LSH-APG: Building Approximate Proximity Graphs Through Locality-Sensitive Hashing

ABSTRACT

Approximate nearest neighbor (ANN) search in high-dimensional spaces is a fundamental but computationally expensive problem in the database. State-of-the-art solutions can be categorized into LSH-based methods and graph-based methods. LSH-based methods are costly to reach a high query quality due to the hash-boundary issue and graph-based methods always achieve the better query performance by greedy expansion from an arbitrary entry point in an approximate proximity graph (APG). However, these APGs suffer from a huge construction cost since it requires finding proper neighbors for each vertex. To overcome this issue, in this paper we propose a novel approach named LSH-APG to build the APG from lightweight LSH indexes. By consecutively inserting the points into the index based on its nearest neighbors found in the APG, it handles the high construction cost issue with an efficient but accurate LSH-based query strategy on the APG. First, it adopts LSH indexes to quickly retrieve some neighbors as the entry points of the search on the APG, and then further improves the query quality via the graph. Second, an accurate and scalable LSH-based pruning strategy is proposed to filter out those neighbors which are far away from the query point. Moreover, an efficient update strategy is designed to maintain our index as the dataset evolves. The theoretical analysis shows that the query cost of LSH-APG is affected little by the data cardinality n, and thus can be built in $O(n)$ cost. Extensive experiments on real-world datasets and synthetic datasets demonstrates that LSH-APG achieve a much lower construction cost than state-of-the-art graph-based methods and also a better query performance.

KEYWORDS

Graph based method, nearest neighbor search, locality-sensitive hashing

ACM Reference Format:

. 2018. LSH-APG: Building Approximate Proximity Graphs Through Locality-Sensitive Hashing. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/1122445.1122456

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 [34] and data mining [36]. It is well known that finding the exact NN in large high-dimensional datasets can be very time-consuming [20]. A more efficient and potentially more practical alternative is to perform an approximate nearest neighbor (ANN) search. For example, the c-approximate nearest neighbor (c-ANN) search problem finds a point whose distance to the query point q is at most cr, where r is the distance from q to its exact NN and c is a user-given approximation ratio [8, 16].

A large number of solutions have been proposed to support efficient ANN search in high-dimensional spaces. These methods can broadly be divided into two categories: locality-sensitive hashing (LSH)-based methods [2, 15, 16, 28, 35, 37] and graph-based methods [14, 17, 17, 31]. 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 c-ANN result with a guaranteed high probability by only checking the points around the query point in low-dimensional spaces[2, 16]. 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 corespondent points are sufficiently close to each other in the original space. The search to find the ANN of q can begin from an arbitrary point with a greedy expansion approach in an APG until q is in reach. A popular query strategy used in the graph-based methods is as Algorithm 1.

The graph-based methods have been extensively studied due to their higher query processing efficiency than the LSH-based methods in practice. Compared with LSH-based methods that can suffer from the problem of hash-boundary issues (i.e., some points which are close in the original space may be mapped into different hash buckets thus cannot be identified by LSH as neighbors) [15, 16, 37], the graph-based methods can achieve much higher accuracy [14, 38]. 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, 25], as such an approach will require finding proper neighbors for each and every vertex. In addition, graph-based methods consider only static datasets. It remains to be an open problem to efficiently maintain APG (i.e., without reconstructing the whole or a large part of the graph) as the dataset evolves.

Motivated by the above observations, in this paper, we propose LSH-APG, a novel approach to build APGs from lightweight LSH indexes to facilitate efficient ANN query processing. By exploiting the properties of LSH functions and analyzing the problem in the 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 of the search in a proximity graph, and then utilizes graph-based techniques

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00

https://doi.org/10.1145/1122445.1122456

to further improve the query quality. To boost query processing efficiency, an accurate and scalable pruning strategy is proposed to filter out those 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. In particular, 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 improvement of the query efficiency by the LSH framework but also allows for a formal analysis to algorithmic correctness and complexity.

The main contributions in this paper include:

- Based on a comprehensive analysis of the state-of-the-art LSH-based methods and graph-based methods, a new solution named LSH-APG is developed by adopting a well-designed LSH framework to accelerate indexing and query processing using 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 and query quality.
- The LSH framework is extended to other graph-based methods to demonstrate its effectiveness. In addition, an efficient update strategy is designed for efficient and effective maintenance of index structures as the database evolves.
- We provide a theoretical analysis of LSH-APG and demonstrate that LSH-APG can be built in an expected computational cost of $O(ndC_Q)$ and answer an ANN query with a constant success probability in expected $O(dC_Q)$ computational cost, where n, d and C_Q are data cardinality, data dimensionality and expected number of accessed points during the search, respectively. The effectiveness of the LSH framework is also theoretically proven.
- Extensive experiments show that LSH-APG can greatly reduce indexing cost and achieve the best trade-off between query processing efficiency and query accuracy over the existing methods for ANN queries.

2 RELATED WORK

We discuss the mainstream ANNS methods in high-dimensional spaces, including LSH-based methods and graph-based methods.

2.1 LSH-based Methods

LSH is originally proposed in [8, 16, 20]. In these methods, LSH maps 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. [35] and [15] 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 [19, 28, 29] and metric based strategy [27, 41]. 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 [37].

Algorithm 1: Greedy Search in the Graph($q, \mathcal{G}$)

Input: A query point q, the graph index $\mathcal{G}$
Output: k nearest points to q

$e_p \leftarrow$ an random point in $\mathcal{G}$;
$R \leftarrow EP$s; //store the k best results
$V \leftarrow \{e_p\}$;
$E \leftarrow \{e_p\}$;
while ***TRUE*** **do**
    $e_p \leftarrow$ pop the nearest element in E to q;
    $o_k \leftarrow$ the found k nearest points in R;
    for *each* $o \in e_p.neighbors$ **do**
        if $o \notin V$ **then**
            $V \leftarrow V \cup o$;
            if $dist(q, o_k) > dist(q, o)$ **then**
                Insert o into E and R;
    if $dist(q, o_k) > dist(q, e_p)$ **then**
        return k nearest points in R;

2.2 Graph-based Methods

Graph-based methods use the proximity graph to facilitate ANN search and have shown better query performance compared to LSH-based methods, in terms of both accuracy and efficiency recently [31, 32, 38]. However, the construction cost of exact proximity graph, *e.g.,* Delaunay graph (DG) [23], relative neighborhood graph (RNG) [21], minimum spanning tree (MST) [32] and kNN graph (KNNG) [9], 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 accuracy. Dong et al.[9] 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 [12], NSG [14] and others [26, 40] and some derivatives are developed [5]. However, it needs iterating about 10 times to find the high-quality neighbors, which is still time-consuming. NSW [30] 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 [31] adopts the same strategy as NSW but limits

Table 1: List of Key Notations.

Notation	Description
$\mathbb{R}^d$	d-dimensional Euclidean space
$\mathcal{D}$	The dataset
n	The cardinality of dataset
d_i	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

the maximum degree for each point to alleviate the hubness issue. HCNNG [32] and VRLSH [10] 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 [14, 17, 25, 31]. In this manner, the similar edges will be cut off to reduce the average degree of the graph based on some occlusion rules. NSG [14] 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 [25] and NSSG [13] 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, but all of them bring much larger construction costs due to the extra computation cost for occlusion rules.

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 the 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 $o_1, o_2 \in \mathcal{D}$.

