Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces

Xi Zhao[1], Yao Tian[1], Kai Huang[1], Bolong Zheng[2], Xiaofang Zhou[1]
[1]Hong Kong University of Science and Technology, Hong Kong, China
[2]Huazhong University of Science and Technology, Wuhan, China
{xzhaoca,ytianbc,zxf}@cse.ust.hk,ustkhuang@ust.hk,zblchris@gmail.com

ABSTRACT

The approximate nearest neighbor (ANN) search in high-dimensional spaces is a fundamental but computationally very expensive problem. Many methods have been designed for solving the ANN problem, such as LSH-based methods and graph-based methods. The LSH-based methods can be costly to reach high query quality due to the hash-boundary issues, while the graph-based methods can achieve better query performance by greedy expansion in an approximate proximity graph (APG). However, the construction cost of these APGs can be one or two orders of magnitude higher than that for building hash-based indexes. In addition, they fail short in incrementally maintaining APGs as the underlying dataset evolves. In this paper, we propose a novel approach named LSH-APG to build APGs and facilitate fast ANN search using a lightweight LSH framework. LSH-APG builds an APG via consecutively inserting points based on their nearest neighbor relationship with an efficient and accurate LSH-based search strategy. A high-quality entry point selection technique and an LSH-based pruning condition are developed to accelerate index construction and query processing by reducing the number of points to be accessed during the search. LSH-APG supports fast maintenance of APGs in lieu of building them from scratch as dataset evolves. Its maintenance cost and query cost for a point is proven to be less affected by dataset cardinality. Extensive experiments on real-world and synthetic datasets demonstrate that LSH-APG incurs significantly less construction cost but achieves better query performance than existing graph-based methods.

PVLDB Reference Format:
Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, Xiaofang Zhou. Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces. PVLDB, 14(1): XXX-XXX, 2023.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Jacyhust/LSH-APG/tree/main/cppCode/LSH-APG.

1 INTRODUCTION

Given a dataset and a distance function, the nearest neighbor (NN) search problem aims to find the point in the dataset with the minimum distance to a given query point. It has applications in a wide range of areas such as machine learning [4], pattern recognition [43] and data mining [45]. It is well known that finding the exact NN in large high-dimensional datasets can be very time-consuming [28]. A more efficient and potentially more practical alternative is to perform an approximate nearest neighbor (ANN) search, which has been widely studied in the database community [2, 8, 12, 28]. There are many categories of methods designed for efficient ANN search, such as locality-sensitive hashing (LSH)-based methods [2, 20, 22], graph-based methods [19, 25, 38] and so on. The LSH-based methods are known for their robust theoretical guarantee on query result accuracy and simple and efficient implementation. By mapping points in a high-dimensional space to low-dimensional spaces via a set of LSH functions, it is possible to find a high-quality result with a guaranteed high probability by only checking the points around the query point in low-dimensional spaces [2, 22]. On the other hand, the graph-based methods build an approximate proximity graph (APG) where each data point is represented as a vertex in the graph, and there is an edge between two vertices if the corresponding points are sufficiently close to each other in the original space. When the query q comes, the search begins from an arbitrary point o (*i.e.,* the entry point) in the APG and computes the distance between o's neighbors and q. The closest neighbor to q is chosen as the next entry point. The process repeats until it reaches a point v, which is closer to q than any of its neighbors in the APG.

The graph-based methods have been extensively studied due to their higher query accuracy than the LSH-based methods in practice [19, 47]. Compared with LSH-based methods that may suffer from the problem of hash-boundary issues (*i.e.,* some points that are close in the original space may be mapped into different hash buckets, thus not being identified by LSH-based approaches as neighbors) [20, 22, 46], the graph-based methods can achieve much higher accuracy [19, 47]. However, the cost of APG construction can be one or two orders of magnitude higher than that of LSH-based methods for very large datasets [3, 32], as such an approach will require finding proper neighbors for every point in the dataset. To address this issue, many heuristic strategies such as NN-Descent [14] and HCNNG [39] have been proposed for reducing the construction cost of APGs. However, such strategies cannot guarantee the quality of built APG is good enough and lowers the query performance. In addition, dataset in our real life is dynamic in nature, but existing graph-based methods fail short in incrementally maintaining APGs as the underlying dataset evolves.

Motivated by the above observations, in this paper, we propose LSH-APG, a novel approach to build APGs from lightweight LSH indexes and facilitate efficient ANN query processing. By exploiting the properties of LSH functions and analyzing the problem using different structures of proximity graphs, LSH-APG can overcome the drawbacks of both LSH (*i.e.,* the hash-boundary issues) and graph-based methods (*i.e.,* high graph construction time and poor maintainability for dynamic datasets). It adopts LSH indexes to quickly retrieve some query results as the entry point for a search in a proximity graph, followed by using graph-based techniques to further improve the accuracy of the query result. To boost query

processing efficiency, an accurate and scalable pruning strategy is proposed to filter out neighbors which are far away from the query point. This strategy can significantly reduce the number of points that need to be accessed during the search on the graph. A consecutive insertion strategy is used to build the graph index. It handles the high construction cost issue with the help of the LSH framework. All points are consecutively inserted into the index structure where each point is regarded as a query point and inserted into the graph index based on its nearest neighbors. This strategy not only reduces the construction cost due to improved search efficiency using the LSH framework but also enables a formal correctness and complexity analysis for our approach.

In summary, our main contributions in this paper include:

- Based on a comprehensive analysis of state-of-the-art LSH-based methods and graph-based methods, a new solution named LSH-APG is developed for efficient index construction and ANN query in high-dimensional spaces. LSH-APG adopts a well-designed LSH framework to accelerate indexing and query processing for proximity graphs with higher-quality entry point selection and LSH-based pruning conditions. LSH-APG has a much lower construction cost without scarifying query efficiency or query quality.
- We provide a cost model to demonstrate the effectiveness of LSH-APG, which shows the expected query cost of LSH-APG is nearly independent on the cardinality of the dataset. The effectiveness of the LSH framework is also theoretically proven.
- Based on our construction strategy and index structure, an efficient update strategy is designed to maintain index structures as the database evolves. The expected cost of deleting/inserting a point is also nearly independent on the cardinality of the dataset.
- Extensive experiments show that LSH-APG can greatly reduce indexing cost and achieve the best trade-off between query processing efficiency and query accuracy compared with the existing methods for ANN queries.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 introduces the problem definition, basic concepts and analyzes the limitations in the existing methods. The framework of LSH-APG is presented in Section 4. The pruning condition and update strategies are discussed in Sections 5 and 6. Section 7 reports experimental results. We conclude this paper in Section 8.

2 RELATED WORK

We analyze the existing works by the categories of LSH-based methods [20, 22, 35, 44, 46], graph-based methods [19, 25, 38] and other methods [5, 6, 21, 23, 30, 49].

2.1 LSH-based Methods

LSH-based method is originally proposed in [12, 22, 28]. In these methods, LSH functions map data points in the high-dimensional space into several low-dimensional hash buckets and the ANN query is answered by checking the buckets where the query falls. However, to guarantee a high query accuracy and sub-linear query cost, these methods require preparing multiple suit of LSH indexes with different bucket width, which causes undesirably large index sizes and limits their applications. [44] and [20] addressed this issue via the *virtual rehashing* technique, but they are hard to achieve a very high query quality due to the hash boundary issue, a shortcoming shared by all static LSH-based methods. To relieve the hash boundary issue, recent studies focus on designing dynamic LSH methods that dynamically construct query-centric hash buckets for every query point via novel LSH frameworks, such as collision counting based strategy [27, 35, 36] and metric based strategy [34, 51]. However, their query cost is no longer sub-linear due to the overhead of dynamic bucketing. In this year, Tian et al. combined the dynamic query strategy with the static LSH framework and proposed DB-LSH, which achieved the best query complexity theoretically among the existing LSH-based methods [46].

2.2 Graph-based Methods

Graph-based methods use the proximity graphs to facilitate ANN search and have shown better query performance compared to other methods, in terms of both accuracy and efficiency [38, 39, 47]. However, the construction cost of exact proximity graphs, *e.g.,* Delaunay graph (DG) [31], relative neighborhood graph (RNG) [29], minimum spanning tree (MST) [39] and kNN graph (KNNG) [14], is at least $O(n^2)$, which is unaffordable for large-scale datasets.

To lower the construction cost, several indexing strategies tried to build an approximate proximity graph (APG) at the cost of a bit of query quality. Dong et al. [14] proposed NN-Descent that approximates KNNG. In this method, an APG is built from a random graph and the edges for each point are updated by conducting local search iteratively among close neighbors of the query. The construction complexity of NN-Descent is lowered to $\tilde{O}(n^{1.14})$. Due to its efficiency compared to the brute-force manner, NN-Descent algorithm is used in many graph based methods, such as EFANNA [17], NSG [19] and others [33, 50] and some derivatives are developed [8]. However, it needs iterating about 10 times to find the high-quality neighbors, which is still time-consuming. NSW [37] builds the approximate KNNG and DG via consecutive insertion strategy, where the point to be inserted is regarded as a query in the current graph, and connected to its neighbors to finish the insertion. This construction manner is very efficient, but it can cause hubness issue, *i.e.,* high out-degrees for some vertexes, which makes the query inefficient. HNSW [38] adopts the same strategy as NSW but limits the maximum degree for each point to alleviate the hubness issue. HCNNG [39] and VRLSH [15] cluster or partition the data into many subgroups, and then build an APG in each subgroup. It could reduce the construction cost of graph index to $O(n)$ but requires building the graph multiple times to achieve a good performance.

To further boost query efficiency, many studies proposed the neighbor selection strategies to diversify the distribution of neighbors [19, 25, 32, 38]. In this manner, the similar edges will be cut off to reduce the average degree of the graph based on some occlusion rules. NSG [19] and HNSW consider the distribution of neighbors by their distance. In these two methods, for any two neighbors u and v of o, the value of $dist(u, v)$ must be larger than $dist(o, u)$ and $dist(o, v)$. Otherwise, the longer edge between (o, u) and (o, v) would be discarded. DPG [32] and NSSG [18] consider the distribution of neighbors by the angle between two edges (o, u) and (o, v). DPG maximizes the average angle between any two edges of o and NSSG limits the angle between two edges no less than a given threshold. These strategies demonstrate better query performance,

Table 1: List of Key Notations.

