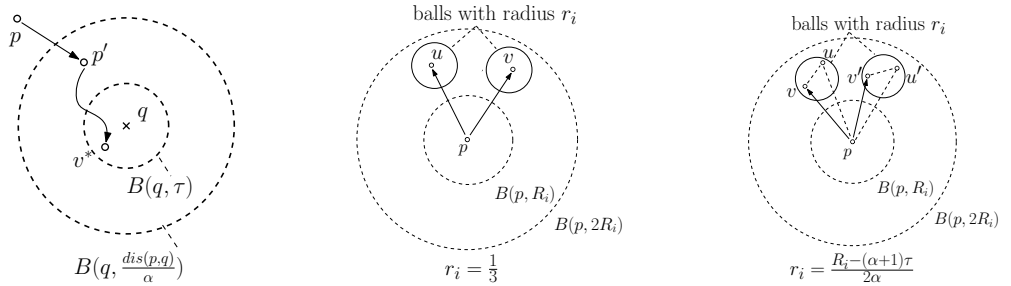
Definition 4 (α-convergent graph). *Every point of P is a vertex of G and vice versa. For each point $p \in P$, run the pruning procedure (Algorithm 2) with $\mathcal{V} = P \setminus \{p\}$, and define the returned shortcut set S as the out-neighbors of p in G.*

We begin by proving the following property.

Lemma 2 (α-reducible property). *Consider any point q whose exact NN $v^* \in P$ satisfies $\delta(q,v^*) \le \tau$. For every point $p \in P$ with $p \neq v^*$, either (i) v^* is an out-neighbor of p in G, or (ii) there exists an edge (p,p') in G such that $\delta(p',q) \le \delta(p,q)/\alpha$.*

Proof. Consider any $p \in P$ such that $p \neq v^*$ and v^* is not an out-neighbor of p in G. According to Definition 3, as v^* is pruned from p's candidate set, there must exist an out-neighbor p' of p such that (by setting $u = v^*$ and $v = p'$):

$$\delta(p,v^*) > \alpha \cdot \delta(p',v^*) + (\alpha+1)\cdot\tau. \tag{2}$$



(a) α-reducible property (b) Out-degree analysis: Case 1 (c) Out-degree analysis: Case 2

Fig. 2. Geometric illustrations of the pruning rule. (a) Illustration of the α-reducible property. (b,c) Two cases in the out-degree analysis.

Combining the above with the triangle inequality $\delta(p,q) \geq \delta(p,v^*) - \delta(q,v^*)$, we have

$$\begin{aligned} \delta(p,q) &\geq \delta(p,v^*) - \delta(q,v^*) \\ \text{(by (2))} &> \alpha \cdot \delta(p',v^*) + (\alpha+1)\cdot\tau - \delta(q,v^*) \\ &\geq \alpha \cdot \delta(p',v^*) + \alpha \cdot \tau \end{aligned} \tag{3}$$

where the last inequality used the assumption that $\delta(q,v^*) \leq \tau$. On the other hand, according to the triangle inequality, we have

$$\delta(p',q) \leq \delta(p',v^*) + \delta(v^*,q) \leq \delta(p',v^*) + \tau. \tag{4}$$

Inequalities (3) and (4) together imply that $\delta(p,q)/\delta(p',q) \geq \alpha$, finishing the proof of the lemma. □

See an illustration of the property in Figure 2a. Given any query q satisfying $\delta(q,v^*) \leq \tau$, Lemma 2 implies that greedy routing from any starting vertex quickly converges to the exact NN of q: at each step, if the current vertex is not v^*, then either the next hop reaches v^* or the distance from the next vertex to q decreases by an α factor.

3.3 Theoretical Analysis

This subsection will establish:

Theorem 3. *Let P be a set of n points and G be the α-CG of P. Consider any query point q whose exact NN in P is v^*.*

- *If $\delta(q,v^*) \leq \tau$, a greedy routing on G can find v^* in $O((\alpha\cdot\tau)^d \cdot \log\Delta \log_\alpha \Delta)$ time.*
- *Otherwise, a greedy routing on G can find an $(\frac{\alpha+1}{\alpha-1}+\epsilon)$-ANN of q in $O((\alpha\cdot\tau)^d \cdot \log\Delta \log_\alpha \frac{\Delta}{(\alpha-1)\epsilon})$ time, for any $\epsilon > 0$.*

The α-CG G has space $O(n\cdot(\alpha\tau)^d \log\Delta)$ and can be constructed in $O(n^2\cdot((\alpha\tau)^d \log\Delta + \log n))$ time.

The above result shows that our α-CG captures the Vamana graph. When $\tau = 1$, our α-CG achieves the same space and query time while providing a slightly better accuracy guarantee (it can find the exact NN when $\delta(q,v^*) \leq 1$). For $\tau > 1$, our method further improves query accuracy with only a modest increase in space and search time: α-CG can find the exact NN of q when $\mathrm{d}(q,v^*) \leq \tau$, with both space and query time incurring an additional $O(\tau^d)$ factor. Readers may observe the exponential dependence on α and τ; these are theoretical upper bounds, and in our experiments, small values of α and τ already yield strong empirical performance.

The rest of the subsection serves as a proof of Theorem 3. We first show that the maximum out-degree of G is $O((\alpha\tau)^d \log\Delta)$ and analyze its construction time. After that, we analyze the search path length of the greedy routing algorithm under the two cases $\delta(q,v^*) \leq \tau$ and $\delta(q,v^*) > \tau$.

3.3.1 **Space and Construction Time.** We first prove that G has a low maximum out-degree:

Lemma 4. *The α-CG G of P has maximum out-degree $O((\alpha \cdot \tau)^d \log \Delta)$, where d is the doubling dimension of P.*

Proof. Consider any point $p \in P$. The out-neighbors of p in G is the returned set of pruning(p, $P \setminus \{p\}$, α), which is the shortcut set S of p on $P \setminus p$. For each integer $i \in [0, \lceil \log_2 \Delta \rceil]$, define:

$$\text{Ring}_i = \{p' \in P \mid R_i < \delta(p, p') \leq 2R_i\}, \text{ where } R_i = \frac{\Delta}{2^{i+1}}. \tag{5}$$

Recall that Δ is the diameter of P, and the minimum pairwise distance of points in P is 1. For every $p' \in P$, because $\delta(p, p') \in [1, \Delta]$, there is a unique $i \in [0, \lceil \log_2 \Delta \rceil]$ such that $p' \in \text{Ring}_i$. Hence, we partition $P \setminus \{p\}$ into $\lceil \log_2 \Delta \rceil + 1$ subsets.

Consider any $i \in [0, \lceil \log_2 \Delta \rceil]$. We will prove that p has $O((\alpha \cdot \tau)^d)$ out-neighbors in Ring_i. Because there are $O(\log \Delta)$ rings, the out-degree of p is thus $O((\alpha \cdot \tau)^d \log \Delta)$.

Case 1: $R_i \leq 2(\alpha + 1)\tau$. In this case, we use balls of radius $r_i = \frac{1}{3}$ to cover the ball $B(p, 2R_i)$ and hence Ring_i. According to Lemma 1, the number of such balls is

$$\begin{aligned} O\left(\left(\frac{2R_i}{1/3}\right)^d\right) &= O\left(\left(\frac{4(\alpha+1)\tau}{1/3}\right)^d\right) \\ \text{(by } \alpha > 1) &= O\left(\left(\frac{8\alpha \cdot \tau}{1/3}\right)^d\right) = O((\alpha \cdot \tau)^d) \end{aligned}$$

where the last equality used the assumption $d = O(1)$. Since the minimum distance between any two points in P is 1, each ball of radius $\frac{1}{3}$ contains at most one point (see Figure 2b). Thus, $|\text{Ring}_i| = O((\alpha \cdot \tau)^d)$, meaning that p has $O((\alpha \cdot \tau)^d)$ out-neighbors in Ring_i.