Definition 1 ((c, k)-ANN Query [37]). *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 $\|q, o_i\| \le c \cdot \|q, o_i^*\|$.*

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) [37, 41]). *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 $\|o_1, o_2\| \le r$, $\Pr[h(o_1) = h(o_2)] \ge p_1$;
(2) If $\|o_1, o_2\| > cr$, $\Pr[h(o_1) = h(o_2)] \le p_2$,

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 [19]:

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

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 $\frac{\|P(o_1), P(o_2)\|^2}{\|o_1, o_2\|^2} \sim \chi^2(m)$.*

Proof. The proof of this lemma can be found in [41]. □

Another commonly used LSH family in the Euclidean space is defined as follows [8]:

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

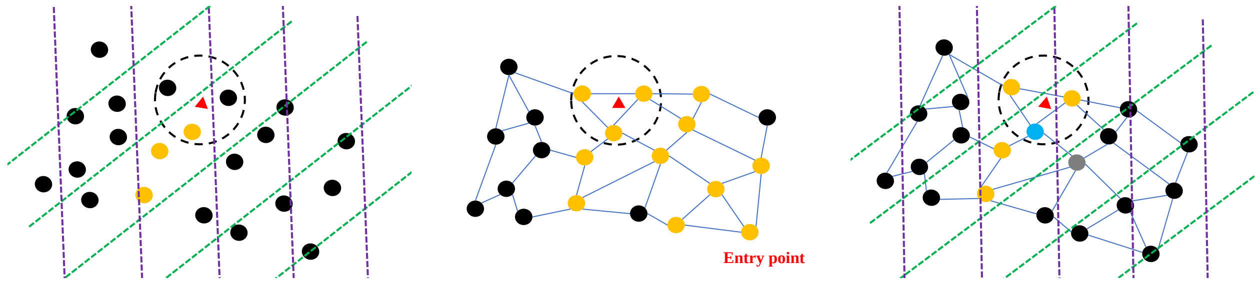
where w is a pre-defined integer and b is a real number chosen uniformly from $[0, w)$. To distinguish these two LSH families, we call h^* as the projected function and h as the hash function.

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 vertexes if the corespondent points are sufficiently close to each other in the original space. There are mainly three strategies to build the proximity graph [38].

- **Cluster & Merge [32].** The dataset $\mathcal{D}$ is clustered into several small groups and the exact proximity graph is built in each group. Then, we obtain the approximate proximity graph by merging the all subgraphs.
- **Iteration [9, 14].** The APG is built from an initial random graph. For each vertex, we iteratively update its neighbors according to the local information. The final APG is obtained when the iteration converges.
- **Consecutive Insertion [30, 31].** In this manner, the graph is built by inserting the point one by one, like that in R-Tree.

The number of edges directly affects the query performance[38]. Intuitively, the more the out-edges of a vertex, the more candidates and computations need to be performed, and the slower the query



(a) Search in the LSH Method

(b) Search in the Graph

(c) Search in LSH-APG

Figure 1: Difference in the LSH-based method, graph-based method and LSH-APG. 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. (a) is the LSH-based method. The two groups of parallel lines (purple lines and green lines) are two hash functions and they divide the space into several parallelogram buckets. When conducting the ANN search, we check the points in the bucket where the query point falls. In this example, we find a correct ANN result with cost 3 but fail to find the exact NN result. (b) is an APG. To simplify the model, we assume the APG is undirected. Each data point in graph has 2-4 neighbors. The query begins from the bottom right point and approaches to the correct results via greedy search in the APG. All 3 correct results are found with cost 12. (c) is LSH-APG. We select the blue point, the closest point to q in the bucket where q falls, as the entry point, and return all 3 correct results with a much lower cost 5.

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 and how LSH-APG improves them.

LSH 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. To answer the ANN query, we only check the points in the bucket where the query point falls, *i.e.,* the 3 brown points in Figure 1(a). Both the LSH family in Equation 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 structures. As shown in the 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 greedy search in the graph. The orange points are the points accessed during the search for the 3NN 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 500 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. 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, but it depends on the specific 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 [31], 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.

Our solution. From the above analysis, we can see that LSH indexes are fast to build but difficult to achieve the high recall. On the contrary, graph-based methods perform well on the query processing but suffer from the high construction cost. To address this dilemma, we propose LSH-APG. LSH-APG follows the consecutive insertion strategy and simple selection strategy. An LSH framework is used to accelerate the construction of the APG. First, we adopt 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. As shown in Figure 1(c), we select the closest point to the query point q in the bucket where the q falls as the entry point *i.e.,* the red point, and thus the number of hops is reduced to 2, half of that in Figure 1(b). Then, we design an 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 the red point. With the help of the LSH framework, LSH-APG can be built much faster than the competitors with maintaining the same query performance. We demonstrate the effectiveness of the LSH framework via the theoretical analysis and experimental results.

4 LSH-APG STRUCTURE

LSH-APG consists of two parts, the hash indexes $\mathcal{I}_H$ and the graph index $\mathcal{I}_G$. In this section, we give a detailed description of their construction.

4.1 Building the LSH indexes

In LSH-APG, a fast LSH framework is required for quickly finding a high-quality entry point. Following the idea in [35], we build our hash indexes $\mathcal{I}_H$ as follows: first, we randomly choose K LSH functions $h_1, \dots, h_K$ as described by Eq. 2. For a point o to be inserted, we compute its K hash values $h_1(o), \dots, h_K(o)$ and consider them as a K-dimensional point $H(o) = (h_1(o), \dots, h_K(o))$. Then, we transform $H(o)$ into a one-dimensional value $z(H(o))$ via the Z-order curve [7] and store the $z(H(o))$ in a B+-Tree. 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 $H_i = \{h_{i,1}, \dots, h_{i,K}\}, i = 1, \dots, L,$ *i.e.,* there are total $L \times K$ LSH functions used.

4.2 Building the graph index

The graph index $\mathcal{I}_G = (V, E)$ is a directed graph, where $V = \mathcal{D}$ and E is constructed as follows: first, we conduct a kANN query for the incoming point o with $k = T$ in the current graph index. T is a pre-defined integer. Suppose that the found T ANNs are $o_1, \dots, o_T$. Next, we construct two edges for each point $o_i, i = 1, \dots, T$, one the edge $e = (o, o_i)$ from o to o_i, the other an edge $e = (o_i, o)$ from o_i to o. Let $N(o) = \{v | (o, v) \in E\}$ be the set of all neighbors of o and $|N(o)|$ be the degree (or out-degree) of o. Then, for each neighbor o_i of o, we check whether $|N(o_i)|$ reaches the maximum capacity T', where $T' \le T$ is a specified integer. If so, we select the closest T' points to o_i in $N(o_i)$, following the simple neighbor selection strategy. To control the quality of $\mathcal{I}_G$, we set $T' = 2T$ by default,*i.e.,* the degree of each vertex is in the range $[T, 2T]$. LSH-APG with this setting usually achieves a better performance than LSH-APG with $T' = T$. An intuition explanation is that: the real-world dataset is usually distributed unevenly. The data points in dense regions are more likely to have more neighbors, and thus have a higher degree than data points in sparse regions. Therefore, it is hard to choose a unified degree for all data points. The degrees in a range $[T, 2T]$ better fit the data distribution. Algorithm 2 describes the construction of $\mathcal{I}_H$ and $\mathcal{I}_G$ in details.