Notation	Description
$\mathbb{R}^d$	d-dimensional Euclidean space
$\mathcal{D}$	The dataset
n	The cardinality of dataset
LID	The local intrinsic dimensionality of dataset
o, v, u	A data point
q	A query point
$\|o_1, o_2\|$	The distance between o_1 and o_2
$e = (o_1, o_2)$	The directed edge from o_1 to o_2
$h^*(o), h(o)$	Hash function
$\chi^2(m)$	The χ^2 distribution with freedom m
C_Q	The expected number of points accessed per query

but all of them bring much larger construction costs due to the extra computation cost for occlusion rules.

2.3 Other Methods

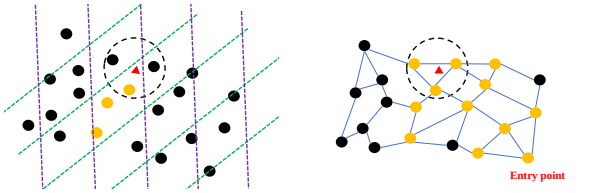
Tree-based methods [5, 6, 49] and quantization-based methods [21, 23, 30] are also able to solve ANN problems. Tree-based methods, such as M-Tree [9], R-Tree [24], and KD-Tree [40] and their variants, divide the space into subspaces using pivots or hyperplanes, which reduces the search space by only considering points in overlapping subspaces. However, as the data dimensionality increases, the efficacy of this partitioning strategy decreases, and almost all subspaces overlap with the search region [7, 48], making tree-based methods unsuitable for efficient ANN search in high-dimensional spaces. Quantization-based methods, such as VA-file [48] and Product Quantization (PQ) [21, 30], quantize the data and cluster it according to its quantization value, enabling the search for candidate points with the same quantization value as the query point. However, these methods can struggle with achieving high query accuracy due to quantization errors, particularly in high-dimensional spaces. Additionally, they lack any theoretical quality guarantees, unlike LSH-based methods. In summary, each category of methods has its specific applications. Tree-based methods are suitable for finding exact nearest neighbors in multi-dimensional spaces [5, 49]. Quantization-based methods perform well on well-clustered datasets [48]. However, when it comes to high-dimensional spaces, only LSH-based and graph-based methods can guarantee high query quality on any dataset.

3 PRELIMINARIES

In this section, we first introduce the problem definition, the concepts of LSH and graph-based methods. Then, a comprehensive analysis of limitations in the existing ANN methods is presented, which inspires us to design LSH-APG. Frequently used notations are summarized in Table 1.

3.1 Problem Definition

In this paper, we study the c-ANN and (c, k)-ANN queries in the Euclidean space. Let $\mathcal{D}$ be a set of points in d-dimensional Euclidean space $\mathbb{R}^d$ with cardinality $|\mathcal{D}| = n$. Let $\|o_1, o_2\|$ denote the Euclidean distance between points $o_1, o_2 \in \mathcal{D}$.



(a) Search in the LSH-Based Method (b) Search in the Graph-Based Method

Figure 1: Illustration of LSH-based and graph-based methods. The red triangle is the query point q. The three points in the black dashed circle are the 3NN of q. The orange points denote those that are accessed during the search.

Definition 1 ((c, k)-ANN Query [46]). *Given a query point q, an approximation ratio $c > 1$ and a positive integer k, a (c, k)-approximate nearest neighbor query returns k points $o_1, \ldots, o_k$ that are sorted in ascending order w.r.t. their distances to q. If o_i^* is the i-th nearest neighbor of q in $\mathcal{D}$, it satisfies that $\|q, o_i\| \leq c \cdot \|q, o_i^*\|$.*

Remark 1. *The c-ANN query is the (c, k)-ANN query with $k = 1$.*

Remark 2. *In the graph-based methods, we usually do not explicitly use c to control the query quality. Without confusion, we abbreviate (c, k)-ANN as kANN for simplicity.*

3.2 Locality Sensitive Hashing

Definition 2 (Locality Sensitive Hashing (LSH) [46, 51]). *Given a distance r and an approximation ratio $c > 1$, a family of hash functions $\mathcal{H} = \{h : \mathbb{R}^d \to \mathbb{R}\}$ is called (r, cr, p_1, p_2)-locality-sensitive, if for $\forall o_1, o_2 \in \mathbb{R}^d$, it satisfies both conditions below:*

(1) If $\|o_1, o_2\| \leq r$, $\Pr[h(o_1) = h(o_2)] \geq p_1$;
(2) If $\|o_1, o_2\| > cr$, $\Pr[h(o_1) = h(o_2)] \leq p_2$,

where $h \in \mathcal{H}$ is chosen at random, p_1, p_2 are collision probabilities and $p_1 > p_2$.

A typical LSH family in the Euclidean space is defined as follows [27]:

$$h^*(o) = \vec{a} \cdot \vec{o}, \tag{1}$$

where $\vec{o}$ is the vector representation of a point $o \in \mathbb{R}^d$ and $\vec{a}$ is a d-dimensional vector where each entry is chosen independently from the standard normal distribution.

Lemma 1. *Let $P(o) = (h_1^*(o), \ldots, h_m^*(o))$ be an m-dimensional vector where h_i^* is chosen from the LSH family in Eq. 1. Then, for $\forall o_1, o_2 \in \mathcal{D}$, we have $\dfrac{\|P(o_1), P(o_2)\|^2}{\|o_1, o_2\|^2} \sim \chi^2(m)$.*

Proof. This lemma can be derived based on the property of the normal distribution. □

Another commonly used LSH family in the Euclidean space is defined as follows [12]:

$$h(o) = \left\lfloor \frac{h^*(o) + b}{w} \right\rfloor, \tag{2}$$

where w is a pre-defined integer and b is a real number chosen uniformly from $[0, w)$. To distinguish these two LSH families, we call h^* as the projected function and h as the hash function.

3.3 Graph based Methods

The foundational structure of graph-based methods is a proximity graph, denoted as $G = (V, E)$, where the vertex set V represents all data points in the dataset $\mathcal{D}$, and the edge set E is the collection of all edges between vertices if the corespondent points are sufficiently close to each other in the original space. There are mainly three strategies to build the proximity graph [47].

- **Cluster & Merge [39].** The dataset $\mathcal{D}$ is clustered into several small groups and the exact proximity graph, such as MST [39], is built in each group. Then, we obtain the approximate proximity graph by merging all the subgraphs.
- **Iteration [14, 19].** The APG is built from an initial random graph where each vertex has several neighbors. For each vertex, we iteratively update its neighbors according to the local information, *e.g.,* its neighbors and the neighbors of its neighbors. The final APG is obtained when the iteration converges.
- **Consecutive Insertion [37, 38].** In this manner, the graph is built by inserting the point one by one, like that in R-Tree. When inserting a point o, we find the points close to o in the current graph and choose them as o's neighbors.

The number of edges directly affects the query performance [47]. Intuitively, the more the out-edges of a vertex, the more candidates and computations need to be performed, and the slower the query processing will be. There are two commonly used neighbor selection strategies to control the distribution of edges, *i.e.,* the simple selection strategy and the heuristic selection strategy. The simple selecting strategy is to choose the closest M neighbors, where M is a pre-defined threshold. In the heuristic selecting strategy, neighbors are chosen based on the distribution so as to preserve their diversity. Take HNSW and NSG for example, if the point o has two edges (o, u) and (o, v) that satisfies $\|o, u\| < \|u, v\| < \|o, v\|$, the edge (o, v) is said to be conflicted to (o, u) and the longer edge (o, v) will be discarded. The behind idea is that edges (o, u) and (o, v) are too similar, and thus there is no need to store both of them.

3.4 Limitations in the existing ANN methods

In this subsection, we analyze the drawbacks of the LSH-based and graph-based methods.

LSH-based methods. Figure 1(a) shows the framework of the LSH-based methods, where the data points are mapped into several hash buckets (parallelograms formed by purple and green lines). Then, the points in the 2-dimensional projected space are indexed by some simple structures, such as hash table, B+-Tree and R-Tree, which makes the LSH indexes easy to be maintained for dynamic datasets. To answer the ANN query, we only check the points in the bucket where the query point falls, *i.e.,* the 3 orange points in Figure 1(a). Both the LSH family in Equations 1 and 2 satisfy that the collision probability decreases monotonically with the distance between points. Thus, LSH-based methods provide a guarantee of query quality. However, it is hard for these methods to achieve a very high recall due to their simple query strategy and hash boundary issue. As shown in Figure 1(a), 2 of 3-NN results are not found by this LSH index. To find them, it requires to build more LSH indexes, and thus incurs higher query cost.

Graph-based methods. Figure 1(b) shows the framework of the graph-based methods, where each vertex in the graph has 2-4 neighbors. The query procedure begins from the bottom right point (a random entry point) and approaches to the correct results via the greedy search in the APG. The orange points are the points accessed during the search for the 3ANN of q. Although these methods hardly offer a theoretical guarantee, they always perform better than LSH-based methods. Experimental results show that to reach the recall of 0.95, the cost of DB-LSH is about 100 times higher than that of graph-based methods. The bottleneck of graph-based methods, however, comes from their huge construction cost. In the cluster and merge strategy, the subgraphs built for each cluster are disconnected and their graph quality is unsatisfactory since many close point pairs are divided into different clusters. Thus, we need to repeat the cluster and merge operations several times to improve the graph quality, which is time-consuming. Moreover, the graph built via this strategy requires being reconstructed as the data evolves since the clusters inevitably evolve with the data. In the iteration strategy, the construction cost is high because it requires updating the neighbors for every vertex in each iteration and the number of iterations to convergence increases with the data cardinality. Compared to them, the consecutive insertion strategy is more efficient and can be easily adapted to support the index maintenance for dynamic dataset, since the index is built point by point. However, the construction cost and graph quality of APG built via this strategy depends heavily on a well-designed query strategy. Besides, the neighbor selection strategies have a significant impact on the construction cost. The heuristic selection strategy greatly increases the computational cost since we need to compare every two neighbors. To lower its cost, some methods, such as HNSW [38], only adopt the heuristic selection strategy to cut off the edges of a vertex when its degree reaches a given maximum capacity. The simple selecting strategy is more efficient, but some edges could be similar especially when the data is in a dense region, which incurs the unnecessary computational cost in the query processing.