Case 2: $R_i > 2(\alpha + 1)\tau$. We cover Ring_i using balls of radius $r_i = \frac{R_i - (\alpha+1)\tau}{2\alpha}$. For any two points $u, v \in \text{Ring}_i$, if u and v are in the same ball with radius r_i, then, we have:

$$\begin{aligned} \delta(u, v) &\leq 2r_i = \frac{R_i - (\alpha+1)\tau}{\alpha} \\ \Rightarrow \delta(u, v) \cdot \alpha &\leq R_i - (\alpha+1)\tau < \delta(p, u) - (\alpha+1)\tau \\ \Rightarrow \delta(p, u) &> \delta(u, v) \cdot \alpha + (\alpha+1)\tau. \end{aligned}$$

Similarly, we can obtain $\delta(p, v) > \delta(u, v) \cdot \alpha + (\alpha + 1)\tau$. According to the pruning rule defined by (1), at most one of u and v can be in the shortcut set S of p. Hence, for any ball B with radius r_i, at most one point in $\text{Ring}_i \cap B$ can be an out-neighbor of p. For instance, in Figure 2c, when v and v' are the our-neighbors of p, then edges (p, u) and (p, u') will not exist in our PG.

Next, we will show that $B(p, 2R_i)$, and hence Ring_i, can be covered by $O(\alpha^d)$ balls with radius r_i, implying p has $O(\alpha^d)$ out-neighbors in Ring_i. By Lemma 1, the number of balls with radius r_i needed to cover Ring_i is at most:

$$\begin{aligned} O\left(\left(\frac{2R_i}{r_i}\right)^d\right) &= O\left(\left(2R_i \cdot \frac{2\alpha}{R_i - (\alpha+1)\tau}\right)^d\right) \\ &= O\left(4\alpha\left(1 + \frac{(\alpha+1)\tau}{R_i - (\alpha+1)\tau}\right)^d\right) \\ \text{(by } R_i \geq 2(\alpha+1)\tau) &= O((8\alpha)^d) = O(\alpha^d) \end{aligned}$$

where the last inequality used the assumption $d = O(1)$.

Combining the above two cases and the fact $\tau \geq 1$, we conclude that the maximum out-degree of G is $O((\alpha \cdot \tau)^d \log \Delta)$. □

Construction time. The pruning(p, $\mathcal{V}$, α) procedure (Algorithm 2) finishes in $O(|\mathcal{V}| \cdot (\log |\mathcal{V}| + |S|))$ time. To see this, Line 1 of algorithm 2 finishes in $O(|\mathcal{V}| \log |\mathcal{V}|)$ time. For each loop of Line 3, we need to check if a point $u \in \mathcal{V}$ can be pruned by a point in S. A linear scan finishes in $O(|S|)$ time. Because there are $|\mathcal{V}|$ iterations, the total time of the algorithm is thus $O(|\mathcal{V}| \cdot (\log |\mathcal{V}| + |S|))$.

Recall that the out-neighbor set $N^+(p)$ of each point $p \in P$ is obtained by calling pruning(p, $\mathcal{V}$, α) with $\mathcal{V} = P \setminus \{p\}$. According to Lemma 4, we have $|S| = |N^+(p)| = O((\alpha \cdot \tau)^d \log \Delta)$, implying that the pruning procedure computes the out-neighbors of p in $O(n \cdot (|N^+(p)| + \log n)) = O(n \cdot ((\alpha \cdot \tau)^d \log \Delta + \log n))$ time. As $|P| = n$, we can conclude that the total construction time of the α-CG is $O(n^2 \cdot ((\alpha \cdot \tau)^d \log \Delta + \log n))$.

3.3.2 **Query Time. When $\delta(q, v^*) \leq \tau$.** The α-reducible property has an important implication on the behavior of the **beam search** algorithm, even when $L = 1$: If the current hop vertex p visited by the algorithm is not the exact NN v^*, p has an out-neighbor (thus the next hop vertex) whose distance to q is reduced by an α-multiplicative factor. Based on this property, we have the following result about the number of hop vertices visited:

Lemma 5. *Consider any query point q whose exact NN v^* in P satisfies $\delta(q, v^*) \leq \tau$. The greedy routing (beam search with $L = 1$) algorithm starting with any entry point s in G can find the exact NN of q by visiting $O(\log_\alpha \Delta)$ hop vertices.*

Proof. Let $\sigma = (p_1, p_2, ..., p_\ell)$ be the sequence of ℓ hop vertices visited by greedy grouting. We thus have $p_1 = s$. To see $p_\ell = v^*$, suppose the last visited hop vertex $p_\ell \neq v^*$. Then, according to Lemma 2, there exists an out-neighbor of p_ℓ in G that is closer to q than p_ℓ, contradicting that the algorithm terminates at p_ℓ.

Next, we prove $\ell = O(\log_\alpha \Delta)$. For any entry point $s \in P$, $\delta(s, q) \leq \delta(s, v^*) + \delta(v^*, q) \leq diam(P) + \tau \leq 2\Delta$. Recall that $\tau \leq \Delta$ and $diam(P)$ is the diameter of the dataset P; as the minimum pairwise distance in P is 1, we have $\Delta = diam(P)$ (see Section 2.1).

Our first claim is that for every $i \in [1, \ell - 2]$, we have $\delta(p_{i+1}, q) \leq \delta(p_i, q)/\alpha$. To see this, observe that v^* cannot be an out-neighbor of p_i; otherwise, the greedy grouting would terminate at p_{i+1}, visiting $i + 1 \leq \ell - 1$ hop vertices and thereby contradicting the fact that σ has size ℓ. Then, by Lemma 2, we have $\delta(p_{i+1}, q) \leq \delta(p_i, q)/\alpha$, and

$$\begin{aligned} \delta(p_{\ell-1}, q) &\leq \delta(p_1, q)/\alpha^{\ell-2} \\ \Rightarrow \ell &\leq \log_\alpha(\delta(p_1, q)/\delta(p_{\ell-1}, q)) + 2. \end{aligned} \tag{6}$$

Now, we prove that $\delta(p_{\ell-1}, q) \geq 1/2$. If $\delta(v^*, q) \geq 1/2$, then, since v^* is the exact NN of q in P, we have $\delta(p_{\ell-1}, q) > \delta(q, v^*) > 1/2$. On the other hand, if $\delta(v^*, q) < 1/2$, by triangle inequality and the fact that the minimum pairwise distance in P is 1, we obtain

$$\delta(p_{\ell-1}, q) \geq \delta(p_{\ell-1}, v^*) - \delta(q, v^*) \geq 1 - 1/2 = 1/2.$$

Hence, in both cases $\delta(p_{\ell-1}, q) \geq 1/2$. Plugging this bound into Inequality (6), we obtain

$$\ell \leq \log_\alpha(2\Delta/(1/2)) + 2 = O(\log_\alpha \Delta).$$

Thus, the algorithm finds the exact NN of q after visiting $O(\log_\alpha \Delta)$ hop vertices, and the lemma follows. □

According to Lemma 4, every vertex of G has a maximum out-degree at most $O((\alpha \cdot \tau)^d \log \Delta)$. Since a greedy routing can find the exact NN of q by visiting $O(\log_\alpha \Delta)$ hop vertices (Lemma 5), the search algorithm finishes in $O((\alpha \cdot \tau)^d \log \Delta \log_\alpha \Delta)$ time. This proves the first bullet of Theorem 3.

Algorithm 3 α**-CNG-construction**

Input: data points P, K for K-NN graph, out-degree threshold M, queue size L, candidate size C, initial parameter α_0, max parameter $\alpha_{\max}$, step size $\Delta\alpha$, runing parameter τ
Output: α-CNG G

/* Phase 1: approximate K-NN graph and entry point */
$G_0 \leftarrow$ **BuildApproxKNNGraph**(P, K)
$s \leftarrow$ **beam-search**$(G_0, \text{centroid}(P), \text{random\_point}, L, k = 1)$
/* Phase 2: candidate generation and pruning */
$G \leftarrow$ a graph with vertex set P and no edges
for each point $p \in P$ **do**
    $\mathcal{V} \leftarrow$ **beam-search**(G_0, p, s, L, C)
    $N^+(p) \leftarrow$ **adaptive-pruning**$(p, \mathcal{V}, M, \alpha_0, \alpha_{\max}, \Delta\alpha)$
end for
/* Phase 3: backward edge insertion and lazy pruning */
for each (u, v) in G, insert (v, u) into G
for each p with $|N^+(p)| > M$ **do**
    $N^+(p) \leftarrow$ **adaptive-pruning**$(p, N^+(p), M, \alpha_0, \alpha_{\max}, \Delta\alpha)$
end for
/* Phase 4: connectivity examination */
DFS-tree $T \leftarrow$ **DFS**(G, s)
for each $p \in P \setminus T$, add necessary edges from T to p
return G

When $\delta(q, v^*) > \tau$. We say a proximity graph G' is *α-shortcut reachable* if for every two vertices p, z of G' such that (p, z) is not in G', then, there exists an edge (p, p') in G' such that $\delta(p', z) \leq \delta(p, z)/\alpha$. Indyk and Xu [35, Theorem 3.4] proved that given an α-shortcut reachable PG, a greedy routing starting at any vertex can answer an $(\frac{\alpha+1}{\alpha-1}+\epsilon)$-ANN query after visiting $O(\log_\alpha \frac{\Delta}{(\alpha-1)\epsilon})$ hop vertices. Our α-CG G is also α-shortcut reachable:

- For any two vertices p, z such that (p, z) is not in G, by definition of the pruning rule (Definition 3), there must exist a vertex p' such that $\delta(p', z) < \frac{1}{\alpha}(\delta(p, z) - (\alpha+1)\tau) < \frac{1}{\alpha}\delta(p, z)$, implying that G is α-shortcut reachable.