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

Input: Dataset $\mathcal{D}$ and parameter T, T'
Output: LSH-APG index $\mathcal{I}_G$ and $\mathcal{I}_H$
$\mathcal{I}_H \leftarrow \emptyset$;
$\mathcal{I}_G \leftarrow \emptyset$;
for *each point* $o \in \mathcal{D}$ **do**
  candidates $\leftarrow$ **call** kANN-Query($o, \mathcal{I}_G, \mathcal{I}_H, p_\tau = 0.95, T$);
  for *each* $e \in$ *candidates* **do**
    $\mathcal{I}_G \leftarrow \mathcal{I}_G \cup \{(o, e), (e, o)\}$;
    if $e.degree > T'$ **then**
      $o_f \leftarrow e$'s furthest neighbor in $\mathcal{I}_G$;
      $\mathcal{I}_G \leftarrow \mathcal{I}_G - \{(e, o_f)\}$;
  Insert o into the corresponding LSB-Tree in the $\mathcal{I}_H$;
return $\mathcal{I}_H$ and $\mathcal{I}_G$;

5 QUERY PROCESSING

We proceed to introduce the ANN query processing algorithm using LSH-APG. The query algorithm not only decides the efficiency and accuracy of ANN search, but also affects the index quality and indexing efficiency.

5.1 LSH Based Pruning Condition

Our LSH based pruning condition is designed based on the following inequality:

$$\|P(q), P(o)\| < \sqrt{\chi^2_{p_\tau}(m)} \cdot d_k, \tag{3}$$

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 unnecessary to continue computing the distance $\|q, 0\|$. During

Lemma 2. *Given c and $\kappa > 0$, we set $m = \kappa \log n$ and $p_\tau = F(\frac{1}{2c^2}; m)$, where $F(x; m)$ is the cumulative distribution function of χ^2 distribution with freedom m. With the LSH based pruning condition, a point whose distance to q is less than r will be accessed with at least the probability $\frac{1}{2}$ and we access at most $O(n^\alpha)$ points whose distance to q is greater than cr, where r is the current search radius and $\alpha = 1 - \frac{9\kappa(c^{-2/3}-1)^2}{4}$.*

The proof of this lemma is after Lemma 12 (Section 7). It indicates that for a close point (distance to q is less than r), the pruning condition will not filter out it with at least a constant probability, which guarantees that the query can return some required ANN results. For a far point (distance to q is greater than cr), the pruning condition will filter out it with high probability and the total number of these far points accessed is bounded by $O(n^\alpha)$. In our method, we set $m = \kappa K$ and do not explicitly give the c but directly use p_τ to control the pruning efficiency where p_τ is the probability that $\|q, o\|$ is greater than $\|q, R_k\|$ based on the χ^2 distribution.

Remark 3. *Different from the quality guarantee in the LSH methods that r is the parameter in an (r, c)-NN query [41], r in the above lemma is the current search radius. Hence, we can not compute how* *many points we do access during the query, like that in LSH methods. However, the pruning condition guarantees that for a point o, if $\|q, o\|$ is c times larger than the search radius r, it will hardly be accessed. This property could help filter many neighbors when searching in the graph, especially for the graph with the high degree.*

5.2 ANN Query in LSH-APG

The (c, k)-ANN query in LSH-APG is described in Algorithm 3. First, we find entry points in the LSH indexes. To search the close points to q in $\mathcal{I}_H$, we compute the hash values of q. Then, all L $H_i(q)$ can be obtained and we conduct ANN search for $z(H_i(q))$ in corresponding LSB-Tree. The search procedure in the LSB-Tree follows [35], so we do not describe it here again. After that, we choose the found best k results in $\mathcal{I}_H$ as entry points during the query in graph $\mathcal{I}_G$.

Lemma 3 (Theorem 1 of [35]). *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 proof of Lemma 3 is similar to the proof in many LSH methods [16, 35, 37], and thus we do not provide it again. It indicates we can find a c'^2-ANN result with a small query cost.

Remark 4. *In our method, We set $m = K = O(1)$ and $L = O(1)$, following the mainstream LSH methods [29, 37]. Although Lemma 3 indicates that we requires $L = n^{\rho}$ LSH indexes to ensure the correctness, Tao [35] said even single index is also expected to return results with high quality. Experimental shows 2-5 LSH indexes is enough to find good results and thus we regard L as $O(1)$.*

Next, we conduct kANN query in $\mathcal{I}_G$ by using the greedy strategy as that in the most graph based methods. Different from them, we use the k closest points found in $\mathcal{I}_H$ as the entry points (Line 6) and employ the LSH based pruning condition to reduce the computational cost (Line 17). When access the point o in the graph, we denote $\|q, o\|$ as the search radius. Lemma 3 shows that the reduction of the initial search radius can lower the hop numbers to terminate, and thus improve the query efficiency. Moreover, a closer entry point decreases the probability that the query terminate at a local minimal where the final search radius is still very high. Based on the above two benefits, we have the following conclusion.

Theorem 1. *The expected query cost, C_Q, and the final search radius, s, of LSH-APG only depend on the local intrinsic dimensionality of the dataset. Moreover, s is bounded by $(1 + \gamma)r_o$, where $\gamma < 1$ and r_o is the expected edge length in LSH-APG.*

The proof of this theorem is in Section 7. The theorem guarantees the query cost and query quality of LSH-APG.

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 $\mathcal{I}_H$, parameter p_τ, k
Output: k nearest points to q

1 $n \leftarrow$ the number of points in the indexes;
2 $m \leftarrow 2 \log n$;
3 $t \leftarrow \sqrt{\chi^2_{p_\tau}(m)}$;
4 Compute q's projected value $h_1^*(q), \ldots, h_{LK}^*(q)$;
5 $P(q) \leftarrow (h_1^*(q), \ldots, h_m^*(q))$;
6 $EPs \leftarrow$ k approximate nearest points to q in $\mathcal{I}_H$;//very rough search
7 $V \leftarrow$ the visited points in $\mathcal{I}_H$;
8 $R \leftarrow EPs$; //store the k best results
9 **while** $|EPs| > 0$ **do**
10     $ep \leftarrow$ pop the nearest element in EPs to q;
11     $R_k \leftarrow$ the furthest points in R to q;
12   **if** $\|ep, q\| > \|q, R_k\|$ **then**
13   **break**;
14   **for** *each* $o \in N(ep)$ **do**
15   **if** $o \notin V$ **then**
16             $V \leftarrow V \cup \{o\}$;
17   **if** $\|P(q), P(o)\| < t \cdot \|q, R_k\|$ **then**
18   Compute $\|q, o\|$;
19   update R and EPs;
20 **return** R;

as that in Algorithm 1, it is natural to extend the LSH framework on them to accelerate the query processing. In addition, we try to improve the indexing efficiency of the graph-based methods via the LSH framework by analyzing their drawbacks.

For the method with the iteration strategy, we can build an initial graph via LSH indexes, which can reduce the number of iteration to converge. For the consecutive insertion strategy, the benefit from LSH framework can be obtained by improving the query strategy via LSH. For the methods with cluster & merge strategy, such as HCNNG and VRLSH, the LSH indexes can be adopted to partition the data, which is more efficient than using the clustering-based methods since they requires more distance computations. Then, when building the subgraph in a group, we can only compute the distance of the points in the same hash buckets instead of the whole point pairs in the group since the points in the same buckets are more likely to be close. However, the clustering-based methods are hard to acquire this kind of information. Experimental results show that with the LSH framework, the construction cost of HCNNG can be reduced by $40 - 90\%$ on SIFT100M.

6 EXTENSION

In this section, we consider extending the LSH framework to other graph based methods and provide an efficient update strategy, which greatly enhance the scalability of LSH-APG.