4 LSH-APG FRAMEWORK

Observe that LSH indexes have a relatively low recall although they can be quickly built and maintained as the underlying dataset evolves; while graph-based methods perform well in terms of query processing, they suffer from the high construction cost due to the complex construction strategy and edge selection strategy. To address this dilemma, we propose a novel framework called LSH-APG that can reduce the construction cost without sacrificing query performance for ANN search. The main idea is to quickly build APGs via a consecutive insertion and simple selection strategy, based on which the LSH framework is further adopted to accelerate the construction of APGs and the query processing by providing a closer entry point and filtering out some irrelevant edges. Consider Figure 2, LSH-APG selects the closest point to the query point q in the bucket where the q falls as the entry point (*i.e.,* the blue point), and thus the number of hops is reduced to 2, half of that in Figure 1(b). Then, it adopts the LSH-based pruning condition to avoid accessing all the neighbors during the search. In the figure, the grey point is one of neighbors of the red point, but we do not need to check it since its distance to q is much larger than that of

Algorithm 1: Building Naive-APG($\mathcal{D}, T, T'$)

Input: Dataset $\mathcal{D}$ and parameter T, T'
Output: $\mathcal{I}_G$

1 $\mathcal{I}_G \leftarrow \emptyset$;
2 **for** *each point* $o \in \mathcal{D}$ **do**
3 $candidates \leftarrow T$ ANN results of o found in $\mathcal{I}_G$;
4 **for** *each* $e \in candidates$ **do**
5 $\mathcal{I}_G \leftarrow \mathcal{I}_G \cup \{(o, e), (e, o)\}$;
6 **if** $e.degree > T'$ **then**
7 $o_f \leftarrow e$'s furthest neighbor in $\mathcal{I}_G$;
8 $\mathcal{I}_G \leftarrow \mathcal{I}_G - \{(e, o_f)\}$;
9 **return** $\mathcal{I}_G$;

the red point. Compared to existing methods, LSH-APG has the following advantages:

(1) *Low construction cost.* As the APG is built via the consecutive insertion strategy without considering the distribution of edges, the distance computation cost is greatly reduced. In addition, LSH framework further helps reduce the construction cost to nearly $O(n)$.
(2) *Guaranteed query cost and quality.* LSH-APG adopts a suit of lightweight LSH indexes to quickly find a better entry point for the query in the graph, which reduces the number of hops required to terminate the algorithm and ensures the worst query quality is bounded.
(3) *Accurate pruning condition.* LSH-APG filters out some edges during the search by an accurate yet efficient LSH-based pruning condition for reducing the unnecessary distance computations.
(4) *Incremental index maintenance.* LSH-APG supports incremental index maintenance in a low cost as the data evolves.

In what follows, we begin with a basic framework called Naive-APG (see Section 4.1), which can achieve (1). Then, we further develop LSH-APG by integrating the LSH framework with Naive-APG to realize (2) and (3) (Sections 4.2– 5). To accomplish (4), we present the index updating strategy (see Section 6) for incrementally maintaining the indexes as the underlying dataset evolves.

4.1 Naive-APG: the Basic Structure

The index of Naive-APG, denoted as $\mathcal{I}_G = (V, E)$, is a directed NN graph. $V = \mathcal{D}$ is the dataset and $E = (u, v)$ is the edge set where v is an approximate nearest neighbor of u in the dataset. Unlike the NN graph where each vertex connects to the unified number of edges, we allow the degree to vary in a range $[T, T']$ to better fit the data distribution, where T is the initial degree and T' is the maximum degree. An intuition explanation is that: The data points in dense regions are difficult to be differentiate from its nearby points, and thus requires more edges than data points in sparse regions. We build $\mathcal{I}_G$ via the consecutive insertion strategy as described in Algorithm 1. In this strategy, we find T ANN results for the incoming point o in the current graph index (Line 3). Next, we mutually connect o with its each ANN result e (Line 5). Finally, we check whether the degree of e reaches the maximum capacity T'. If so, we select the closest T' points to o_i in $N(o_i)$, following the simple neighbor selection strategy (Lines 6-8).

Remark 3. *When $T' = T$, all vertices in $\mathcal{I}_G$ will have a unified number of edges, as that in most NN graph. However, we find such a structure has a low performance due to the following reasons: 1) The real-world dataset is usually distributed unevenly and this setting does not consider this difference totally; 2) The points inserted earlier will often be accessed when we insert the points later. So, providing them more edges help the points coming later find the better ANN result. However, a too large T' will increase the number of distance computation during the search and incurs a high query cost. Considering these factors, we set $T' = 2T$ by default. More discussion about settings of T and T' is reported in the experiments (see Sec. 7.2.)*

4.2 LSH-APG: Optimized by LSH Indexes

Algorithm 1 guarantees the APG can be build in a rather low cost since the required distance computations just comes from finding ANN results for o and graph-based index has a low query cost. However, an underlying problem in this strategy is that the graph quality is not guaranteed. If we cannot find good enough ANN results for o, the o's edge quality is limited; in turn, a low-quality edge selection of o will prevent us from finding the high-quality ANN results for next-coming points. To address this issue, we adopt a suit of lightweight LSH indexes to quickly find a better entry point for the query in the graph to guarantee the query quality.

The chosen LSH indexes $\mathcal{I}_H$ need to be built quickly and answer an NN query efficiently. Since the final ANN results will be refined in the graph index, its query quality is not supposed to be very high, which is in line with the property of LSH indexes. To achieve this goal, we build our hash indexes $\mathcal{I}_H$ following the idea in [44]: first, we randomly choose K LSH functions $h_1, \ldots, h_K$ as described by Eq. 2. For a point o to be inserted, we compute its K hash values $h_1(o), \ldots, h_K(o)$ and consider them as a K-dimensional point $H(o) = (h_1(o), \ldots, h_K(o))$. Then, we transform $H(o)$ into a one-dimensional value $z(H(o))$ via the Z-order curve [11] and store the $z(H(o))$ in a B+-Tree (other sorted indexes are also suitable). Repeating this process L times, we build L B+-Trees, which forms $\mathcal{I}_H$. For the convenience, we denote the K LSH functions used the for i-th projected space as $H_i = \{h_{i,1}, \ldots, h_{i,K}\}, i = 1, \ldots, L$, *i.e.,* there are total $L \times K$ LSH functions used. Algorithm 2 describes how to build $\mathcal{I}_H$ and $\mathcal{I}_G$ in the same time, which is much more efficient than building them independently. Such an index $\mathcal{I}_H$ is proved to return good enough ANN results.

Lemma 2 (Theorem 1 of [44]). *By setting $K = O(\log n)$ and $L = n^\rho$ where $\rho = O(1/c')$, $\mathcal{I}_H$ can answer a c'^2-ANN ($c' \geq 2$) in $O(dL)$ query cost with at least constant probability of $1/2 - 1/e$.*

The lemma indicates that we can find a c'^2-ANN result with a small query cost $O(dL)$. As stated in [44], even single LSB-Tree is also expected to return results with high quality. In our method, $c' = 2$ and we set $K = O(1)$ and $L = O(1)$ following the mainstream LSH methods [36, 46].

Example 1. *Assume $T = 2$ and $T' = 4$, LSH-APG is like that in Figure 2. The parallelograms formed by purple and green lines are the hash buckets. When removing the hash buckets, we obtain Naive-APG.*

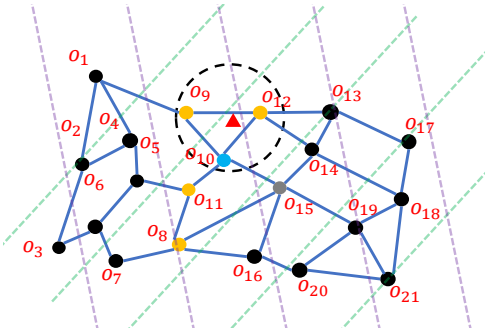


Figure 2: Search in LSH-APG

For simplicity, we consider all the edges are bi-directed. Then, there are total 64 edges. o_1 has 3 neighbors o_2, o_4 and o_9.

4.3 ANN Query in LSH-APG

The query algorithm not only decides the efficiency and accuracy of ANN search, but also affects the index quality and indexing efficiency. Algorithm 3 outlines the ANN query processing, which consists of two parts: find entry points using $\mathcal{I}_H$ (Line 1 - 4) and improve query quality using $\mathcal{I}_G$ (Line 5 - 21). The algorithm starts by computing the hash values of q (Line 1). Then, we can conduct a kANN search for q using $\mathcal{I}_H$ (Line 2). Since the search procedure in $\mathcal{I}_H$ is the same as that in the LSB-Tree [44], we do not give detailed description here due to the space limitation. According to the property of LSH, we can obtain high-quality results in $O(L)$ costs [44]. Next, we take points in EPs as the entry points to conduct a kANN query in $\mathcal{I}_G$. Specifically, in each iteration (hop), the nearest point e_p in EPs to q is popped from EPs (Line 8) and we denote the current found k-th NN as R_k (Line 9). If the distance between q and e_p is already greater than that between q and R_k, the algorithm terminates immediately (Line 10). Otherwise, we continue the search from e_p by checking its neighbors. To further boost query efficiency, we design an LSH-based pruning condition (Line 15) to reduce the number of neighbors checked. That is, for each neighbor o of e_p, if $\|P(q), P(o)\| > t \cdot \|q, R_k\|$, we filter out o without computing the distance $\|q, o\|$. According to Lemma 5, to be introduced in Section 5, we can guarantee that $\|q, o\| > \|q, R_k\|$ with the high probability p_τ, and thus it is reasonable to prune o directly. By updating EPs and the result set R iteratively, we get kANN of q finally. Generating entry points via the LSH framework greatly reduces the initial search radius. The reduction of the initial search radius can lower the hop numbers required to terminate the algorithm, and thus improve the query efficiency. Moreover, a closer entry point decreases the probability that the query terminates at a local minimal where the final search radius is still very high.