Together with Lemma 4, we can conclude that the query time is $O((\alpha\tau)^d \log\Delta \log_\alpha \frac{\Delta}{(\alpha-1)\epsilon})$. The second bullet of Theorem 3 then follows. This finishes the proof of Theorem 3.
Remark. The main difference among our α-CG and existing PGs, such as MRNG, τ-MG, and Vamana, lies in the edge pruning rules, which significantly impact query performance. This paper finds a crafted edge pruning rule that leads to a new PG with enhanced guarantees, supported by a non-trivial theoretical analysis.

4 Alpha-Convergent Neighborhood Graph with Efficient Construction

Despite the superior asymptotic query performance of the α-CG, its construction time is $\Omega(n^2)$ in the worst case. In line with existing works such as NSG [22], Vamana [62], and τ-MNG [56], we propose a practical variant that approximates α-CG. This section also explores strategies for adaptively pruning candidates (setting the parameter α) and constructing the index efficiently.

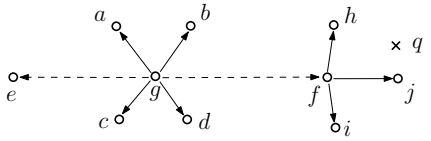


Fig. 3. When the shortcut set $S = \{a, b, c, d, e, f\}$ of g has size larger than $M = 4$, the two edges (g, e) and (g, f) are pruned.

4.1 The Overall Framework

We employ a standard framework utilized in existing PG methods, such as NSG [22] and τ-MNG [56], to generate a *local-neighbor set* (as opposed to $P \setminus p$) as the candidate set for each point $p \in P$. Then, we apply an adaptive pruning strategy (Section 4.2) to select up to M points as the out-neighbors of p, where M is a predefined maximum out-degree that ensures efficient graph storage and computation. We refer to resulting graph G as the *α-convergent neighborhood graph* (α-CNG). This framework consists of the following phases, as summarized in Algorithm 3.

The initial phase constructs an approximate K-NN graph G_0, where each data point is connected to its approximate K-NNs in P. Efficient implementations are available for constructing such graphs, e.g., [18, 31]. We then identify the *navigating node* s by performing a beam search, starting from a random point with the centroid of P (the geometric mean of all points in P) as the query point. The returned NN is designated as s.

Next, we generate a *local-neighbor set* $\mathcal{V}$ for each point p using G_0. We execute a beam search starting from node s, using p as the query. All points visited during this search (whose distances to p are computed) are gathered. Let $\mathcal{V}$ be the set of the C closest points to p that are collected. We then apply our adaptive pruning method to select a subset of at most M points from $\mathcal{V}$, which will be assigned as the out-neighbors of p in the α-CNG G.

Subsequently, backward edges are added to the graph for each edge inserted in the previous phase, ensuring bidirectional connectivity. If any vertex p's out-degree exceeds M, its out-neighbor set is pruned accordingly using our adaptive pruning method. Finally, we verify the graph connectivity by performing a depth-first search (DFS) starting from s. To ensure that all nodes are reachable from s, we add necessary edges for any nodes not included in the DFS tree, in line with the NSG method [22].

4.2 Adaptive Local Pruning

To incorporate our pruning strategy (Algorithm 2) into the above framework, a critical question is the parameter setting for α, which greatly impacts the query performance of our α-CNG. According to Lemma 2, each time a new hop vertex is visited, the distance to q decreases by at least a factor of α, unless the routing directly reaches the exact NN v^*. While a larger α accelerates convergence to the ANNs, it also leads to an exponential increase in the out-degree of each node (Lemma 4). Therefore, achieving a balance between out-degree and convergence rate is essential for efficient query performance. Despite the use of α in Vamana, a widely utilized PG, little attention has been paid to determining its optimal value.

Recall that in our framework, each node can have an out-degree of at most M. However, the shortcut set S returned by the pruning method (Algorithm 2) may exceed this size, particularly when α is large. A common approach used in the literature is to return the M closest points in S to p, but this may lead to the omission of long-distance shortcut edges. For instance, as illustrated in Figure 3, the vertices from a to f form the shortcut set of g. When $M = 4$, both e and f become disconnected from g, resulting in the pruning of the longest shortcut edges (g, e) and (g, f) from the PG. Consequently, given the query point q in Figure 3, a greedy routing will fail to find the exact

Algorithm 4 adaptive-pruning$(p, \mathcal{V}, M, \alpha_0, \alpha_{\max}, \Delta\alpha)$

Input: point p, candidate set $\mathcal{V}$, maximum out-degree threshold M, initial parameter α_0, max parameter $\alpha_{\max}$, step size $\Delta\alpha$
Output: a set of at most M shortcut points

1: $\alpha \leftarrow \alpha_0$
2: $S \leftarrow \emptyset$
3: **while** $|S| < M/2$ and $\alpha \leq \alpha_{\max}$ **do**
4: $\quad S \leftarrow$ **pruning**$(p, \mathcal{V}, \alpha)$
5: $\quad \alpha \leftarrow \alpha + \Delta\alpha$
6: **end while**
7: **return** M closest points of S to p

NN j of q when starting from node g. This example illustrates that when the size of the shortcut set exceeds the threshold M, the longest shortcut edges are omitted from the constructed graph, thereby reducing graph connectivity.

We propose an adaptive strategy (Algorithm 4) that gradually adjusts the parameter α and preserves as many long-distance shortcut edges as possible. Since each data point has a different local-neighbor set, a global α may lead to varying shortcut set sizes across data points. Therefore, we tune the parameter α for each data point p locally. Specifically, we start with a small initial value of $\alpha = \alpha_0$ and run the pruning method. Whenever the returned shortcut set S is smaller than the threshold $M/2$, we increase α by a small step size $\Delta\alpha$ and rerun the pruning method to find a larger S. Note that the size of S is highly sensitive to α, so we opt to increase α by a small value each step. This procedure aims to find the first α for which the corresponding shortcut set S exceeds the size of $M/2$. We then return the M points in S closest to p. This avoids selecting an excessively large α and S, which could lead to the pruning of long-distance shortcut edges. In the last iteration, the pruning method can terminate early once it has already collected M points.

4.3 Efficient Graph Construction

This subsection presents two optimizations to achieve efficient graph construction: (1) a distance reusing mechanism for adaptive edge pruning, and (2) a lazy pruning strategy for backward edge insertion. We also provide an analysis of the construction time. **A distance-reusing mechanism.** Although the adaptive pruning strategy offers a practical solution for tuning the parameter α and preserving long-distance shortcut edges, it introduces additional computational overhead. Recall that for each data point p and its candidate set $\mathcal{V}$, the adaptive algorithm calls the pruning method (Algorithm 2) multiple times for a sequence of increasing α values. As analyzed in Section 3.3, the running time of the pruning method is $O(|\mathcal{V}| \cdot (\log |\mathcal{V}| + |S|))$. Let $\alpha_0, ..., \alpha_h$ be the sequence of tested α values and $S_0, ..., S_h$ be the corresponding shortcut sets returned. Since we only need to sort $\mathcal{V}$ once (Line 1 of algorithm 2), the total time is $O(|\mathcal{V}| \cdot (\log |\mathcal{V}| + \sum_{i=0}^{h} |S_i|) = O(|\mathcal{V}| \cdot (\log |\mathcal{V}| + M \cdot h))$, as $|S_i| \leq M$ for each $i \in [0, h]$. It can be verified that the running time is dominated by the number of distance computations.