6.1 The Extension of LSH Framework

The LSH framework provides the better entry points and allows us not to access all the neighbors, which reduce the query cost of LSH-APG. Since the most graph-based methods adopt the query strategy

6.2 Updates

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. So 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 it from $\mathcal{I}_H$. Then, some edges would be added to guarantee the query performance. it is trivial to remove a point from the LSB-Tree and the challenge comes from the deletion operations in $\mathcal{I}_G$ since the in-edges 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. To delete o, we first mark o and all the out-edges of o as the Deleting status. Then, we conduct a range search in LSH-APG with the search radius d_m and begin from o's closest neighbor in $\mathcal{I}_G$. 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 will 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 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 for o. Hence, we first set a maximum search cost C_{Dm} to control the cost of the range search during the deletion. A large C_{Dm} leads to a higher deletion ratio, which brings little bad effect on the subsequent queries. Then, for the unseen in-edges of o in the range search, we remains the rest deletion procedure in the following queries. If an unseen in-edge is found when conduct the 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 of o to be unseen for a long time and occupy the space, we will traverse the graph and discard all the edges to be deleted when their amount reaches 10% of the total number of edges in $\mathcal{I}_G$.

The deletion cost of LSH-APG is including the cost of finding o's in-edges, C_{Dm}, and of adding new neighbors for some points. Usually, $C_{Dm} = C_Q$ can ensure the most of in-edges are found due to the property of the NN-graph, *i.e.,* neighbors are more likely to be neighbors of each other [9]. To add the 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 the similar conclusion:

Lemma 4. *The expected deletion cost of LSH-APG is only depends on the local intrinsic dimensionality of the dataset.*

7 QUALITY ANALYSIS

In this section, we provide a theoretical analysis of LSH-APG. First, we give a model to analyze the query cost and the query quality of LSH-APG via C_Q and the final search radius, *i.e.,* the search radius when the query terminates. We find that C_Q and the final search radius depend little on n, which is surprising but in line with our experimental results. With the model, we compute the benefit from the LSH framework. Then, we analyze the indexing, space and query complexities of LSH-APG.

7.1 Analysis of LSH-APG

We aim to find a model to analyze the correctness and cost of LSH-APG. For the correctness, we prove the final search radius is bounded. For the cost, we prove C_Q is independent on n and can be reduced by the LSH framework. To compute C_Q and the final search radius, we consider the the hop numbers l and the final search radius s are the functions of the initial search radius, and denote them as the $l(r)$ and $s(r)$, respectively.

The models of $l(r)$ and $s(r)$. We assume that the length of the edges in $\mathcal{I}_G$ is r_o and for each vertex its neighbors are uniformly distributed around it. The hop number l depends on the initial search radius r. Given the entry point e_p with $\|q, e_p\| = r$, we denote as $l(r)$ the hop numbers of query processing starting with e_p. To solve $l(r)$, we need to know how fast the search radius decreases and where the query terminates. Here, we define $p(r)$ as the probability that the query terminates when the search radius is r and $\delta(r) = r - r'$ as the hop length, where r' is the search radius of the next hop.

Lemma 5. *Assume that $l(r) \gg \delta(r)$ and $l(r)$ is differentiable, $l(r)$, $p(r)$ and $\delta(r)$ satisfy the following equation:*

$$[1 - p(r)]\delta(r)l'(r) + p(r)l(r) = 1, \tag{4}$$

where $l'(r)$ is the derivative of $l(r)$ with respect to r.

Proof. When the query starts with search radius r, the query could terminate with the probability $p(r)$ and go to the next hop with the probability $1 - p(r)$. If the query terminates, $l(r) = 1$; otherwise, $l(r) = l(r - \delta(r)) + 1$. Hence,

$$l(r) = p(r) \cdot 1 + [1 - p(r)][l(r - \delta(r)) + 1] \tag{5}$$

Considering that $l(r) \gg \delta(r)$, we have

$$\frac{l(r) - l(r - \delta(r))}{\delta(r)} \approx l'(r)$$

Thus, $l(r - \delta(r)) = l(r) - \delta(r)l'(r)$ and

$$l(r) = p(r) \cdot 1 + [1 - p(r)][l(r) - \delta(r)l'(r) + 1] \tag{6}$$

Simplifying the above equation, we prove the lemma. □

Lemma 6. *Assume that $s(r) \gg \delta(r)$ and $s(r)$ is differentiable, $s(r)$, $p(r)$ and $\delta(r)$ satisfy the following equation:*

$$[1 - p(r)]\delta(r)s'(r) + p(r)s(r) = rp(r), \tag{7}$$

where $s'(r)$ is the derivative of $s(r)$ with respect to r.

Proof. The proof is similar to that of Lemma 5. □

Solving $\delta(r)$ and $p(r)$. We now begin to solve $\delta(r)$. Let o_n be the closest points to q among ep's neighbors and θ_m be the vectorial angle between $\vec{e_p q}$ and $\vec{e_p o_n}$. We have the following observation:

Observation 1. *$\delta(r)$ satisfies $\delta(r) = r - \sqrt{r^2 + r_o^2 - 2rr_0 \cos\theta_m}$. When $r \gg r_0$, $\delta(r) = r_0 \cos\theta_m$.*

Proof. Since $\|e_p, q\| = r$, $\|e_p, o_n\| = r$ and the vectorial angle between $\vec{e_p q}$ and $\vec{e_p o_n}$ is θ_m. According to the law of cosines, $\|o_n, q\| = \sqrt{r^2 + r_o^2 - 2rr_0 \cos\theta_m}$. Thus, $\delta(r) = \|e_p, q\| - \|o_n, q\| = r - \sqrt{r^2 + r_o^2 - 2rr_0 \cos\theta_m}$. □

To analyze the distribution of θ_m, we first consider the distribution of the vectorial angle θ_o between $\vec{e_p q}$ and $\vec{e_p o}$, where o is one of neighbors of e_p. Since e_p's neighbors are uniformly distributed around it, θ_o follows the distribution of the vectorial angle between any two random vectors in $\mathbb{R}^{d_i}$, where d_i is the local intrinsic dimensionality (LID) of $\mathcal{D}$ [1].

Lemma 7 (Theorem 1 in [6]). *The probability density function of θ_o is*

$$f_{\theta_o}(\theta) = \frac{\Gamma(\frac{d_i}{2}) sin^{d_i-2}\theta}{\sqrt{\pi}\Gamma(\frac{d_i-1}{2})}, \tag{8}$$

where $\Gamma(x)$ is the Gamma function [24].

Then, the cumulative distribution function of θ_o can be written as $F_{\theta_o}(\theta) = \int_0^\theta f_{\theta_o}(t)dt$. Assume e_p has T_e neighbors. Based on the above assumptions, $\vec{e_p o_n}$ is the edge that has the minimal angle to $\vec{e_p q}$. Then, θ_m is the minimal one among T_e vectorial angles, and thus we have

Corollary 1. *The cumulative distribution function of θ_m is*

$$F_{\theta_m}(\theta) = 1 - [1 - F_{\theta_o}(\theta)]^{T_e}, \tag{9}$$

Correspondingly, the probability density function of θ_m can be written as $f_{\theta_m}(\theta) = F'_{\theta_m}(\theta)$.

We now compute $p(r)$ and $p(r)$ is the probability that the query terminates, which indicates that $\delta(r) < 0$. Hence, we have

Lemma 8. *$p(r)$ satisfies*

$$p(r) = 1 - F_{\theta_m}(\arccos \frac{r_0}{2r}). \tag{10}$$

Proof. $p(r) = \Pr[\delta(r) < 0] = \Pr[\cos\theta_m < \frac{r_0}{2r}] = \Pr[\theta_m > \frac{r_0}{2r}] = 1 - F_{\theta_m}(\arccos \frac{r_0}{2r})$. □

The correctness and the query cost of LSH-APG. Finally, we can go back to $l(r)$ and $s(r)$. Since the formation of $\delta(r)$ and $p(r)$ are extremely complex, it is nearly impossible to directly solve Eq. 4 and 7. However, it is obvious that $l(r)$ and $s(r)$ only depends on the initial search radius r_i, θ_m and r_o. The distribution of θ_m is only affected by LID and independent on n. r_o is the local characteristic od the dataset around e_p, which is also affected little by n. Therefore, $l(r)$ and $s(r)$ are independent on n.