Example 2. *Let us take the points in Figure 2 as example to illustrate the kANN query with $k = 2$. We do not consider the LSH-based pruning condition in this example. When search in the LSH-APG, we can find o_8, o_{10} and o_{11} collide with the query point q (the red triangle). o_{10} and o_{11} is put in EPs. o_{10} is closest so far and we access it. Its neighbors o_9, o_{12} and o_{15} are accessed and put in EPs. o_{12} and o_9 are then accessed in order. The query terminates at o_9 since its farthest to q than 2-nd of q, o_{10}. The computation cost of this process is 8. On contrary, when search in the Naive-APG with o_{21} as the random entry point. We need to access o_{21}, o_{19}, o_{15}, o_{10}, o_12 and o_9 in order. The computation cost reaches 12, 50% higher than that of LSH-APG.*

Algorithm 2: Building LSH-APG($\mathcal{D}, T, T'$)

Input: Dataset $\mathcal{D}$ and parameter T, T'
Output: LSH-APG index $\mathcal{I}_G$ and $\mathcal{I}_H$

1 $\mathcal{I}_H \leftarrow \emptyset$, $\mathcal{I}_G \leftarrow \emptyset$;
2 **for** *each point* $o \in \mathcal{D}$ **do**
3 *candidates* $\leftarrow$ **call** kANN-Query$(o, \mathcal{I}_G, \mathcal{I}_H, p_\tau = 0.95, T)$;
4 **for** *each* $e \in$ *candidates* **do**
5 $\mathcal{I}_G \leftarrow \mathcal{I}_G \cup \{(o, e), (e, o)\}$;
6 **if** $e.degree > T'$ **then**
7 $o_f \leftarrow e$'s furthest neighbor in $\mathcal{I}_G$;
8 $\mathcal{I}_G \leftarrow \mathcal{I}_G - \{(e, o_f)\}$;
9 Insert o into the corresponding LSB-Tree in the $\mathcal{I}_H$;
10 **return** $\mathcal{I}_H$ and $\mathcal{I}_G$;

Algorithm 3: kANN Query$(q, \mathcal{I}_G, \mathcal{I}_H, p_\tau, k)$

Input: A query point q, LSH-APG index $\mathcal{I}_G$ and $\mathcal{I}_H, p_\tau, k$
Output: k nearest points to q

1 Compute q's projected values $h^*_1(q), \ldots, h^*_{L\times K}(q)$;
2 EPs$\leftarrow$ the set of k approximate nearest points to q in $\mathcal{I}_H$;
3 $V \leftarrow$ the set of visited points during the above search;
4 $R \leftarrow EP$s; //the result set of k best results found so far
5 $m \leftarrow K$, $P(q) \leftarrow (h^*_1(q), \ldots, h^*_m(q))$;
6 $t \leftarrow \sqrt{\chi^2_{p_\tau}(m)}$;
7 **while** $|EPs| > 0$ **do**
8 $e_p \leftarrow$ pop the nearest element in EPs to q;
9 $R_k \leftarrow$ the furthest points in R to q;
10 **if** $\|e_p, q\| > \|q, R_k\|$ **then**
11 **break**;
12 **for** *each* $o \in N(e_p)$ **do**
13 **if** $o \notin V$ **then**
14 $V \leftarrow V \cup \{o\}$;
15 **if** $\|P(q), P(o)\| < t \cdot \|q, R_k\|$ **then**
16 Compute $\|q, o\|$;
17 Insert o into EPs;
18 Insert o into R;
19 **if** $|R| > k$ **then**
20 Remove the furthest point to q in R;
21 **return** R;

4.4 Cost Model of LSH-APG

To demonstrate the performance of LSH-APG, we design a cost model to analyze the query cost and query quality of LSH-APG. First, we demonstrate the query cost and query quality is little affected

by the data cardinality (Theorem 2). Then, we prove that when the query terminates, the distance between returned points and the query point is small enough. Finally, we compute the benefit of LSH indexes on the query cost (Lemma 2). The rest parts are served as illustrating these three conclusions, with secondary lemmas and detailed proofs moved to the technical report [42].

Analysis of the query cost and quality. In our algorithm, the query cost C_Q mainly comes from the computation cost during search in the graph, which depends on the hop number of the search l, *i.e.,* the number of iterations from Line 7 to Line 20 in Alg. 3, and the average degree of a vertex in $\mathcal{I}_G$, T_e. Intuitively, C_Q can be expressed as $C_Q = lT_e$ [19]. Since T_e is bounded by T', we analyze the query cost via l. As for the query quality, we prove the final search radius s, *i.e.,* the search radius when the query terminates, is bounded. Assume the search in the graph begins from the entry point e_p with $\|q, e_p\| = r$, the query can terminate at e_p or go to the next hop with accessing another point e_n where $\|q, e_n\| = r'$. If the query terminate at e_p, $l(r) = 1$ and $s(r) = r$; otherwise, $l(r) = 1 + l(r')$ and $s(r) = s(r')$. Let $p(r)$ be the probability that the query terminates at e_n and $\delta(r) = r - r'$ be the hop length, $l(r)$ and $s(r)$ satisfy the following equations:

$$\begin{aligned} l(r) =& p(r) \cdot 1 + [1 - p(r)][l(r - \delta(r)) + 1], \\ s(r) =& p(r) \cdot r + [1 - p(r)]s(r - \delta(r)). \end{aligned} \tag{3}$$

If $p(r)$ and $\delta(r)$ are computed, we can determine the values of $l(r)$ and $s(r)$. Since the query terminates at e_n if and only if e_n is closer to q than e_p, we have $p(r) = \Pr[r' > r] = \Pr[\delta(r) > 0]$. Obviously, $\delta(r)$ depends on the neighbors of e_p. Thus, it is the local information of e_p and mostly independent on n. Hence, $p(r)$ is also independent on n. Based on Eq. 3, we have the following conclusion:

Theorem 1. *The query cost and query quality of LSH-APG is independent on n.*

In addition, it is obvious that $s(r)$ can be extremely small when $p(r) \approx 0$. Fortunately, we can decrease $p(r)$ to nearly 0 by increasing the number of edges T_e when $r > r_o$ where r_o is the average distance between e_p and its neighbors. We aim to estimate the bound of $s(r)$ by introducing some assumption. Assume T_e neighbors of e_p are uniformly distributed around it with the average distance r_o. That is, for one of e_p's neighbor o_i, its normalized point o_i' satisfies $\vec{e_p o_i'} = \vec{e_p o_i} / \|\vec{e_p o_i}\|$. Then, o_i' is uniformly drawn from a LID-dimensional unit sphere centered at e_p where LID is the local intrinsic dimensionality of $\mathcal{D}$. LID denotes how many "degrees of freedom" are necessary to describe the dataset [10] and is used to measure the difficulty of answering NN queries in the dataset [1]. Similar assumptions can be found in [19, 41]. Under such an assumption, $s(r)$ satisfies:

Theorem 2. *Assume the expected edge length of e_p is r_o and its neighbors are uniformly distributed around it, the final search radius s is expected to be bounded by $(1 + \gamma)r_o$ where $\gamma < 1$.*

Proof. (Sketch.) We derive the formation of $\delta(r)$ and prove that $p(r)$ can be as small as possible when T_e is large. Then, the query hardly terminates when $s(r) > (1+\gamma)r_o$. Due to the space limitation, we put the detailed proof in our technical report [42]. □

Remark 4. *In our cost model, we assume that e_p's neighbors are uniformly distributed around it. Additionally, this assumption shapes the specific form of $\delta(r)$, however, it does not impact the validity of our conclusion that the values of $l(r)$ and $s(r)$ are independent of n. This is because the neighbors of a vertex are primarily determined by the points in close proximity to it, rather than the entire dataset, which implies that the value of $\delta(r)$ is not heavily impacted by n.*

The benefit from $\mathcal{I}_H$. Since $l(s)$ depends on the initial search radius r_i. $\mathcal{I}_H$ provides us the closer entry point o_h, which essentially reduces the initial search radius. Assume that $\|q, o_h\| = r_1$ and a random entry point has the distance r_2 to q ($r_0 < r_1 < r_2$). Then, we denote the benefit from $\mathcal{I}_H$ as $B_{\mathcal{I}_H} = l(r_2) - l(r_1)$, which implies how many the number of hops is reduced by using $\mathcal{I}_H$.

Lemma 3. *Let $\Delta r = r_2 - r_1$ be the reduction of the initial search radius compared to the random entry point, the expected $B_{\mathcal{I}_H}$ is $\frac{\Delta r}{\lambda r_o}$ where $\lambda < 1$ and depends on the average degree T_e.*

Proof. (Sketch.) When $r \gg r_o$, $\delta(r)$ changes little and thus $B_{\mathcal{I}_H}$ is nearly directly proportional to Δr. Due to the space limitation, we put the detailed proof in our technical report [42]. □

This lemma indicates that the benefit of the entry point selection is $O(\Delta r)$. When $r > r_1$, $\delta(r)$ is nearly a constant and the search radius decreases slowly, which incurs a large query cost in the graph-based methods. On the contrary, although the LSH-based methods are hard to achieve as a high query performance as graph-based methods, they can quickly find a point o with $\|q, o\| = r_1 \leq c'r$, and thus improve the query performance of LSH-APG.