To reduce the computational overhead, we reuse the intermediate computation results based on the following observation. When invoking pruning$(p, \mathcal{V}, \alpha_i)$ to obtain the shortcut set S_i, according to inequality (1), a point $u \in \mathcal{V}$ is pruned if and only if there exists a point $v \in S_i$ satisfying the condition $\delta(p, u) > \alpha_i \cdot \delta(u, v) + (\alpha_i + 1) \cdot \tau$. Rearranging this leads to

$$\frac{\delta(p, u) - \tau}{\delta(u, v) + \tau} > \alpha_i. \tag{7}$$

Define $\bar{\alpha}(u,v) = \frac{\delta(p,u)-\tau}{\delta(u,v)+\tau}$. Hence, a point $v \in S_i$ can prune u if and only if $\bar{\alpha}(u,v) > \alpha_i$. When it is the first time to evaluate $\bar{\alpha}(u,v)$ for $\alpha = \alpha_i$, we can store and reuse it in subsequent iterations by simply comparing $\alpha(u,v)$ with α_j for $j > i$. This ensures that $\bar{\alpha}(u,v)$ is evaluated at most once for each pair $(u,v) \in \mathcal{V} \times \mathcal{V}$.

Denote by $S^+ = \bigcup_{i=0}^{h} S_i$, i.e., the set of all points in $\mathcal{V}$ that ever appeared in S_i for some $i \in [0,h]$. As we only need to compute $\bar{\alpha}(u,v)$ for $v \in S^+$ and $u \in \mathcal{V}$. The total number of stored (u,v) pairs and thus the number of distance computations is $O(|\mathcal{V}| \cdot |S^+|)$. Empirical analysis reveals that successive shortcut sets S_i and S_{i+1} exhibit significant overlap. The underlying reason is that as α increases, the pruning condition becomes more permissive, allowing most of the points in S_i to remain in S_{i+1}, which implies that $|S^+|$ is close to M. Therefore, with the distance-reusing heuristic, the total number of distance computations is close to $O(|\mathcal{V}| \cdot M)$ in practice (rather than $O(|\mathcal{V}| \cdot M \cdot h)$), and the overhead introduced by calling the pruning method multiple times becomes limited.

Lazy pruning during backward edge insertion. In the backward edge insertion phase (Section 4.1), the conventional way used by τ-MNG and NSG inserts each backward edge one by one. Whenever a node's out-degree exceeds M, a pruning procedure is invoked to enforce the out-degree constraint. We propose a lazy pruning strategy that invokes our adaptive pruning procedure at most once for each node. Specifically, for each point p, we first collect all backward edges starting from p and merge them with p's existing out-edges to form a candidate set $\mathcal{V}$. If $|\mathcal{V}| > M$, we invoke the adaptive-pruning procedure to select the final out-neighbors; otherwise, $\mathcal{V}$ is assigned directly as the new neighbor set of p.

Total construction time. The construction time of α-CNG is dominated by two parts: (i) approximate K-NN graph construction, and (ii) candidates pruning for all $p \in P$. We utilize the NN-decent algorithm [18] to compute the approximate K-NN graph, whose empirical time complexity is sub-quadratic.

Thanks to the lazy pruning strategy, we invoke the adaptive-pruning method at most twice for each $p \in P$ (once in phase 2 and once in phase 3). Since each candidate set $\mathcal{V}$ has a size at most C ($C \leq 500$ in our experiments), the running time for each adaptive pruning is $O(C \cdot (\log C + M \cdot h_{\max})$, where $h_{\max}$ is the maximum number of α values tested when pruning $\mathcal{V}$. Thus, the total running time is $O(n \cdot C(\log C + M \cdot h_{\max})) + f(n)$ where $f(n)$ is the running time of the NN-decent. We conclude that our construction time is sub-quadratic and comparable to existing practical PGs.

4.4 Discussions

Parameter configuration. Our experiments found that the shortcut set size may already reach $M/2$ when $\alpha < 1$ for certain data points. Therefore, the adaptive pruning method begins with $\alpha_0 = 0.9$; we set $\Delta\alpha = 0.05$ and $\alpha_{\max} = 1.6$.

Although τ is a problem parameter, in our experiments, we set τ to a small value within the range $[0, 10]$ (and in most cases, τ is smaller than 1) to avoid the large maximum out-degree. Both τ-MNG [56] and our paper found that the optimal settings for τ may vary across different datasets. Hence, we apply a grid search on the test queries. In practical applications, historical queries can be leveraged to determine this parameter. Specifically, we first compare the results obtained with $\tau = 0$ against those with $\tau \in \{10,\ 1,\ 0.1,\ 0.01,\ 0.001\}$ to identify a coarse range of τ, and then fine-tune τ to locate the best value.

Updates. This paper focuses on the development of static PGs that enhance both query accuracy and query time. Supporting updates is beyond the scope of this work. Nonetheless, concepts from existing research—such as periodic global rebuilding [56], lazy deletion with masking [61], and

Table 2. Dataset statistics

Dataset	Dim.	# Base	# Query	Source	Type
SIFT	128	1M	10K	[39]	Image
CRAWL	300	1.98M	10K	[52]	Text
WIKI	384	1M	1K	[51]	Text
MSONG	420	1M	200	[9]	Audio
LAION-I2I	768	1M	10K	[59]	Image
GIST	960	1M	1K	[39]	Image
DEEP100M	96	100M	10K	[7]	Image
BIGANN100M	128	100M	10K	[60]	Image

in-place update strategies [71]—could be applied to our α-CNG and may be considered for future work.

5 Experimental Evaluation

Section 5.1 describes the datasets and competing methods in our evaluation. Section 5.2 assesses the query performance of our approach compared to baselines, as well as its scalability on large datasets. Section 5.3 analyzes the impact of the parameters τ and α in our methods. Section 5.4 evaluates the index construction performance. Finally, Section 5.5 explores the effectiveness of our edge pruning rule by replacing those used in other PGs with our own.

5.1 Experiment Settings

All experiments were conducted on a Linux server equipped with an Intel(R) Xeon(R) Gold 6430 CPU and 512 GB RAM, running Ubuntu 20.04. All methods were implemented in C++ and compiled with g++ using the -O3 optimization flag.

Datasets. We utilized eight real-world datasets, which are widely adopted in ANN search evaluation [12, 20, 22, 27, 50, 56, 73]. These datasets include six at the 1M scale and two at the 100M scale. The 1M-scale datasets span various application domains, including image (SIFT, LAION-I2I, GIST), audio (MSONG), and text (WIKI, and CRAWL). We also assessed the scalability of our methods using the 100M-scale datasets, including DEEP100M[5] and BIGANN100M. Table 2 summarizes the key statistics, including the dimensionality (Dim.), the number of base points (# Base), the number of query points (# Query), data source, and data type.

Competing Methods. Our first method, α-CNG, employs an adaptive pruning strategy (Section 4.2). To evaluate the effectiveness of this strategy, we examined Fixed-α-CNG, a variant that employs a global α for all data points. Since previous studies [46, 68] have consistently shown that PG-based methods outperform non-PG methods such as LSH and IVF, we focus our comparisons on PG-based baselines. We compared our methods against four state-of-the-art PG algorithms: HNSW [50], Vamana [62], NSG [22], and τ-MNG [56], selected for their robust performance. Three additional popular algorithms—NSSG [20], DPG [46], and FANNG [31]—were excluded as τ-MNG [56] outperforms their performance. All baseline implementations use publicly available source code.

The construction parameters for all structures were selected based on official recommendations and empirical evaluations. HNSW used a configuration of $M = 32$ and $ef_C = 500$ for all datasets, while Vamana used the settings in [62], with $M = 70$, $L = 75$, and $\alpha = 1.2$.

[5]DEEP100M consists of the first 100 million vectors from the public DEEP1B[7] dataset.