Lemma 9. *Assume that for each vertex in $\mathcal{I}_G$, the length of its edges is identical and its neighbors are uniformly distributed around it, $l(r)$ and $s(r)$ are independent on n and only affected by the local intrinsic dimensionality of the dataset.*

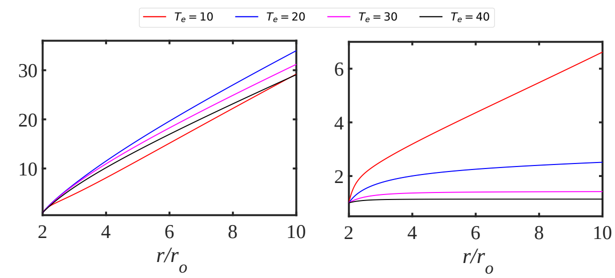
The above lemma ensures that $l(r)$ only depends on d_i, and thus we denote l as $O(\varphi(d_i))$. $C_Q = tl$ and t is bounded by T'. Therefore, C_Q is also $O(\varphi(d_i))$.

Lemma 10. *The expected query cost in LSH-APG is only depends on d_i.*

As for $s(r)$, we aim to further prove that it is close to r_o. $s(r) = \int_{r_o}^{r_s} s'(r)dr + s(r_o)$ and $s(r_o) < r_o$. Hence, we prove that $\int_{r_o}^{r_s} s'(r)dr$ and gives the following conclusion:

Theorem 2. *The expected final search radius of LSH-APG is at most $(1+\gamma)r_o$, where γ only depends on the d_i. Usually, $\gamma < 1$.*

Due to the space limitation, we put the detailed proof of Lemma 2 in our technical report [33]. Note that Lemma 10 and Theorem 2 are the general properties for all kinds of approximation kNN graphs. By using these two conclusions, we prove Theorem 1. Figure 2 shows the curves of $l(r)$ and $s(r)$ with varying r. From Figure 2(a) we can see that $l(r)$ is nearly linear with r when $r > 2$. Denote $C_Q = O(\varphi(d_i))$, we can estimate the value of $\varphi(d_i)$ from the curve. From Figure 2(a) we can see that $s(r)$ will converge to a maximum value s_{max} when r increase and a larger T_e incurs a smaller s_{max}. For a given LID, we can reduce the value of s_{max} to at most $2r_o$ by increasing T_e, which is in line with $\gamma < 1$.



(a) $l(r)$ (b) $s(r)/r_o$

Figure 2: The curves of $l(r)$ and $s(r)$

The benefit from $\mathcal{I}_H$. Since $l(s)$ depends on the initial search radius r_i. By using $\mathcal{I}_H$, we can obtain the closer entry point o_h, which essentially reduce 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, the benefit from $\mathcal{I}_H$ is $B_{\mathcal{I}_H} = l(r_2) - l(r_1)$.

Lemma 11. *The expected $B_{\mathcal{I}_H}$ is $\frac{\Delta r}{r_o \mathbb{E}[\cos\theta_m]}$, where $\mathbb{E}[\cos\theta_m]$ is the expectation of $\cos\theta_m$ and Δr is the reduction of the initial search radius compared to the random entry point.*

Proof. When $r > r_1$, the query is very unlikely to terminate and when can consider $p(r) = 0$. Thus, Eq. 4 can be simplified as $\delta(r)l'(r) = 1$ and we have

$$l(r) = \int_{r_1}^{r} \frac{1}{\delta(r)}dr + l(r_1).$$

Then, $B_{\mathcal{I}_H} = l(r_2) - l(r_1) = \int_{r_1}^{r_2} \frac{1}{\delta(r)}dr$. Further, consider $r \gg r_o$, we approximate $\delta(r)$ as

$$\begin{aligned}\delta(r) =& r - r\sqrt{1 + (\frac{r_o}{r})^2 - 2\frac{r_o}{r}\cos\theta_m} \\ \approx& r - r[1 + \frac{r_o^2}{2r^2} - \frac{r_o}{r}\cos\theta_m] \approx r_o\cos\theta_m\end{aligned}$$

Hence, we have $B_{\mathcal{I}_H} = \frac{r_2 - r_1}{r_o\cos\theta_m}$ and $r_2 - r_1 = \Delta r$. Therefore, the expected $B_{\mathcal{I}_H}$ is $\frac{\Delta r}{r_o\mathbb{E}[\cos\theta_m]}$. □

This lemma indicates that the benefit of the entry point is $O(\Delta r)$, which is in line with Figure 2(a). When $r > r_1$, $\delta(r)$ is nearly a constant and the search radius decreases slowly. 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 $\|q, o\| = r_1 = c'r$, and thus improve the query performance of LSH-APG.

7.2 Analysis of the Pruning Condition

Before proving Lemma 2, we introduce the following conclusion:

LEMMA 12 (LEMMA 4 IN [41]). *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: 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 $\alpha_2 = F(t^2/c^2; m)$.

PROOF. According to the Lemma 12, those points outside $B(q, cr)$ are very likely to have projected distances to q larger than tr. The total number of this kind of points is less than $2\alpha_2 n$ and the total number of points to be verified is at most $2\alpha_2 n + k$. We try to bound α_1 and $2\alpha_2 n + k$. By using the Wilson–Hilferty transformation [22, 39], for a random variable $X \sim \chi^2(m)$, $\sqrt[3]{X/k}$ is approximately distributed with $N(1 - \frac{2}{9m}, \frac{2}{9m})$, and thus $Y = \frac{\sqrt[3]{X/k} - (1 - \frac{2}{9m})}{\sqrt{\frac{2}{9m}}} \sim N(0, 1)$. Consider $t^2 = m(1 - \frac{2}{9m})^3$, $\alpha_1 = \Pr[X \le m(1 - \frac{2}{9m})^3] = \Pr[Y \le 0] = 0.5$, which indicates that a point in $B(q, cr)$ will be accessed with at least the probability 0.5. Likewise, $\alpha_2 = \Phi(c_1(\sqrt{m} - \frac{2}{9}\sqrt{1/m}))$, where $c_1 = \sqrt{\frac{9}{2}}(c^{-2/3} - 1) < 0$ and $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. Next, we prove that α_2 can be bounded by $\exp(-\frac{c_1^2 m}{2})$. Denote $g(u) = \Phi(c_2(u - \frac{2}{9u})) - \exp(-\frac{c_1^2 m}{2})$, it is easy to demonstrate that $g(u)$ first decreases and then increases with u in $(0, +\infty)$, so $g(\sqrt{m}) < \max\{g(0), g(+\infty)\}$. $g(0) = g(+\infty) = 0$ and thus $g(\sqrt{m}) < 0$, which implies $\alpha_2 < \exp(-\frac{c_1^2 m}{2})$. Considering that $m = t \log n$, we have $\exp(-\frac{c_1^2 m}{2}) = n^{-\frac{c_1^2 t}{2}}$ and the total number of points to be verified is at most $2n^{1-\frac{c_1^2 t}{2}} + k$. □

7.3 Complexity Analysis

We consider T as a constant, following the mainstream graph based methods [14, 31, 32]. According to Theorem 1 and Lemma 4, we have the following conclusion.

THEOREM 3. *LSH-APG has the space complexity $O(n)$ and can be built with time $O(nd\varphi(d_i))$, where d_i is the LID of $\mathcal{D}$ and $\varphi(d_i)$ is the expectation of C_Q and independent on n. The query, insertion and deletion cost of LSH-APG are all $O(d\varphi(d_i))$.*

8 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 2 Intel(R) Xeon(R) Gold 5218 CPUs @ 2.30GHz (64 threads) and 254 GB RAM.