4.5 Complexity Analysis

Following the mainstream graph-based methods [19, 38, 39], we consider T and T' as a constant. According to Theorems 1 and 2, we have the following conclusion:

Theorem 3. *LSH-APG has the space complexity of $O(n)$ and can be built in $O(ndC_Q)$ time, where C_Q is the expected computation cost and independent on n. The query cost of LSH-APG is $O(dC_Q)$.*

5 LSH-BASED PRUNING CONDITION

The query cost of a graph-based method comes from the number of distance computations between q and the neighbors of e_p when accessing e_p. The previous query strategy checks all e_p's neighbors, which is unnecessary and time-consuming since some neighbors are obviously far away from q and unlikely to be the required result. Hence, we design an LSH-based pruning condition as follow to filter out some neighbors that might be far:

$$\|P(q), P(o)\| < \sqrt{\chi^2_{p_\tau}(m)} \cdot d_k, \tag{4}$$

where $P(o)$ is the m-dimensional projected vector defined as in Lemma 1, $\chi^2_p(m)$ is the quantile of $\chi^2(m)$ distribution at p_τ and d_k is the current found k-th best NN result. The intuition of the pruning condition is that when $\|o, q\|$ is greater than d_k, $\|P(q), P(o)\|$ will also be greater than $\sqrt{\chi^2_{p_\tau}(m)} \cdot d_k$ with high probability p_τ. Then, it is reasonable to discard o without computing the distance $\|q, o\|$.

Lemma 4 (Lemma 4 in [51]). *Given a query q, an approximation ratio c and parameter t, we define the following two events:*

- ***E1:*** *For a point o that $\|q,o\| \le r$, its projected distance to q, $\|P(q),P(o)\|$, is smaller than tr.*
- ***E2:*** *There are fewer than βn ($\beta > \alpha_2$) points whose distances to q exceed cr but projected distances to q are smaller than tr.*

Then, we have that the probability that E1 occurs is at least α_1, and the probability that E2 occurs is at least $1 - \frac{\alpha_2}{\beta}$, where $\alpha_1 = F(t^2; m)$ and $\alpha_2 = F(t^2/c^2; m)$.

Lemma 4 reveals that $\|P(q), P(o)\|$ can be a proper estimator of $\|q, o\|$. Based on it, we have:

Lemma 5. *With the LSH-based pruning condition, the probability that a point o is filtered increases with $\|q, o\|$. Assume $p_\tau = \frac{1}{2}$ and the current search radius is r, for any $c > 1$, the point whose distance to q is less than r will not be filtered with at least the probability of $\frac{1}{2}$ and we access at most $O(n^\alpha)$ points whose distance to q is greater than cr, where $\alpha = 1 - \frac{9\kappa(c^{-2/3}-1)^2}{4}$ and $\kappa = \frac{m}{\log n}$.*

Proof. (Sketch.) We derive the upper bound of number of points whose distance to q is greater than cr by using Lemma 4 and prove the conclusion. Due to the space limitation, we put the detailed proof in our technical report [42]. □

This lemma indicates the point whose distance to q is less than the search radius r will be checked with at least the probability of $\frac{1}{2}$. On the contrary, the farther a point is to q, the more likely it is filtered out. The number of checked points whose distances to q are greater than cr is bounded by $O(n^\alpha)$, which depends on the values of c, n and m. The total number of points filtered by the pruning condition depends on how many "far" points we meet. This pruning condition helps reduce the query cost and still guarantees the query quality. Usually, a large p_τ is used to better balance the query quality and query efficiency. We only prove this lemma with $p_\tau = \frac{1}{2}$ as an example. For other values of p_τ, α is also bounded.

Example 3. *When conducting the kANN query with $k = 2$ in the LSH-APG with LSH-based pruning condition, we need to access o_{10}, o_12 and o_9 in order, as that in Example 2. However, when accessing o_{10}, o_{15} can be filtered out. Likewise, when accessing o_{12}, o_{13} and o_{14} can be filtered out. Thus, the computation cost is reduced to 5.*

6 INDEX MAINTENANCE

As the data evolves, we need to update LSH-APG by reconstructing some edges for related points. The updates of LSH-APG includes the insertion and deletion. It is simple and natural to insert a new point into LSH-APG since LSH-APG is built via the consecutive insertion strategy. Therefore, we focus on designing an efficient deletion strategy. To delete a point o, we need to discard all the out-edges and in-edges of o in $\mathcal{I}_G$ and remove o from $\mathcal{I}_H$. It is trivial to remove a point from the $\mathcal{I}_H$, but it is challenging to delete in-edges in $\mathcal{I}_G$ since they are not recorded in the graph.

Let $RN(o) = \{v|(v,o) \in E\}$ be the set of reverse neighbors of o in $\mathcal{I}_G$ and $d_m = \max_{v \in RN(o)} \|o, v\|$ be the maximum length. We store the in-degree of o, $|RN(o)|$, and d_m in LSH-APG to facilitate the deletion, which is much space-saving than storing all the in-edges. Hence, Lines 1-2 only cost $O(1)$ time. To delete o, we first mark o and all the out-edges of o as the Deleting status (Line 4). Then, we

Algorithm 4: Delete-Point $(o, \mathcal{I}_G, \mathcal{I}_H, C_{Dm})$

Input: A point to be deleted o, $\mathcal{I}_G$, $\mathcal{I}_H$, C_{Dm}
Output: The updated indexes $\mathcal{I}_G$ and $\mathcal{I}_H$

1 $RN(o) \leftarrow \{v|(v,o) \in E\}$;
2 $d_m \leftarrow \max_{v \in RN(o)} \|o, v\|$;
3 Delete o from $\mathcal{I}_H$;
4 Mark o and all the out-edges of o as the Deleting status;
5 $EPs \leftarrow \{v|(o,v) \in E\}$;
6 $V \leftarrow \emptyset$ stores the set of visited points;
7 $m \leftarrow K$, $t \leftarrow \sqrt{\chi^2_{p_\tau}(m)}$,$cnt \leftarrow 0$;
8 $t \leftarrow \sqrt{\chi^2_{p_\tau}(m)}$;
9 $d_k \leftarrow$ the current k-th **while** $|EPs| > 0 \&\& cnt < C_{Dm}$ **do**
10 $cnt \leftarrow cnt + 1$;
11 $e_p \leftarrow$ pop the nearest element in EPs to q;
12 **for** *each* $u \in N(e_p)$ **do**
13 **if** $u \notin V$ **then**
14 $V \leftarrow V \cup \{u\}$;
15 **call Access**(u);
16 **Function Access**(u) **is**
17 **if** $\|P(q), P(u)\| < t \cdot d_m$ **then**
18 Compute $\|q, u\|$;
19 Insert u into EPs;
20 **if** $u \in RN(o)$ **then**
21 Remove the edge (u, o) from $\mathcal{I}_G$;
22 **if** $|N(u)| < T$ **then**
23 $N(u) \leftarrow N(u) \cup \{y|y \in N(N(u))\}$;
24 $N(u) \leftarrow$ The T' closest points in $N(u)$ to u;

conduct a range search in LSH-APG with the search radius d_m from o's closest neighbor in $\mathcal{I}_G$ (Lines 9-15). Once a point u is found, we check whether u is in the $RN(o)$. If so, (u, o) is discarded and the in-degree of o is decreased by one. Moreover, we increase the number of u's neighbors to T' by finding points in neighbors of u's neighbors once the degree of u is less than T. By this manner, we maintain the degree of each vertex in the range $[T, T']$ and reduce the bad influence of the deletion on the graph quality.

However, d_m could be very large in some cases and it is costly to check all the points whose distance to q is within d_m. Moreover, $\mathcal{I}_G$ is a directed graph and some points in $RN(o)$ is unreachable from o. Hence, we first set a maximum search cost C_{Dm} to control the cost of the range search during the deletion. The larger C_{Dm}, the more likely it is to find an in-edge, but it incurs a higher cost. Then, for an in-edge of o not found in the range search, we leave it to the deletion procedure in the following queries. If it is found during an ANN query later, we discard it and decrease the in-degree of o by one. Once the in-degree of o becomes 0, we discard all the out-edges of o and o itself. Finally, to avoid some in-edges not being found for a long time and thus wasting the space, we also traverse the graph and discard all the edges to be deleted when their amount reaches 10% of the total number of edges in $\mathcal{I}_G$.

Table 2: Summary of Datasets

Datasets	Cardinality	Dim.	LID	Size (GB)
MNIST*	60,000	784	12.7	0.184
Deep1M*	1,000,000	256	26.0	1.00
Gauss10M†	10,000,000	32	26.3	1.19
Rand10M†	10,000,000	32	23.9	1.19
Gist1M*	1,000,000	960	36.2	3.58
SIFT10M*	10,000,000	128	22.0	4.77
SIFT100M*	100,000,000	128	23.7	47.7
Tiny80M*	79,302,017	384	44.6	113

*Real-world Datasets; †Synthetic Datasets

The deletion cost of LSH-APG includes the cost of finding o's in-edges, adding new neighbors for some points and C_{Dm}. Due to the property of the NN-graph, *i.e.,* neighbors are more likely to be neighbors of each other [14], $C_{Dm} = C_Q$ can ensure that the most of in-edges are found. To add new neighbors for a point u, it is sufficient to find points in neighbors of e's neighbors rather than conduct a query for u. C_{Dm} is bounded by C_Q and thus we have the following conclusion as follows:

Theorem 4. *The deletion cost of LSH-APG is independent on n.*

7 EXPERIMENTAL STUDY

In this section, we conduct extensive experiments on real-world and synthetic datasets to provide a comprehension analysis on LSH-APG. We implement LSH-APG[1] and the competitors in C++ compiled with g++ using -Ofast optimization and openMP for parallelism. All experiments are run on a Ubuntu server with 4 Intel(R) Xeon(R) Gold 6218 CPUs (160 threads) and 1.5 TB RAM.