NSG, τ-MNG, Fixed-α-CNG, and α-CNG shared core graph parameters K, M, L, and C. We set K=200 for all datasets. For SIFT and GIST, we used the settings M=50, L=40, C=500 and M=70, L=60, C=500, respectively, following NSG repository recommendations. For WIKI and LAION-I2I, we applied a configuration of M=70, L=60, and C=500. For MSONG, and CRAWL, we set M=100, L=100, and C=500. For the two large-scale datasets DEEP100M and BIGANN100M, we adopted the configurations of M=100, L=100, C=500 and M=80, L=100, and C=500 respectively. The pruning-related parameters, α and τ, were empirically tuned for optimal performance. For α-CNG, we set $\alpha_0 = 0.9$ (except for BIGANN100M, where $\alpha_0 = 1$ was used to include more long edges), $\Delta\alpha = 0.05$, and $\alpha_{\max} = 1.6$.

Metrics. Query accuracy was measured using recall@k, defined as recall@$k = \frac{|\text{KNN}(q) \cap \text{Res}(q)|}{k}$, where KNN($q$) denotes the exact k-NN of the query q, and Res(q) is the set of query results returned by the algorithm. Query efficiency was evaluated using two metrics: (1) the number of distance computations (NDC), which dominates the overall search cost and provides a platform-independent measure of efficiency; (2) the number of hops, defined as the number of hop vertices (search steps) visited by the search algorithm. For each dataset, we recorded the average recall@100, average NDC, and average number of hops of all provided queries.

5.2 Search Performance

Table 3. Speedups of our α-CNG in NDC and # hops over the best-performing baseline (in bold). For each dataset, all methods reached the same recall@100, i.e., 0.99 for SIFT and LAION-I2I, 0.95 for GIST at 0.95, and 0.90 for WIKI, CRAWL, and MSONG.

	NDC						# Hops					
Method	SIFT	CRAWL	WIKI	MSONG	LAION-I2I	GIST	SIFT	CRAWL	WIKI	MSONG	LAION-I2I	GIST
HNSW	4224	10284	5204	**12365**	N/A	8179	171	190	244	315	N/A	281
Vamana	4373	**9286**	4966	13150	11523	8177	**166**	**172**	223	**281**	**810**	232
NSG	3903	14837	4731	18289	**10297**	7677	175	233	210	456	904	272
τ-MNG	**3842**	10599	**4556**	14402	13776	**6971**	177	239	**189**	319	1004	**199**
α-CNG	3911	4067	4166	10438	6829	6024	143	102	124	167	282	137
Speedup	0.98x	2.28x	1.09x	1.18x	1.51x	1.16x	1.16x	1.69x	1.52x	1.68x	2.88x	1.45x

Recall vs. NDC. We first assessed the trade-off between (average) recall and (average) NDC on the six datasets at the 1M scale. Figure 4 reports the results obtained by varying the queue size L of beam search. Our α-CNG outperformed all baselines, achieving higher recall with fewer NDCs. The only exception is the SIFT dataset, for which all methods exhibited strong performance due to its low intrinsic dimensionality [56]. Fixed-α-CNG also surpassed the four baselines, except on MSONG, but was outperformed by α-CNG.

To provide a more detailed comparison, we report the NDC and number of hops at fixed recall@100 levels in Table 3. We configure different recall levels for the datasets based on their query performance in Figure 4, i.e., 0.99 for easy datasets SIFT and LAION-I2I, 0.95 for GIST, and 0.90 for difficult datasets CRAWL, WIKI, and MSONG. As shown in Table 3, our α-CNG reduces NDC by over 15% compared to the best-performing baseline on four out of the six datasets, and the maximum speedup can be 2.28x. We also observe that none of the baselines consistently outperformed the others, making our α-CNG a good choice due to its consistently good performance across different datasets.

Recall vs. # hops. We next evaluated the (average) number of hops by varying L. Figure 5 shows that α-CNG substantially reduced the number of hops across all datasets, and Fixed-α-CNG also outperformed the baselines on most datasets. Table 3 confirms that the reductions in the number of hops over the best-performing baseline are substantial: exceeding 45% on five out of the six datasets,

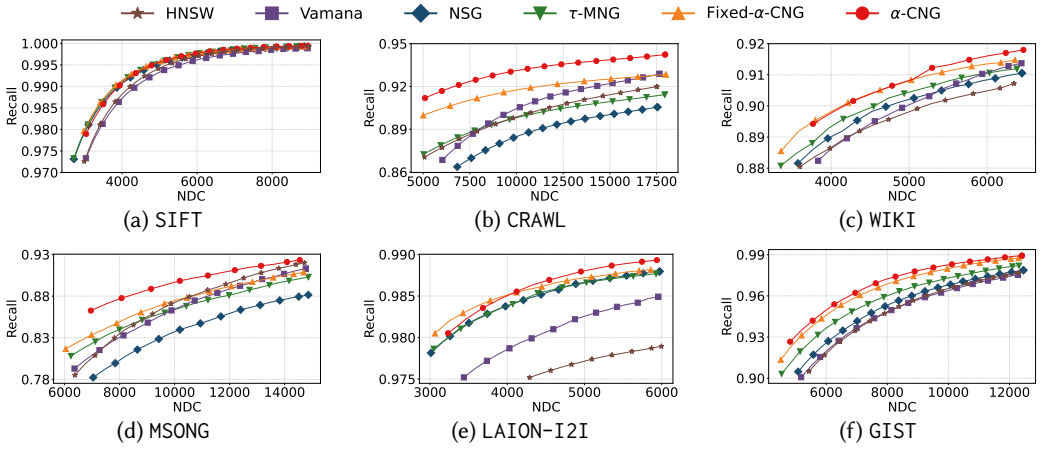


Fig. 4. Recall@100 vs. NDC

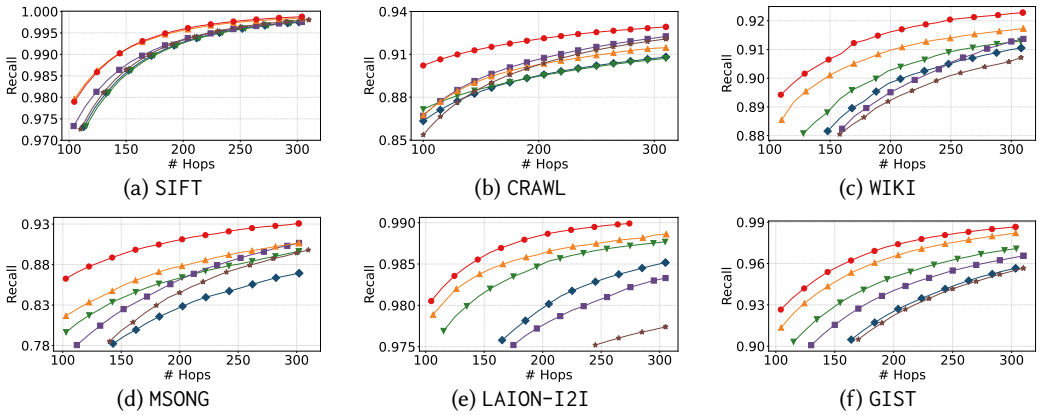


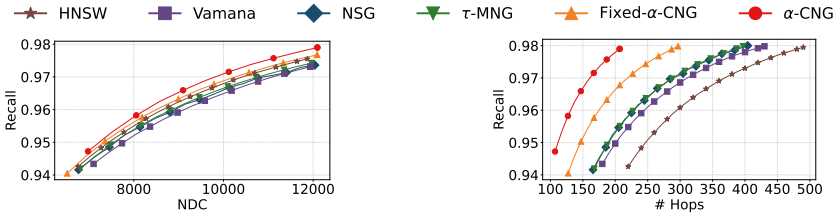
Fig. 5. Recall@100 vs. number of hops

with a maximum speedup of 2.88x. These gains arise from two key factors: α-CNG approximates α-CG for faster convergence, and our adaptive pruning strategy preserves more long-distance shortcut edges, effectively reducing the number of hops. The reduced hops of α-CNG are advantageous for disk-based [62] or distributed deployments of PGs, where hops correspond to I/O operations.

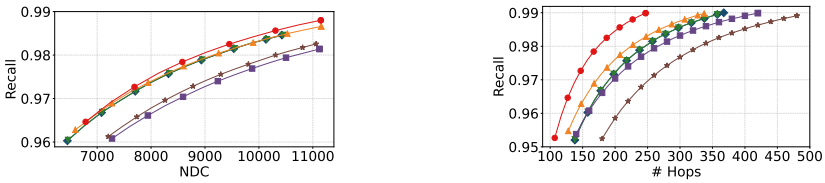
Scalability. Finally, we tested the scalability of all competing methods on the 100M-scale datasets (see Figure 6). Although the improvements in NDC are less pronounced than those observed at 1M-scale, α-CNG consistently outperformed all baselines, and Fixed-α-CNG matched the best baseline. Both our variants significantly reduced the number of hops (over 40% when recall@100=0.98), demonstrating that our methods converge quickly on large-scale datasets.