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

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

Table 3: Parameter Settings of Algorithms

Algorithm	Parameters
LSH-APG	$K = 16, L = 2, T = 24, T' = 2T, p_\tau = 0.95$
DB-LSH	$c = 1.5, K = 12, L = 5$
HNSW	$M = 48, ef = 80$
HCNNG	$MC = 500, NC10$
NSG	$L = 40, R = 50, C = 500$

8.1 Experimental Settings

Datasets. We employ 8 datasets varying in cardinality and dimensionality, whose informations are summarized in Table 2 in the ascending order of their sizes. **LID.** in the table is used to measure the difficulty of answering NN queries in the dataset [1]. 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[14, 31, 37, 38], including our competitors. The rest 2 synthetic datasets, **Rand10M** and **Gauss10M** are generated from uniform distribution $U(-1, 1)$ and Gaussian distribution $N(0, 1)$ 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 method and graph based methods. Among LSH methods, DB-LSH [37] is proven to have the lowest query complexity and query accuracy. Among graph based methods, a comprehensive experimental comparison 38 of graph based methods have shown that HNSW [31], HCNNG [32] and NSG [14] 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 as shown in Table 3. For HCNNG, MC is the maximum size of the cluster and NC is the number of clustering.

Evaluation Metric. We compare the algorithms from four aspects: indexing quality, indexing efficiency, query quality and query efficiency. The former two are the parts of the indexing performance and the latter two are the parts of the query performance.

Indexing Quality. We report the average degrees, the range of degrees, the standard deviation of degrees of each graph index 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 [11]. 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, which can reflect the quality of the graph. NMCS is defined as follow,

Definition 3 (NMCS (Normalized maximum common subgraph)). *Let $G = (V, E)$ be a graph index, where $V = \mathcal{D}$ is the dataset. Let $G_E = (V, E')$ be the exact NN graph of $\mathcal{D}$ that satisfies:*

- *For any a point $e \in V$, $|G(e)| = |G_E(e)|$;*
- *Let $k' = |G_E(e)|$ and $G_E(e)$ be the k'-NN result of e in $V - e$.*

Then

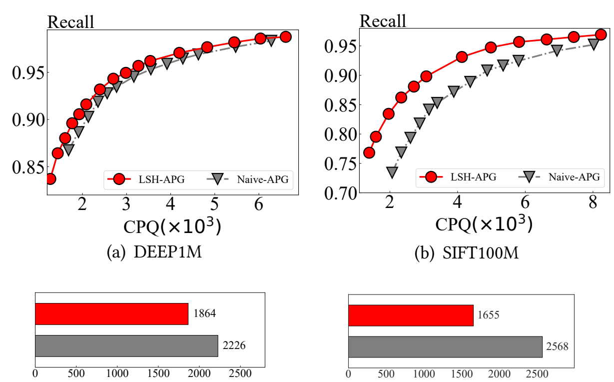
$$NMCS = \frac{\sum_{e \in V} |G(e) \cap G_E(e)|}{\sum_{e \in V} |G(e)|} \tag{11}$$

In the experiments, since the exact NN graph is nearly impossible to compute, we randomly choose 200 vertexes in V to estimate the NMCS.

Query Quality. 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 [37]

$$Recall = \frac{|R \cap R^*|}{k}. \tag{12}$$

Indexing & Query Efficiency. Since all algorithms except DB-LSH adopt openMP parallelizing, their running times are unstable and liable to be affected by the current system status. Therefore, we use the number of distance computations per insertion (**CPI**) and the number of distance computations per query (**CPQ**) instead of running time to evaluate the indexing and query efficiency, respectively. For fairness, we consider the cost of computing hash values and distance in the projected space as parts of distance computations in LSH-APG.



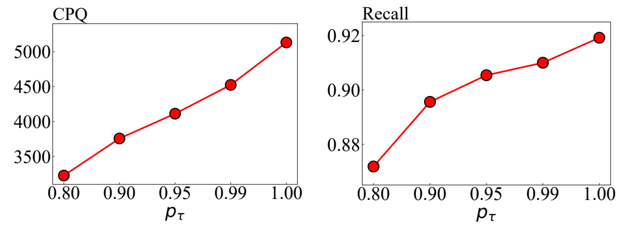
(a) DEEP1M (b) SIFT100M

(c) DEEP1M (d) SIFT100M

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

8.2 Self Evaluation

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



(a) CPQ-p_τ (b) Recall-p_τ

Figure 4: Performance of LSH-APG when Varying p_τ

Evaluation of the LSH framework. To demonstrate the functionality of the LSH framework, we denote by Naive-G our solution without using the LSH framework and compare it with LSH-APG. Here we report the results on DEEP1M and SIFT100M for brevity, as shown in Figure 3. To compare their query performance, we plot their Recall-CPQ curves. The results show that to reach a given recall, LSH-APG reduce about 20% CPQ on DEEP1M and 50% CPQ on SIFT100M, which demonstrates the effectiveness of LSH framework in our solution. To compare their indexing performance, we notice that LSH-APG and Naive-G have nearly the same graph indexes and LSH framework only affects the indexing cost. Therefore, we report the CPIs of LSH-APG and Naive-G and the results show that LSH-APG (the red bar) reduce about 20% CPI on DEEP1M and 60% CPI on SIFT100M, respectively. It indicates that LSH framework plays also an important role in the indexing phase.

Parameter study on p_τ. In this set of experiment, we discuss the effect of p_τ on query performance by varying p_τ in range $\{0.8, 0.9, 0.95, 0.95, 1.0\}$. Here, we only show results on DEEP1M for brevity. p_τ controls the pruning threshold in the query phase. A smaller p_τ makes it more likely to prune an edge. As shown in Figure 4, we obtain the larger recall and CPQ when the value of p_τ increases. It indicates that the pruning condition in LSH-APG can reduce the query cost but also damages the query quality to some extent. Since it inevitably filter some exact results during the pruning. Taking both query efficiency and quality into consideration, we set $p_\tau = 0.9$.

8.3 Evaluation of Indexing Performance

In this set of experiments, We study the index quality and indexing efficiency of all algorithms with the default settings. The results are shown in the Table 4. NSG fails to build the indexes on Tiny80M due to out of memory, so we do not report its results (including the query results) on Tiny80M. From the table we have the following observations when comparing each metric:

(1) In terms of the degree, LSH-APG have the the large average degrees. It is because LSH-APG adopt the simplest neighbor selection strategy and does not delete the similar edges. However, LSH-APG does not need to compare the whole neighbors when querying due to the pruning strategy via LSH, which guarantees its query efficiency. HNSW has the extremely low degree on Gist1M and Tiny80M, although these two datasets have high LIDs. This phenomenon is caused by the heuristic neighbor selection. Comparing the degree among different datasets, we find the distribution of degrees on LSH-APG changes little but the distributions on other graph indexes do not.

Table 4: Overview of index informations. In this table, μ and σ denote the average of degrees and the standard deviation of degrees. Range is the interval where degrees distribute.