7.1 Experimental Settings

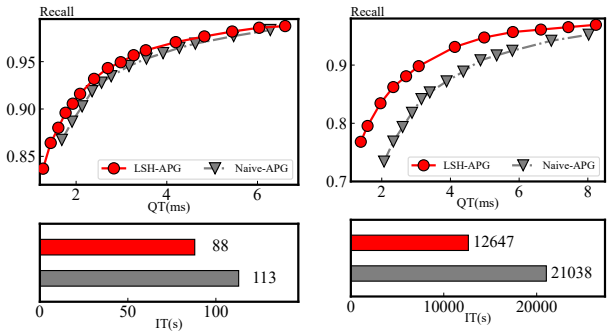
Datasets. We employ 8 datasets varying in cardinality and dimensionality, whose information is summarized in Table 2 in the ascending order of their sizes. **LID** [2] in the table is the estimated local intrinsic dimensionality, as explained in Section 4.4. A larger LID implies that it is harder to find NN on the dataset. Among the datasets, Six are real-world datasets widely used in NN search methods[19, 38, 46, 47], including our competitors. The rest 2 synthetic datasets, **Rand10M** and **Gauss10M** are generated from uniform distribution $\mathbf{U}(-1, 1)$ and Gaussian distribution $\mathbf{N}(0, 1)$ on each dimensionality independently. For NN queries, we randomly select 100 points as query points and remove them from the datasets.

Competitors. To demonstrate the indexing and query performance of LSH-APG, we compare it with the best existing NN search methods, including LSH-based and graph-based methods. Among LSH-based methods, DB-LSH [46] is proven to have the lowest query complexity. Among graph-based methods, a comprehensive experimental comparison of graph-based methods [47] has shown that HNSW [38], HCNNG [39] and NSG [19] are always the best three ones in terms of the query performance. Therefore, we choose DB-LSH, HNSW, HCNNG and NSG as our competitor algorithms.

Parameter Settings. We consider the (c, k)-ANN queries with $k = 50$ for all algorithms in the default settings. Following the settings in the paper or source code of our competitors, we adopt the fixed parameters for each algorithm. For LSH-APG, $K = 16, L = 2, T = 24, T' = 2T, p_\tau = 0.95$. For HNSW, $M = 48, ef = 80$. For NSG, $L = 40, R = 50, C = 500$. For HCNNG, the maximum size of the cluster is 500 and the number executions of hierarchical clustering procedures is 10. For DB-LSH, $c = 1.5, K = 12, L = 5$.

[1]https://github.com/Jacyhust/LSH-APG

[2]There is no unified method to compute the LID and we estimate it based on the Def. 1 in [1] with x being the average distance between the query points and their 50-th NN.



(a) DEEP1M (b) SIFT100M

Figure 3: Comparison on LSH-APG and Naive-G

Evaluation Metric. We compare the algorithms from four aspects: indexing quality, indexing efficiency, query quality and query efficiency. We report the index size (**IS**) of all algorithms and evaluate the quality of them by using the normalized maximum common subgraph (NMCS). Maximum common subgraph (MCS) is used to measure the similarity of two graphs [16]. Here, we adopt **NMCS**, a derived definition from MCS, to compute the similarity between a graph index G for the ANN query and the exact NN graph. Let $G = (V, E)$ be an APG and $G_E = (V, E')$ be the exact NN graph of V that satisfies: 1) For any a point $v \in V$, $|G(v)| = |G_E(v)|$ where $G(v)$ is the neighbors of v in G; 2) Let $k' = |G_E(v)|$. $G_E(v)$ is the k'-NN result of v in $V - \{v\}$. Then

$$NMCS = \frac{\sum_{v \in V} |G(e) \cap G_E(e)|}{\sum_{v \in V} |G(e)|} \tag{5}$$

Since the exact NN graph is nearly impossible to compute, we randomly choose 200 vertexes in V to estimate the NMCS. We evaluate the query quality via recall. Given a query point q, for a (c, k)-ANN query, assume that the algorithm return the set $R = \{o_1, \ldots, o_k\}$ and the exact kNN of q is $R^* = \{o_1^*, \ldots, o_k^*\}$, then recall are defined as follows [46],

$$Recall = \frac{|R \cap R^*|}{k}. \tag{6}$$

We use the running time to evaluate the indexing and query efficiency. The indexes are built with openMP parallelizing and the query are conducted serially. We adopt the total Indexing Time (**IT**) and query time (**QT**) to report the results..

7.2 Self Evaluation

In this subsection, we demonstrate the effectiveness of the LSH framework on LSH-APG and analysis the effect of p_τ.

7.2.1 ***Evaluation of the LSH framework.*** To demonstrate the effectiveness of the LSH framework, we compare Naive-APG with LSH-APG on DEEP1M and SIFT100M datasets, as shown in Figure 3. For the sake of brevity, the results of other datasets can be found in the technical report. To evaluate the query performance, we

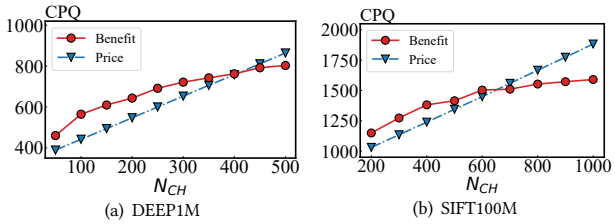


Figure 4: The benefit and price of LSH framework

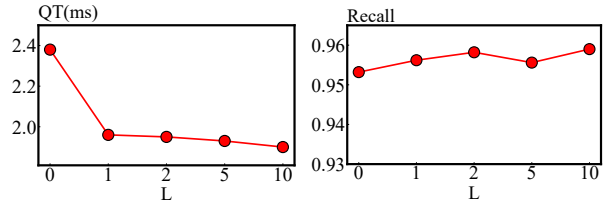


Figure 5: Performance of LSH-APG when varying L

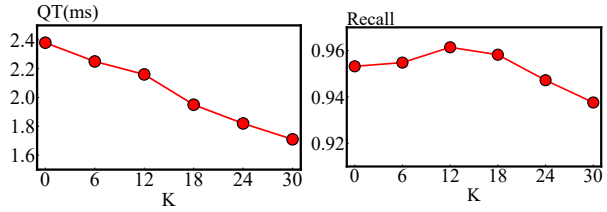


Figure 6: Performance of LSH-APG when varying K

plot the Recall-Time curves and observe that LSH-APG reduces the QT by approximately 20% on DEEP1M and 50% on SIFT100M, thus indicating an improvement in query processing. As for the indexing performance, LSH-APG and Naive-APG have almost the same graph index and the LSH framework only affects the indexing cost. Hence, we present their indexing times and find that LSH-APG reduces the indexing time by 20% on DEEP1M and 45% on SIFT100M. This highlights the importance of the LSH framework in the indexing phase. In addition, we compare the extra cost and the benefit of LSH framework by varying the number of points checked in LSH indexes. The results are shown in Figure 4 where CPQ is the average computation cost per query and N_{CH} is the number of points checked in LSH indexes. As shown in the figures, with the number of points checked in LSH indexes increasing, the extra cost increases rapidly but the benefit increases more gradually. Thus, it is unnecessary to spend much cost on LSH index. However, the benefit from LSH is greater than the extra cost in a large range, which demonstrates that LSH framework can facilitate the query processing.

7.2.2 ***Parameter study on L and K.*** To further study the LSH framework, we evaluate the impact of L and K on the query performance of LSH-APG by setting L in the range $\{0, 1, 2, 5, 10\}$ and K in the range $\{0, 6, 12, 18, 24, 30\}$, respectively. We only show results for DEEP1M for brevity, but the results for other datasets can be found in the technical report. As shown in Figure 5, the QT drops significantly with L increasing from 0 to 2 but gradually with L increasing from 2 to 10. The recall remains rather stable in the whole range. As a larger L implies a higher space consumption, we choose $L = 2$ as the default value. As shown in Figure 6, the QT keeps decreasing with K. But the recall increases and then decreases with K. This is because when L is fixed, a too large K will reduce the candidates found in the LSH indexes. Considering these two factors, we choose $K = 18$ as the default value.

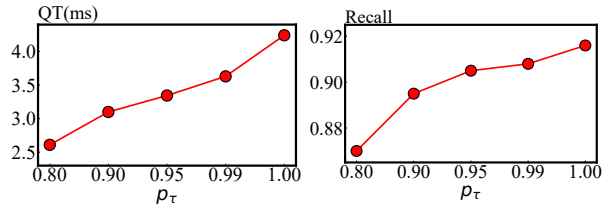


Figure 7: Performance of LSH-APG when varying p_τ

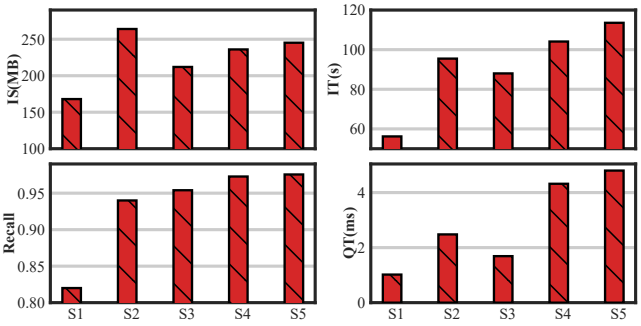


Figure 8: Performance on DEEP1M when Varying T and T'

7.2.3 ***Parameter study on p_τ.*** We investigate the impact of p_τ on query performance by varying p_τ in the range of $\{0.8, 0.9, 0.95, 1.0\}$. For the sake of brevity, we present the results only for the SIFT100M dataset, while the results for other datasets can be found in the technical report. p_τ determines the pruning threshold during the search, and a smaller p_τ increases the likelihood of filtering out neighbors. As depicted in Figure 7, both recall and QT improve with an increase in p_τ, indicating that the pruning condition in LSH-APG can reduce query cost, albeit at the cost of slightly degrading query quality since it filters out exact results to some extent. Considering both query efficiency and quality, we set $p_\tau = 0.9$.