5.3 Effect of Parameters

This subsection explores the effects of τ and α in our edge pruning rule (Definition 3). We evaluated Fixed-α-CNG on three 1M-scale datasets with the highest dimensionality from different data types: WIKI for text, MSONG for audio, and GIST for images.

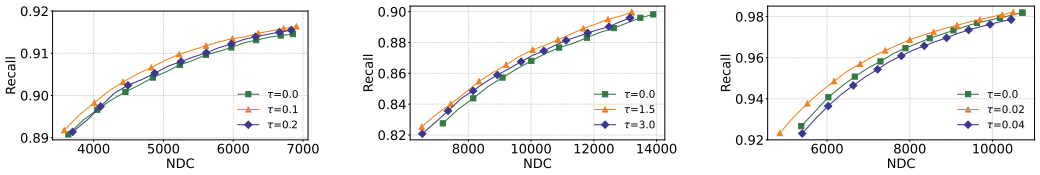


(a) DEEP100M



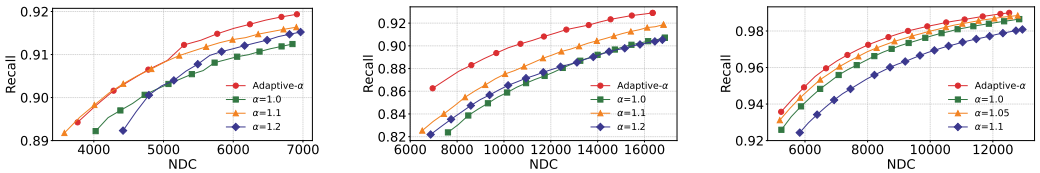
(b) BIGANN100M

Fig. 6. Recall@100 vs. NDC and Recall@100 vs. # hops on 100M-scale datasets



(a) WIKI (b) MSONG (c) GIST

Fig. 7. Recall@100 vs. NDC with varying τ



(a) WIKI (b) MSONG (c) GIST

Fig. 8. Recall@100 vs. NDC with varying α

Varying τ. Figure 7 illustrates the impact of the parameter τ on search performance, where the middle tested τ for each dataset is the optimal τ identified. We observe a trend similar to that reported for τ-MNG [56]: as τ increases from zero, search performance initially improves before deteriorating with further increases in τ. Recall that the search time is proportional to the total out-degrees of the hop vertices. A moderate increase in τ allows connecting more candidates and enhances the graph connectivity, thus reducing the number of hops. However, if τ increases excessively, the out-degree may become too large, as the search must compute distances for more neighbors at each step. Additionally, the increased out-degrees may exceed the limit M for some nodes, resulting in the replacement of long-distance shortcut edges with less useful ones. Overall, a small positive τ enhances graph connectivity and improves search performance, while excessively large values can lead to over-inclusion and reduced pruning effectiveness.

Varying α. We evaluated Fixed-α-CNG for three values of α, along with α-CNG (see Figure 8). The middle α value tested is the optimal value chosen for Fixed-α-CNG. We observe that increasing α

initially enhances the performance of Fixed-α-CNG, but it deteriorates as α continues to rise, for the same reasons observed with increasing τ. Notably, α-CNG outperformed Fixed-α-CNG for all tested α values. This advantage stems from its capability to locally adjust α, allowing each node to retain more shortcut edges while ensuring that its degree does not exceed M. Overall, α-CNG proves to be more practical, as it avoids the need for manual tuning of the parameter α while delivering improved query performance.

5.4 Construction Performance

Table 4. Index construction time and index size across six datasets.

	Index time (secs)						Index size (MB)					
Method	SIFT	CRAWL	WIKI	MSONG	LAION-I2I	GIST	SIFT	CRAWL	WIKI	MSONG	LAION-I2I	GIST
HNSW	42	159	91	145	116	206	256	509	256	254	256	256
Vamana	9	43	19	38	35	67	163	409	114	151	134	118
NSG	43	174	97	223	93	200	115	183	113	97	83	90
τ-MNG	43	182	98	242	98	217	113	329	124	117	127	131
Fixed-α-CNG	42	206	88	215	99	207	158	512	179	203	186	162
α-CNG	53	221	119	227	110	228	188	563	228	225	246	227
Vector Data Size	–						492	2284	1464	1597	2908	3666

Index time. Table 4 compares indexing times. Both our methods, α-CNG and Fixed-α-CNG, achieved performance comparable to that of baseline methods, except for Vamana. Compared to Vamana, our methods employ approximate K-NN graphs for candidate generation and a relaxed edge pruning rule that preserves more shortcut edges. This enhances graph connectivity, improving query performance as shown in Figure 4.

We further observe the following regarding our two methods: (1) Although using a more relaxed pruning, Fixed-α-CNG exhibited index times comparable to NSG and τ-MNG, and occasionally outperformed them. This efficiency stems from our lazy pruning strategy during backward edge insertion, reducing invocations of the pruning procedure. (2) Despite α-CNG iteratively pruning candidate sets for multiple α values, its index time remains only marginally higher than Fixed-α-CNG. This is due to our distance-reusing mechanism that optimizes the total number of distance computations.
Index Size. Table 4 presents the sizes of the indexes and the corresponding raw vector data. The index sizes of our methods are comparable to those of HNSW but larger than the other baselines. This is because we retain additional long-distance shortcut edges to accelerate the convergence to ANN and enhance query performance, as validated in Section 5.2. Since the PGs are usually much smaller than the raw vectors, our methods yield only a marginal increase in the overall data size.

5.5 Effectiveness of Our Edge Pruning Rule

Our two methods, NSG, and τ-MNG, share a framework that generates candidate sets from approximate K-NN graphs. Section 5.2 shows the efficacy of our pruning rule (Definition 3) within this framework. We further evaluated its effectiveness on two widely used PGs — HNSW and Vamana — by creating two variants, HNSW+ and Vamana+, through the integration of our edge pruning rule. Figure 9 presents the recall-NDC trade-offs, with α-CNG included for reference. Both HNSW+ and Vamana+ consistently outperformed their original counterparts. HNSW+ achieved more substantial gains over HNSW and even surpassed α-CNG on MSONG. We attribute this difference to the distinct pruning strategies used by Vamana and HNSW: Vamana employs a more relaxed pruning rule by setting $r = \delta(p,u)/\alpha$ (Fig. 1b), which incorporates the parameter α, while HNSW uses the strict pruning rule with $r = \delta(p,u)$ (Fig. 1a). These experimental results demonstrate the transferability of our edge pruning rule to enhance the query performance of other PG solutions.

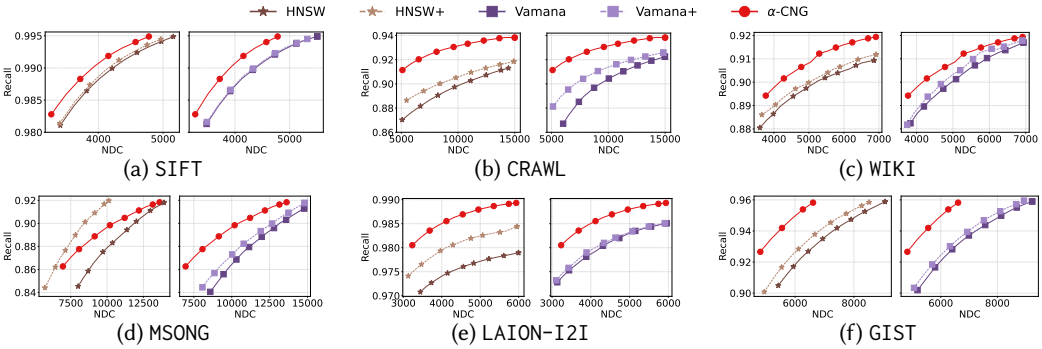


Fig. 9. Effect of integrating our edge pruning rule into HNSW and Vamana

Although α-CNG, HNSW+, and Vamana+ share the same α-pruning rule, α-CNG still achieves better performance because it constructs candidate neighbor sets of higher quality. Specifically, α-CNG first builds a K-NN graph and performs edge pruning over approximate nearest neighbors obtained from that graph. In contrast, HNSW+ and Vamana+ incrementally insert nodes, so each node can only access neighbors that have already been inserted, leading to a limited and often suboptimal candidate set for pruning. Consequently, the edges retained by α-CNG better preserve global connectivity, resulting in higher recall and more efficient search.