		(μ,σ)	Range	NMCS	CPI			(μ,σ)	Range	NMCS	CPI
MNIST	**LSH-APG**	(37.51,9.19)	[24,48]	0.7655	583.59	**GIST1M**	**LSH-APG**	(32.43,9.33)	[24,48]	0.4159	1864.0
	HNSW	(17.21,8.83)	[1,48]	0.4006	928.50		**HNSW**	(10.07,9.86)	[1,48]	0.1159	2144.3
	NSG	(18.19,5.72)	[1,50]	0.4499	3801.7		**NSG**	(16.50,12.53)	[1,77]	0.2929	7027.1
	HCNNG	(11.39,3.03)	[1,26]	0.5353	4801.9		**HCNNG**	(17.50,5.63)	[1,30]	0.1375	5091.8
	DB-LSH	-	-	-	60		**DB-LSH**	-	-	-	60
DEEP1M	**LSH-APG**	(35.84,9.42)	[24,48]	0.5915	1442.9	**SIFT10M**	**LSH-APG**	(36.61,9.31)	[24,48]	0.5194	1476.8
	HNSW	(23.55,10.84)	[1,48]	0.3036	2138.8		**HNSW**	(25.84,10.80)	[1,48]	0.3018	2578.5
	NSG	(20.96,9.17)	[1,50]	0.4510	6291.6		**NSG**	(21.11,9.29)	[1,51]	0.4143	4782.5
	HCNNG	(17.49,4.35)	[2,30]	0.2786	5091.8		**HCNNG**	(17.83,4.09)	[1,30]	0.2163	3341.7
	DB-LSH	-	-	-	50		**DB-LSH**	-	-	-	60
Gauss10M	**LSH-APG**	(32.47,9.21)	[24,48]	0.3645	2266.4	**SIFT100M**	**LSH-APG**	(36.31,9.36)	[24,48]	0.5110	1691.3
	HNSW	(25.43,13.00)	[1,48]	0.2463	8776.5		**HNSW**	(26.68,11.05)	[24,48]	0.2577	3262.2
	NSG	(24.31,13.95)	[1,51]	0.3365	7279.9		**NSG**	(26.84,12.07)	[1,52]	0.3734	7024.7
	HCNNG	(19.68,6.09)	[6,30]	0.0793	3341.8		**HCNNG**	(18.18,4.17)	[1,30]	0.1719	4164.7
	DB-LSH	-	-	-	50		**DB-LSH**	-	-	-	60
Rand10M	**LSH-APG**	(35.76,9.21)	[24,48]	0.5508	2159.2	**Tiny80M**	**LSH-APG**	(32.18,9.15)	[24,48]	0.3159	2494.6
	HNSW	(33.92,9.41)	[1,48]	0.4544	9547.2		**HNSW**	(13.34,10.97)	[1,48]	0.0698	3567.6
	NSG	(36.02,11.78)	[2,50]	0.4126	6573.0		**NSG**	/	/	/	/
	HCNNG	(19.58,4.92)	[7,30]	0.1005	3341.8		**HCNNG**	(17.75,5.99)	[1,30]	0.1436	6380.3
	DB-LSH	-	-	-	60		**DB-LSH**	-	-	-	60

(2) LSH-APG always has the highest NMCS among the graph based algorithms, which indicates it is more similar as an exact NN graph and has the highest-quality neighbors. The higher-quality neighbors enables the query processing to terminate more quickly after finding one of q's NNs.

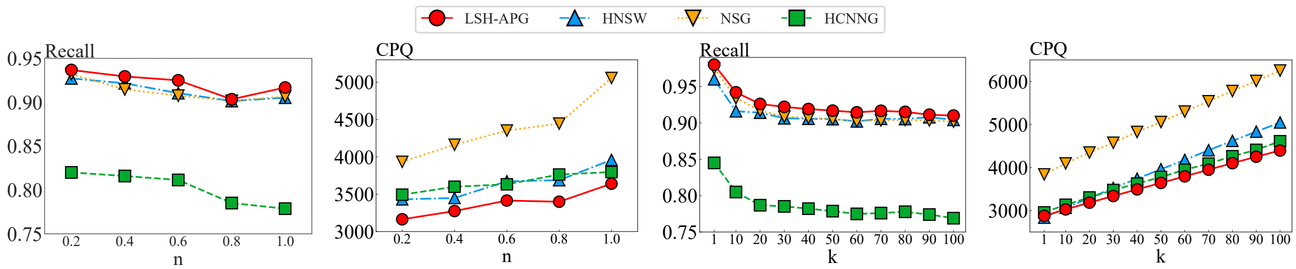
(3) In terms of CPI, DB-LSH achieves the smallest CPI on all datasets, which is much smaller than any algorithms. The CPI of LSH based methods comes from the cost of computing hash values. Usually the number of hash values will not be very high. So, LSH indexes can be built much faster compared to graph indexes. Among the graph based algorithms, LSH-APG has the smallest CPI and only HNSW's CPI is comparable. NSG and HCNNG both have the extremely larger CPIs. The reason of this phenomenon is that: First, by choosing a close entry point and pruning far points when finding neighbors via LSH framework, LSH-APG avoids many unnecessary computation when inserting the points. Second, compared to HNSW, LSH-APG does not adopt the neighbor selection strategy like that in HNSW or NSG, which is computation-consuming since it requires comparing every two neighbors. In NSG, half the CPI comes from iteratively building an approximation kNN graph with a large degree K via NN-Descent strategy. Even so, the CPI of building the NSG in the base of an NNG is still higher than that of LSH-APG. In HCNNG, it requires computing the distance between any two points in a cluster, which incurs a large CPI.

(4) Comparing the NMCS and CPI among different datasets, we can find all the algorithms follows the similar tendency. For example, LSH-APG has the smaller CPI and higher NMCS on DEEP1M than that on GIST1M. Then, HNSW, NSG and HCNNG all have the smaller CPI and higher NMCS on DEEP1M than that on GIST1M, respectively. Moreover, we find the CPI and NMCS mainly depends on the data cardinality and LID instead of the data dimensionality. For LSH-APG, the CPI increase gradually with the data cardinality (about $O(\log n)$) when comparing CPI on SIFT10M and SIFT100M. The CPI and NMCS are also greatly influenced by LID. A larger LID incurs a larger CPI and a smaller NMCS. For other algorithms, their NMCS and CPI are also influenced by data cardinality and LID. Astonishingly, HNSW has the highest CPI on Gauss10M and Rand10M (8776.5 and 9547.2), which is even much higher than that on SIFT100M and Tiny80M (3262.2 and 3567.6). It indicates HNSW is hard to process the random dataset.

8.4 Evaluation of Query Performance

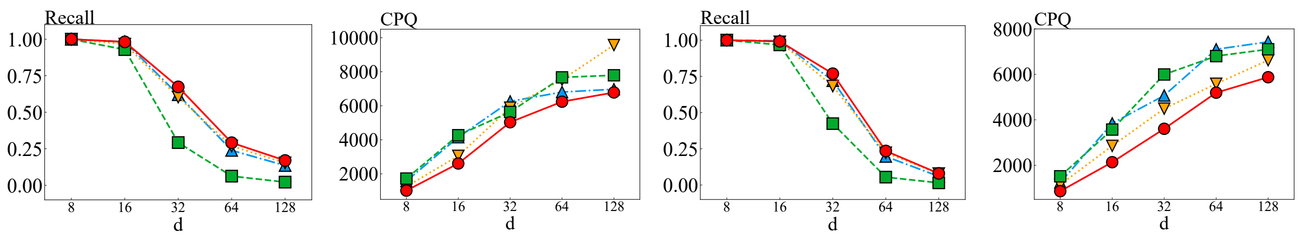
In these sets of experiments, we study the query performance of all algorithms. We first study how the characteristics of dataset, *i.e.,* data cardinality n and dimensionality d, affect the query performance. Then, we study the effect of k. Finally, we analysis the trade-off between the query quality and query efficiency by increasing the number of checked points. DB-LSH always needs the much larger query cost to reach the similar recall than the graph based methods. For example, the CPQ of DB-LSH to reach a recall of 0.95 is about 8M, which is 500 times higher than the worst graph based methods. Therefore, we do not report its results in the rest experiments.