7.2.4 ***Parameter study on T and T'.*** In this experiment, we evaluate the impact of the parameters T and T' on the performance of LSH-APG by setting (T, T') to $S1 = (24, 24)$, $S2 = (48, 48)$, $S3 = (24, 48)$, $S4 = (24, 72)$, and $S5 = (24, 96)$, respectively. We only show the results for DEEP1M for brevity, but the results for other datasets can be found in the technical report. As shown in Figure 8, when comparing $S2$ and $S3$, we observe that the setting of $T' = 2T$ results in higher recall with nearly the same query time (QT). However, the setting of $T = T' = 48$ results in higher indexing time (IT) than the setting of $T = 24, T' = 48$, but with a worse query performance. When fixing T as 24, increasing T' results in a higher IT, QT, and recall (as seen from $S1, S3, S4, S5$). The IT and QT increase almost linearly with T', while recall remains almost constant when $T' = 2T$. Therefore, $T' = 2T$ is a good choice.

7.3 Evaluation of Indexing Performance

We evaluated the index quality and efficiency of all algorithms with their default settings. Our results are depicted in Figure 9. NSG failed to build an index for the Tiny80M dataset, so we omitted its results for this dataset, including its query performance. We

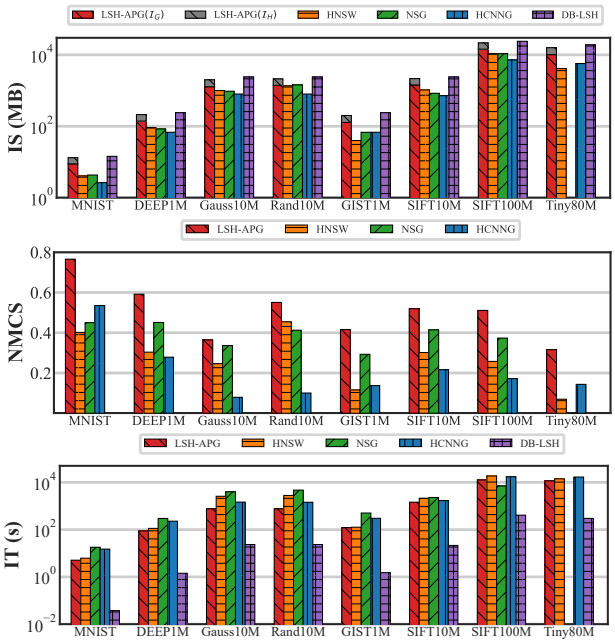


Figure 9: Indexing Performance in All Datasets

have the following observations when comparing each evaluation metric: (1) When comparing the index size of the different algorithms, we found that LSH-APG has the largest index size among the graph-based algorithms, smaller only than DB-LSH. This is because LSH-APG adopts a simple neighbor selection strategy and does not discard similar edges. The index size of $\mathcal{I}_H$ is about 30% of that of LSH-APG, which is acceptable. HNSW has a very small index size on the Gist1M and Tiny80M datasets, but this is due to its heuristic neighbor selection, which can negatively affect the query performance. (2) LSH-APG consistently has the highest NMCS among the graph-based algorithms, indicating that it is the most similar to an exact NN graph and has the highest-quality edges, which benefits the query quality. (3) DB-LSH has the smallest IT among all the algorithms since it only needs to compute a small number of hash functions, which is much faster than finding edges for each point. Among the graph-based algorithms, LSH-APG has the smallest IT, comparable only to HNSW. NSG and HCNNG have larger ITs. This is due to LSH-APG's more efficient entry point selection and LSH-based pruning conditions during insertion, which avoid many unnecessary distance computations. Additionally, LSH-APG does not use the time-consuming heuristic selection strategy like HNSW or NSG. Building an NSG requires a large degree of time, both in constructing an approximation kNN graph and in selecting heuristic neighbors. Additionally, ensuring connectivity also requires a lot of time, especially on datasets like Gauss10M and Rand10M, accounting for 10-20% of the total indexing time. (4) Comparing the NMCS and IT among different datasets, we found that all algorithms follow similar trends. For example, LSH-APG has a smaller IT and higher NMCS on DEEP1M compared to GIST1M. Similarly, HNSW, NSG, and HCNNG all have smaller ITs and higher NMCSs on DEEP1M compared to GIST1M. The IT and NMCS of each algorithm also vary among the datasets. For LSH-APG, the IT increases gradually with the data cardinality when comparing SIFT10M and SIFT100M, which aligns with our cost model.

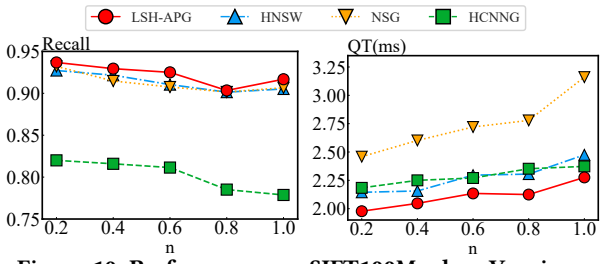


Figure 10: Performance on SIFT100M when Varying n

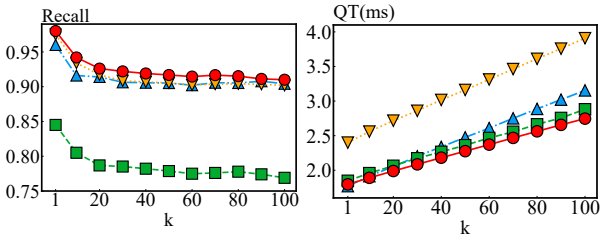


Figure 11: Performance on SIFT100M when Varying k

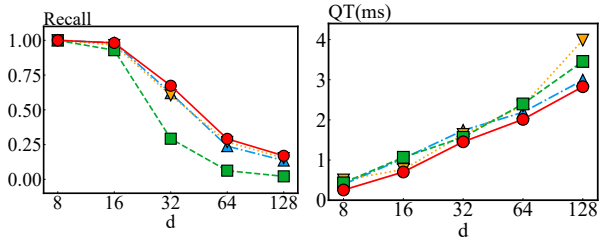


Figure 12: Performance on Rand10M when Varying d

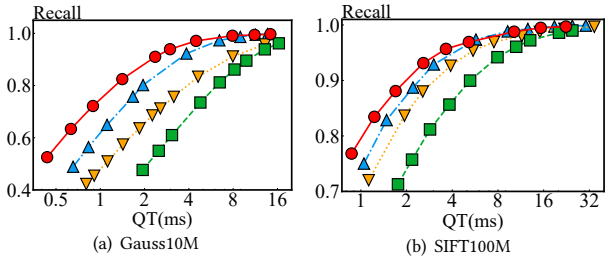


Figure 13: Recall-QT Curves

7.4 Evaluation of Query Performance

In these experiments, we evaluate the query performance of all algorithms by considering different factors such as data cardinality n and dimensionality d, varying k, and increasing the number of checked points. The results of DB-LSH are not included in the rest of the experiments as it requires much higher query costs to achieve similar recall compared to graph-based methods. For instance, the QT of DB-LSH to reach a recall of 0.95 is approximately 7.96 seconds, which is 500 times greater than that of the graph-based methods.

7.4.1 ***Effect of*** *n.* By randomly selecting a portion of the original dataset, we evaluate the query performance of all algorithms in their default settings. We present the results on the SIFT100M dataset in Figures 10 and results of the rest datasets are shown in the technical report [42]. The cardinality n ranges from $\{0.2N_0, 0.4N_0, \ldots, N_0\}$,

where N_0 is the original dataset's cardinality. As n increases, each algorithm's QT increases and recall decreases. However, the increase in QT for LSH-APG is relatively small, supporting our conclusion that LSH-APG's query cost is less impacted by the data cardinality.

7.4.2 ***Effect of*** k. We compare the query performance of all algorithms as k varies in the range $\{1, 10, 20, \ldots, 100\}$. Since , we only present the results on the SIFT100M dataset in Figures 11. The figures show that the QT of each algorithm increases nearly linearly with k, but LSH-APG has the smallest slope. Furthermore, LSH-APG consistently achieves the smallest QT and the highest recall, indicating it is the best-performing algorithm in terms of query processing among the competitors. NSG and HNSW show surprisingly similar recall values, and their QT-k curves have approximately the same slope, further highlighting their similarities.

7.4.3 ***Effect of*** d. In the default settings, we compare the query performance of all algorithms as the dimensionality d of the datasets varies. We only compare the algorithms on two synthetic datasets, Rand10M and Gauss10M, with the dimensions ranging from 8 to 128. The results of Gauss10M are displayed in Figure 12 and the results of Gauss10M are shown in the technical report [42]. As the figures indicate, the QT of each algorithm increases with d. The increasing tendencies of LSH-APG, HNSW, and HCNNG are sublinear, while NSG's are nearly linear, suggesting that NSG is more susceptible to the effects of dimensionality. To our surprise, the recall of each algorithm drops very rapidly with increasing d. When d is 8 or 16, all algorithms reach a recall of almost 1.0, but when d increases to 32, the recall drops to around 0.6 for Gauss10M and 0.75 for Rand10M. When d reaches 64 or even 128, the recall drops to less than 0.3. This result shows that the impact of dimensionality on recall is much greater than the impact of cardinality. This phenomenon can be attributed to the "curse of dimensionality," where as the dimensionality increases, the distance between any two points becomes nearly identical, making it difficult to differentiate NNs from other points during a query. This "curse of dimensionality" becomes particularly pronounced on random datasets compared to real-world datasets, even when d is only 64 or 128, due to their higher LID (local intrinsic dimensionality) with the same dimensionality.