6 Related Work

Approximate nearest neighbor (ANN) search has attracted considerable attention over the past two decades, leading to the development of diverse methodologies aimed at enhancing search performance. Recent experimental studies [3, 5, 45, 46, 68] indicate that PG-based approaches surpass other techniques, including hash-based methods [33, 34, 47, 49, 63, 64, 70], tree-based methods [8, 13, 40, 44, 65, 74], and inverted index-based methods [6, 39].

PG-based methods. The Delaunay Graph (DG) [4], one of the earliest PGs, is the dual graph of the Voronoi diagram with $O(n^{\lceil m/2 \rceil})$ space in the m-dimensional space. DG guarantees finding the exact NN but suffers from high out-degrees and unbounded search time.

Several PGs in the literature provide non-trivial query accuracy guarantees while maintaining graph sparsity. Inspired by the relative neighborhood graph [38], Arya et al. [2] and Fu et al. [22] proposed MRNG. However, it can find the exact NN only when $q \in P$. Fu et al. [21] introduced the satellite system graph (SSG) to support the case when $q \notin P$ and the input is randomly distributed, although the worst-case time remains unbounded. Harwood et al. [31] studied the scenario when $\delta(q, v^*)$ is at most a constant $\tau > 0$; when P is uniformly drawn from $\mathbb{R}^m$, their PG can find the exact NN in $O(n^{2/m}(\ln n)^2)$ time. Later, τ-MNG [56] improved the search time to $O(n^{1/m}(\ln n)^2)$. Recently, Indyk et al. proved that the slow preprocessing version of Vamana guarantees to find an $(\frac{\alpha+1}{\alpha-1} + \epsilon)$-ANN of q in $O(\alpha^d \cdot \log \Delta \cdot \log_\alpha \frac{\Delta}{(\alpha-1)\epsilon})$ time, regardless of whether $\delta(q, v^*) \leq \tau$. Refer to other theoretical works studying cases where $q \in P$ [17] or the data follows specific distributions [43, 58].

The worst-case construction time of all the aforementioned PGs is $\Omega(n^2)$. To reduce construction time, numerous practical PGs have been proposed in the literature (see [14, 22, 24, 27, 31, 48, 50, 56, 62, 67, 73, 75, 76] and the references therein). A recent survey by Azizi et al. [5] identifies several design paradigms for PGs and indicates that neighbor selection (Definition 2) is crucial for improving search performance, warranting further theoretical exploration. This paper aims to contribute to the understanding of this direction.

Additional directions. Research has explored methods to extend ANN search by incorporating attribute constraints, enabling data retrieval that satisfies both vector similarity and user-specified attributes [12, 26, 55, 66, 72, 77]. Other studies have focused on the efficient construction of PGs [18, 53, 73, 76] and on supporting updates [61, 69, 71]. Recently, quantization-based techniques [1, 23, 25, 28, 39, 54] have been developed to compress high-dimensional vectors, thereby accelerating distance computations. These techniques can be integrated into PGs to enhance both construction and search efficiency, as investigated in recent research [27, 61].

7 Conclusions

This paper introduces α-CG, a new PG structure for high-performance ANN search. Specifically, α-CG employs a well-designed pruning rule to eliminate ineffective candidates. We prove that, under a realistic assumption that the distance between the query and its NN is bounded by a constant τ, α-CG guarantees exact NN retrieval in poly-logarithmic time. Without this assumption, it ensures ANN search within the same complexity bounds. To reduce graph construction overhead, we develop an approximate variant α-CNG with an adaptive local pruning rule that avoids manually tuning the parameter α and preserves more useful shortcut edges. We also propose optimizations to accelerate graph construction further. Empirical results show α-CNG consistently outperforms state-of-the-art PG methods, achieving superior accuracy-efficiency trade-offs. Furthermore, our edge pruning rule demonstrates transferability, enhancing query performance when integrated into other popular PGs.

Acknowledgments

Shangqi Lu's research was supported by a start-up fund from HKUST-Guangzhou.

References

[1] Cecilia Aguerrebere, Ishwar Singh Bhati, Mark Hildebrand, Mariano Tepper, and Theodore L. Willke. 2023. Similarity Search in the Blink of an Eye with Compressed Indices. *Proceedings of the VLDB Endowment (PVLDB)* 16, 11 (2023), 3433–3446.

[2] Sunil Arya and David M. Mount. 1993. Approximate nearest neighbor queries in fixed dimensions.. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 271–280.

[3] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. *Information Systems* 87 (2020), 101374.

[4] Franz Aurenhammer. 1991. Voronoi Diagrams—A Survey of a Fundamental Geometric Data Structure. *Comput. Surveys* 23, 3 (1991), 345–405.

[5] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art. *Proceedings of ACM Management of Data (SIGMOD)* 3, 1 (2025).

[6] Artem Babenko and Victor Lempitsky. 2014. The Inverted Multi-Index. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 37, 6 (2014), 1247–1260.

[7] Artem Babenko and Victor Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2055–2063.

[8] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM (CACM)* 18, 9 (1975), 509–517.

[9] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the International Society for Music Information Retrieval Conference*. 10.

[10] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover Trees for Nearest Neighbor. In *Proceedings of International Conference on Machine Learning (ICML)*. 97–104.

[11] Leonid Boytsov, David Novak, Yury Malkov, and Eric Nyberg. 2016. Off the Beaten Path: Let's Replace Term-Based Retrieval with k-NN Search. In *Proceedings of Conference on Information and Knowledge Management (CIKM)*. 1099–1108.

[12] 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 ACM Management of Data (SIGMOD)* 2, 6 (2024), 1–27.

[13] Lawrence Cayton. 2008. Fast Nearest Neighbor Retrieval for Bregman Divergences. In *Proceedings of International Conference on Machine Learning (ICML)*. 112–119.

[14] Meng Chen, Kai Zhang, Zhenying He, Yinan Jing, and X. Sean Wang. 2024. RoarGraph: A Projected Bipartite Graph for Efficient Cross-Modal Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment (PVLDB)* 17, 11 (2024), 2735–2749.

[15] Richard Cole and Lee-Ad Gottlieb. 2006. Searching dynamic point sets in spaces with bounded doubling dimension. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*. 574–583.

[16] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google News Personalization: Scalable Online Collaborative Filtering. In *Proceedings of International World Wide Web Conferences (WWW)*. 271–280.

[17] Haya Diwan, Jinrui Gou, Cameron Musco, Christopher Musco, and Torsten Suel. 2024. Navigable Graphs for High-Dimensional Nearest Neighbor Search: Constructions and Limits. In *Proceedings of Neural Information Processing Systems (NuerIPS)*.

[18] Wei Dong, Moses Charikar, and Kai Li. 2011. Efficient k-Nearest Neighbor Graph Construction for Generic Similarity Measures. In *Proceedings of International World Wide Web Conferences (WWW)*. 577–586.

[19] Christos Faloutsos and Ibrahim Kamel. 1994. Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. 4–13.

[20] Cong Fu, Changxu Wang, and Deng Cai. 2021. High Dimensional Similarity Search with Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 44, 8 (2021), 4139–4150.

[21] 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 (PAMI)* 44, 8 (2022), 4139–4150.

[22] 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 (PVLDB)* 12, 5 (2019), 461–474.

[23] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2025. Practical and asymptotically optimal quantization of high-dimensional vectors in euclidean space for approximate nearest neighbor search. *Proceedings of ACM Management of Data (SIGMOD)* 3, 3 (2025), 1–26.

[24] Jianyang Gao and Cheng Long. 2023. High-Dimensional Approximate Nearest Neighbor Search: With Reliable and Efficient Distance Comparison Operations. *Proceedings of ACM Management of Data (SIGMOD)* 1, 2 (2023), 1–27.

[25] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proceedings of ACM Management of Data (SIGMOD)* 2, 3 (2024), 1–27.

[26] 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 International World Wide Web Conferences (WWW)*. 3406–3416.

[27] Yutong Gou, Jianyang Gao, Yuexuan Xu, and Cheng Long. 2025. SymphonyQG: Towards Symphonious Integration of Quantization and Graph for Approximate Nearest Neighbor Search. *Proceedings of ACM Management of Data (SIGMOD)* 3, 1 (2025), 1–26.

[28] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *Proceedings of International Conference on Machine Learning (ICML)*. 3887–3896.

[29] Anupam Gupta, Robert Krauthgamer, and James R. Lee. 2003. Bounded Geometries, Fractals, and Low-Distortion Embeddings. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 534–543.

[30] Sariel Har-Peled and Manor Mendel. 2006. Fast Construction of Nets in Low-Dimensional Metrics and Their Applications. *SIAM Journal of Computing* 35, 5 (2006), 1148–1184.

[31] Ben Harwood and Tom Drummond. 2016. Fanng: Fast Approximate Nearest Neighbour Graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5713–5722.

[32] Qiang Huang, Jianlin Feng, Qiong Fang, Wilfred Ng, and Wei Wang. 2017. Query-aware Locality-sensitive Hashing Scheme for LP Norm. *The VLDB Journal* 26, 5 (2017), 683–708.

[33] 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 (PVLDB)* 9, 1 (2015), 1–12.

[34] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*. 604–613.

[35] Piotr Indyk and Haike Xu. 2023. Worst-Case Performance of Popular Approximate Nearest Neighbor Search Implementations: Guarantees and Limitations. *Proceedings of Neural Information Processing Systems (NuerIPS)* 36 (2023), 66239–66256.

[36] Masajiro Iwasaki. 2016. Pruned Bi-directed k-Nearest Neighbor Graph for Proximity Search. In *International Conference on Similarity Search and Applications (SISAP)*. 20–33.

[37] 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* (2022).

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

[39] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 33, 1 (2010), 117–128.

[40] Norio Katayama and Shin'ichi Satoh. 1997. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. *Proceedings of ACM Management of Data (SIGMOD)* 26, 2 (1997), 369–380.

[41] Robert Krauthgamer and James R. Lee. 2004. Navigating Nets: Simple Algorithms for Proximity Search. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 798–807.

[42] Brian Kulis and Kristen Grauman. 2009. Kernelized Locality-Sensitive Hashing for Scalable Image Search. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2130–2137.

[43] Thijs Laarhoven. 2018. Graph-Based Time-Space Trade-Offs for Approximate Near Neighbors. In *Proceedings of Symposium on Computational Geometry (SoCG)*, Vol. 99. 57:1–57:14.

[44] Herwig Lejsek, Friðrik Heiðar Ásmundsson, Björn Þór Jónsson, and Laurent Amsaleg. 2008. NV-Tree: An Efficient Disk-Based Index for Approximate Search in Very Large High-Dimensional Collections. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 31, 5 (2008), 869–883.

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

[46] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate Nearest Neighbor Search on High Dimensional Data—Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 32, 8 (2019), 1475–1488.

[47] Kejing Lu and Mineichi Kudo. 2020. R2LSH: A Nearest Neighbor Search Scheme Based on Two-Dimensional Projected Spaces. In *Proceedings of International Conference on Data Engineering (ICDE)*. 1045–1056.

[48] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: Hierarchical Graph Structure Based on Voronoi Diagrams for Solving Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment (PVLDB)* 15, 2 (2021), 246–258.

[49] Kejing Lu, Hongya Wang, Wei Wang, and Mineichi Kudo. 2020. VHP: Approximate Nearest Neighbor Search via Virtual Hypersphere Partitioning. *Proceedings of the VLDB Endowment (PVLDB)* 13, 9 (2020), 1443–1455.

[50] Yu 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 (PAMI)* 42, 4 (2020), 824–836.

[51] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. *arXiv:1609.07843* (2016).

[52] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in Pre-Training Distributed Word Representations. In *Proceedings of the International Conference on Language Resources and Evaluation*.

[53] Naoki Ono and Yusuke Matsui. 2023. Relative NN-Descent: A Fast Index Construction for Graph-Based Approximate Nearest Neighbor Search. In *Proceedings of the ACM International Conference on Multimedia (ACMMM)*. 1659–1667.

[54] John Paparrizos, Ikraduya Edian, Chunwei Liu, Aaron J. Elmore, and Michael J. Franklin. 2022. Fast Adaptive Similarity Search through Variance-Aware Quantization. In *Proceedings of International Conference on Data Engineering (ICDE)*. 2969–2983.

[55] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. Acorn: Performant and Predicate-Agnostic Search over Vector Embeddings and Structured Data. *Proceedings of ACM Management of Data (SIGMOD)* 2, 3 (2024), 1–27.

[56] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient Approximate Nearest Neighbor Search in Multi-Dimensional Databases. *Proceedings of ACM Management of Data (SIGMOD)* 1, 1 (2023), 1–27.

[57] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. 2007. Object Retrieval with Large Vocabularies and Fast Spatial Matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1–8.

[58] Liudmila Prokhorenkova and Aleksandr Shekhovtsov. 2020. Graph-based Nearest Neighbor Search: From Practice to Theory. In *Proceedings of International Conference on Machine Learning (ICML)*, Vol. 119. 7803–7813.

[59] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. 2022. Laion-5B: An Open Large-Scale Dataset for Training Next Generation Image-Text Models. *Advances in Neural Information Processing Systems* 35 (2022), 25278–25294.

[60] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, et al. 2022. Results of the NeurIPS'21 Challenge on Billion-scale Approximate Nearest Neighbor Search. *Advances in Neural Information Processing Systems*, 177–189.

[61] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. 2021. Freshdiskann: A fast and accurate graph-based ann index for streaming similarity search. *arXiv:2105.09613* (2021).

[62] 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. *Proceedings of Neural Information Processing Systems (NuerIPS)* 32 (2019).

[63] 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 (PVLDB)* 8, 1 (2014), 1–12.

[64] Yao Tian, Xi Zhao, and Xiaofang Zhou. 2024. DB-LSH 2.0: Locality-Sensitive Hashing with Query-Based Dynamic Bucketing. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 36, 3 (2024), 1000–1015.

[65] Jingdong Wang, Naiyan Wang, You Jia, Jian Li, Gang Zeng, Hongbin Zha, and Xian-Sheng Hua. 2013. Trinary-Projection Trees for Approximate Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 36, 2 (2013), 388–403.

[66] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2023. An Efficient and Robust Framework for Approximate Nearest Neighbor Search with Attribute Constraint. In *Proceedings of Neural Information Processing Systems (NuerIPS)*.

[67] Mengzhao Wang, Weizhi Xu, Xiaomeng Yi, Songlin Wu, Zhangyang Peng, Xiangyu Ke, Yunjun Gao, Xiaoliang Xu, Rentong Guo, and Charles Xie. 2024. Starling: An I/O-Efficient Disk-Resident Graph Index Framework for High-Dimensional Vector Similarity Search on Data Segment. *Proceedings of ACM Management of Data (SIGMOD)* 2, 1 (2024), 1–27.

[68] 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 (PVLDB)* 14, 11 (2021), 1964–1978.

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

[70] Jiuqi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. 2024. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment (PVLDB)* 17, 9 (2024), 2241–2254.

[71] Haike Xu, Magdalen Dobson Manohar, Philip A Bernstein, Badrish Chandramouli, Richard Wen, and Harsha Vardhan Simhadri. 2025. In-Place Updates of a Graph Index for Streaming Approximate Nearest Neighbor Search. *arXiv:2502.13826* (2025).

[72] 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), 1–26.

[73] Shuo Yang, Jiadong Xie, Yingfan Liu, Jeffrey Xu Yu, Xiyue Gao, Qianru Wang, Yanguo Peng, and Jiangtao Cui. 2024. Revisiting the Index Construction of Proximity Graph-Based Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment (PVLDB)* 18, 6 (2024), 1825–1838.

[74] Peter N. Yianilos. 1993. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 311–321.

[75] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, et al. 2023. {VBASE}: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 377–395.

[76] 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 (PVLDB)* 16, 8 (2023), 1979–1991.

[77] Chaoji Zuo, Miao Qiao, Wenchao Zhou, Feifei Li, and Dong Deng. 2024. SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. *Proceedings of ACM Management of Data (SIGMOD)* 2, 1 (2024), 1–26.

Received July 2025; revised October 2025; accepted November 2025