7.4.4 ***Recall-QT Curve***. The accuracy of an ANN method can be improved by increasing the number of checked points, but this also reduces query efficiency. In these experiments, we analyze the trade-off between recall and query time (QT) through the Recall-QT curve. The algorithm with the smaller QT for a given target recall is considered to have better query performance. The results for four datasets are shown in Figure 13. From the figures, we make the following observations: (1) As QT increases, all algorithms achieve higher recall, which aligns with the principle of ANN methods, where accuracy is traded for efficiency. Additionally, the required QT to reach a given recall *Rec* increases nearly exponentially with *Rec*, highlighting the difficulty and time-consuming nature of finding exact nearest neighbors. (2) Among all algorithms, LSH-APG requires the smallest QT to reach the same recall, indicating the best trade-off between query quality and efficiency. This is due to the better entry point selection in LSH indexes and the effective pruning condition used to filter out neighbors. (3) NSG and HNSW have similar performance, as evidenced by the nearly identical curves on

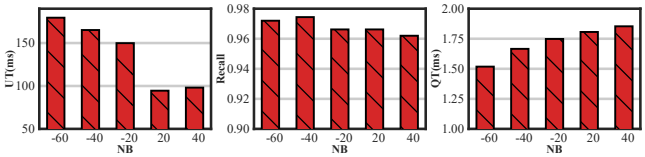


Figure 14: Performance on DEEP1M when Updating

Rand10M. This is because they both use the same heuristic selection strategy, as proven in [47]. However, HNSW generally performs better. (4) HCNNG always has the worst query performance. On Gauss10M and Rand10M, to reach the same recall, HCNNG requires nearly 4 times the QT of LSH-APG. This can be explained by the cluster-based approach used in HCNNG to construct subgraphs, which may not be effective when data in high-dimensional space is not well-clustered.

7.5 Evaluation of Updating

We evaluate the update performance of LSH-APG as described in [26] by conducting batch insertions or deletions. Given the initial graph index of LSH-APG, $\mathcal{I}_G = (V_0, E_0)$, we represent a batch insertion (or deletion) as a $Y\%$ update, where $Y = \frac{|V|-|V_0|}{|V_0|}$ and $|V|$ is the number of points in the graph after the update. In this set of experiments, we examine the impact of Y on the DEEP1M dataset, where $V_0 = 600K$, by varying Y in $\{-60, -40, -20, 20, 40\}$ ($Y < 0$ indicates a deletion operation). The results of query performance and update time per point (UT) are shown in Figure 14. Regarding the updating time, we observe that insertion time is lower than deletion time and that 40% insertion has a slightly higher time compared to 20% insertion. The increase in deletion time with $|Y|$ is due to the decrease in the total number of points in the graph and the increase in the probability that a vertex connects to the points being deleted, resulting in more time spent adding new edges. In terms of query performance, we find that recall remains stable and query time (QT) slightly increases with Y, as the number of points increases. This suggests that update operations do not negatively impact query performance.

8 CONCLUSION

In this paper, we have proposed a novel approach, called LSH-APG, to efficiently build an APG and facilitate ANN query processing in high-dimensional spaces with quality guarantees. LSH-APG handles the high construction cost issue in the graph-based methods by designing an efficient and accurate LSH-based query strategy to consecutively insert data points in the APG. A high-quality entry point selection technique and LSH-based pruning condition have been developed to reduce the number of points to be checked in the search. The expected query cost has been proven to be less affected by dataset cardinality, allowing us to simultaneously reduce the query processing time and the indexing time. Therefore, LSH-APG is well positioned to deal with large-scale datasets. In addition, It is proven that LSH-APG can be maintained incrementally in a low cost as the dataset evolves. A thorough range of experiments showed that LSH-APG can reduce the construction cost by an average of 40% compared to the best competitors, HNSW, while still maintaining the best query performance.

REFERENCES

[1] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E. Houle, Ken-ichi Kawarabayashi, and Michael Nett. 2015. Estimating Local Intrinsic Dimensionality. In *KDD*. 29–38.

[2] Alexandr Andoni and Piotr Indyk. 2016. LSH Algorithm and Implementation (E2LSH). https://www.mit.edu/~andoni/LSH

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

[4] Mahendra Awale and Jean-Louis Reymond. 2019. Polypharmacology Browser PPB2: Target Prediction Combining Nearest Neighbors with Machine Learning. *J. Chem. Inf. Model.* 59, 1 (2019), 10–17.

[5] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD Conference*. ACM Press, 322–331.

[6] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (1975), 509–517.

[7] Christian Böhm. 2000. A cost model for query processing in high dimensional data spaces. *ACM Trans. Database Syst.* 25, 2 (2000), 129–178.

[8] Brankica Bratic, Michael E. Houle, Vladimir Kurbalija, Vincent Oria, and Milos Radovanovic. 2018. NN-Descent on High-Dimensional Data. In *WIMS*. 20:1–20:8.

[9] Paolo Ciaccia, Marco Patella, and Pavel Zezula. 1997. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB*. Morgan Kaufmann, 426–435.

[10] Jose A Costa, Abhishek Girotra, and AO Hero. 2005. Estimating local intrinsic dimension with k-nearest neighbor graphs. In *IEEE/SP 13th Workshop on Statistical Signal Processing, 2005.* IEEE, 417–422.

[11] H. K. Dai and Hung-Chi Su. 2003. On the Locality Properties of Space-Filling Curves. In *ISAAC (Lecture Notes in Computer Science)*, Vol. 2906. 385–394.

[12] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*. 253–262.

[13] Intrinsic dimension. 2023. Available at:. https://en.wikipedia.org/wiki/Intrinsic_dimension#Local_Intrinsic_Dimensionality

[14] Wei Dong, Moses Charikar, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*. 577–586.

[15] Carlos Eiras-Franco, David Martínez-Rego, Leslie Kanthan, César Piñeiro, Antonio Bahamonde, Bertha Guijarro-Berdiñas, and Amparo Alonso-Betanzos. 2020. Fast Distributed *k*NN Graph Construction Using Auto-tuned Locality-sensitive Hashing. *ACM Trans. Intell. Syst. Technol.* 11, 6 (2020), 71:1–71:18.

[16] Mirtha-Lina Fernández and Gabriel Valiente. 2001. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognit. Lett.* 22, 6/7 (2001), 753–758.

[17] Cong Fu and Deng Cai. 2016. EFANNA : An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph. *CoRR* abs/1609.07228 (2016).

[18] Cong Fu, Changxu Wang, and Deng Cai. 2021. High Dimensional Similarity Search with Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), 1–1. https://doi.org/10.1109/TPAMI.2021.3067706

[19] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *PVLDB* 12, 5 (2019), 461–474.

[20] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD*. 541–552.

[21] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 4 (2014), 744–755.

[22] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *VLDB*. 518–529.

[23] Robert M. Gray and David L. Neuhoff. 1998. Quantization. *IEEE Trans. Inf. Theory* 44, 6 (1998), 2325–2383.

[24] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD Conference*. 47–57.

[25] Ben Harwood and Tom Drummond. 2016. FANNG: Fast Approximate Nearest Neighbour Graphs. In *CVPR*. IEEE Computer Society, 5713–5722.

[26] Kai Huang, Huey-Eng Chua, Sourav S. Bhowmick, Byron Choi, and Shuigeng Zhou. 2021. MIDAS: Towards Efficient and Effective Maintenance of Canned Patterns in Visual Graph Query Interfaces. In *SIGMOD Conference*. 764–776.

[27] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-Aware Locality-Sensitive Hashing for Approximate Nearest Neighbor Search. *PVLDB* 9, 1 (2015), 1–12.

[28] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *STOC*. 604–613.

[29] Jerzy W. Jaromczyk and Miroslaw Kowaluk. 1991. Constructing the relative neighborhood graph in 3-dimensional Euclidean space. *Discret. Appl. Math.* 31, 2 (1991), 181–191.

[30] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011), 117–128.

[31] Rolf Klein. 2016. Voronoi Diagrams and Delaunay Triangulations. In *Encyclopedia of Algorithms*. 2340–2344.

[32] 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 Trans. Knowl. Data Eng.* 32, 8 (2020), 1475–1488.

[33] Peng-Cheng Lin and Wan-Lei Zhao. 2019. On the Merge of k-NN Graph. *CoRR* abs/1908.00814 (2019).

[34] Yingfan Liu, Jiangtao Cui, Zi Huang, Hui Li, and Heng Tao Shen. 2014. SK-LSH: An Efficient Index Structure for Approximate Nearest Neighbor Search. *PVLDB* 7, 9 (2014), 745–756.

[35] Kejing Lu and Mineichi Kudo. 2020. R2LSH: A Nearest Neighbor Search Scheme Based on Two-dimensional Projected Spaces. In *ICDE*. IEEE, 1045–1056.

[36] Kejing Lu, Hongya Wang, Wei Wang, and Mineichi Kudo. 2020. VHP: Approximate Nearest Neighbor Search via Virtual Hypersphere Partitioning. *Proc. VLDB Endow.* 13, 9 (2020), 1443–1455.

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

[38] 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.

[39] 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 (2019).

[40] Beng Chin Ooi. 1987. Spatial kd-Tree: A Data Structure for Geographic Database. In *BTW (Informatik-Fachberichte)*, Vol. 136. Springer, 247–258.

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

[42] Technical Report. 2023. Available at:. https://github.com/Jacyhust/LSH-APG/tree/main/Report

[43] Bo Tang and Haibo He. 2015. ENN: Extended Nearest Neighbor Method for Pattern Recognition [Research Frontier]. *IEEE Comput. Intell. Mag.* 10, 3 (2015), 52–60.

[44] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. 2009. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*. 563–576.

[45] Yuan Tian, David Lo, and Chengnian Sun. 2012. Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction. In *WCRE*. IEEE Computer Society, 215–224.

[46] Yao Tian, Xi Zhao, and Xiaofang Zhou. 2022. DB-LSH: Locality-Sensitive Hashing with Query-based Dynamic Bucketing. In *ICDE*. 2250–2262.

[47] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *PVLDB* 14, 11 (2021), 1964–1978.

[48] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB*. Morgan Kaufmann, 194–205.

[49] Peter N. Yianilos. 1993. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *SODA*. ACM/SIAM, 311–321.

[50] Wan-Lei Zhao. 2018. k-NN Graph Construction: a Generic Online Approach. *CoRR* abs/1804.03032 (2018).

[51] Bolong Zheng, Xi Zhao, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, and Christian S. Jensen. 2020. PM-LSH: A Fast and Accurate LSH Framework for High-Dimensional Approximate NN Search. *Proc. VLDB Endow.* 13, 5 (2020), 643–655.