8.4.1 ***Effect of*** n. By randomly selecting some points from the original dataset, we compare the query performance of all algorithms in the default settings. Limited by the space, we only report the result on SIFT100M, as shown in Figure 5(a)-(b). The range of the cardinality n is $\{0.2N_0, 0.4N_0, \ldots, N_0\}$, where N_0 is the size of the original datasets. As shown in the figures, each algorithm has the larger CPQ and nearly the smaller recall when n increases. The increase of CPQs are all sublinear (about $O(\log n)$), which demonstrate the graph based methods can answer ANN query in $O(\log n)$



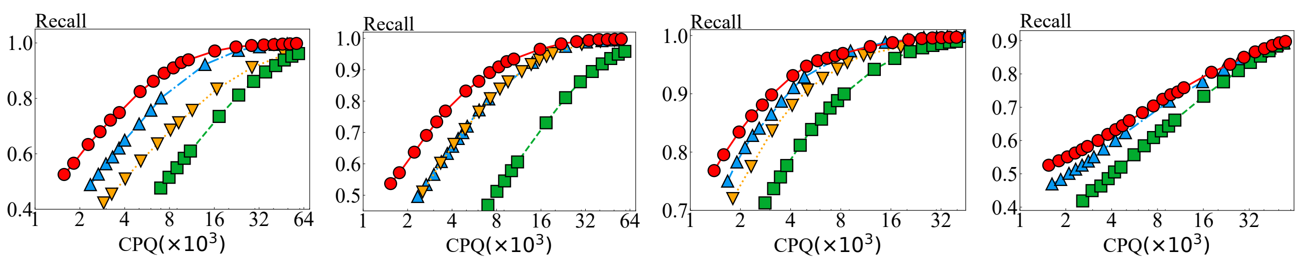
(a) Recall-n (b) CPQ-n (c) Recall-k (d) CPQ-k

Figure 5: Performance on SIFT100M when Varying n and k



(a) Recall on Gauss10M (b) CPQ on Gauss10M (c) Recall on Rand10M (d) CPQ on Rand10M

Figure 6: Performance on Rand10M and Gauss10M when Varying d



(a) Gauss10M (b) Rand10M (c) SIFT100M (d) Tiny80M

Figure 7: Recall-CPQ Curves on all Datasets

cost. In addition, LSH-APG always achieves the smallest CPQ and the highest recall, which indicates it performs best among the competitor algorithms. NSG and HNSW achieves nearly the same recall. The reason is that NSG and HNSW adopt the same edge selection strategy and thus are likely to store the similar edges. However, HNSW has the smaller CPQ than NSG.

8.4.2 ***Effect of*** k. We compare the query performance of all algorithms when varying k of (c, k)-ANN query in $\{1, 10, 20, \ldots, 100\}$. Limited by the space, we only report the result on SIFT100M, as shown in Figure 5(c)-(d). From the figures we can see that CPQ of each algorithm increases nearly linearly with k but LSH-APG has the smallest slope, which demonstrate the superiority of LSH-APG again. When comparing the performance among different algorithm, LSH-APG always achieves the smallest CPQ and the highest recall. NSG and HNSW achieves the close recall and their curves also have about the same slope, which further demonstrate the similarity between them and HNSW is the better one.

8.4.3 ***Effect of*** d. In the default settings, we compare the query performance of all algorithms when varying the dimensionality d of dataset. Due to the unbalanced distribution in real-world datasets, it is not meaningful to pick up parts of dimensionality on them. So, we only compare the algorithms on two synthetic datasets, Rand10M and Gauss10M, the range of d is $\{8, 16, 32, 64, 128\}$. The results are shown in Figure 6. As shown in the figures, the CPQ of each algorithm will increase with d increasing. The increasing tendencies of LSH-APG, HNSW and HCNNG are sublinear but that of NSG are nearly linear, which indicates NSG is more likely to be affected by the dimensionality. As for the recall, we surprisingly find that the recall of each algorithm decreases very rapidly with d. When d is 8 or 16, all algorithms can reach the recall of nearly 1.0; when d comes to 32, recalls all declines to about 0.6 on Gauss10M and 0.75 on Rand10M; when d reaches 64 or even 128, recalls have gone down to less than 0.3. This results show that the effect of the dimensionality on the recall is much greater than that of the cardinality. This result can be explained by the "curse of dimensionality",*i.e.,* when the dimensionality is large enough, the distance between any two points is almost identical and it becomes very difficult to differentiate the NNs and other points for any query. The "curse of dimensionality" phenomenon becomes very obvious on the random datasets, even when d is only 64 or 128. This is because with the same dimensionality, the random datasets have the much higher LID than the real-world datasets.

8.4.4 ***Recall-CPQ Curve.*** For any ANN methods, a more accurate result can be obtained by increasing the number of checked points, which consequently damages the query efficiency. Therefore, there is a trade-off between the query quality and query efficiency. In this set of experiments, we analyze the trade-off between them via the Recall-CPQ Curve. To reach the same target recall, the algorithm with the smaller CPQ has a higher query performance. Figure 7 present the results on four datasets.

From the figures we have the following observations: (1) As the CPQ increases, all algorithms achieve the higher recall, which is in line with the philosophy of ANN methods, *i.e.,* trading accuracy for efficiency. Moreover, the required CPQ to reach a given recall *Rec* increases nearly exponentially with *Rec*, which indirectly demonstrates that finding the exact NNs is extremely hard and time-consuming. (2) Among all algorithms, LSH-APG requires the smallest CPQ to reach the same recall, which indicates LSH-APG achieves the best trade-off between the query accuracy and query efficiency. As analyzed in the previous part, the advantages of LSH-APG are due to the high-quality entry points found in the LSH indexes and the pruning condition that enables us to filter some neighbors in the graph. (3) NSG and HNSW always exhibit the similar tendency and even nearly the same curve on Rand10M, which demonstrates that they may have the similar structure since they adopt the same edge selection strategy. However, in the most cases, HNSW is the better algorithm. (4) HCNNG always achieves the worst query performance. Especially on Gauss10M and Rand10M, to reach the same recall, HCNNG requires nearly 4 times as large CPQ as LSH-APG. HCNNG adopts the cluster based approach to construct subgraphs. However, the data in the high-dimensional space are not "well" clustered and we could not find proper clusters. We consider this is why HCNNG achieve such performance on these datasets.

8.5 Evaluation of the LSH Extension

In this experiments, we take HCNNG as example and evaluate its LSH Extension, LSH-HCNNG, with it on SIFT100M. As said in Section 6.2, we adopt the LSH indexes to partition the data and only compute the distance of the points in the same hash buckets for LSH-HCNNG. We set $MC = 500$ and $NC = 20$ for LSH-HCNNG and vary the number of points in a bucket, NB, in $\{25, 40, 80, 160\}$. The results are reported on Figure 8. From the figure we can see LSH-HCNNG has a much smaller CPI than HCNNG. When $NB = 20$, its CPI is only 10% of HCNNG's. Even though with $NB = 160$, LSH-HCNNG has about a 40% smaller CPI than HCNNG. Moreover, the CPQ of LSH-HCNNG is also smaller due to the better entry point and the pruning condition. However, the recall of LSH-HCNNG is a bit lower than HCNNG, especially when $NB = 20, 40$. This is because LSH-HCNNG only check a small part of point pairs in a group and build an approximate PG; while HCNNG build the exact PG (the minimal spanning tree) in a group and has a higher graph quality. Compared to the huge improvement on CPQ and CPI, this loss is acceptable and the LSH framework is available for HCNNG.

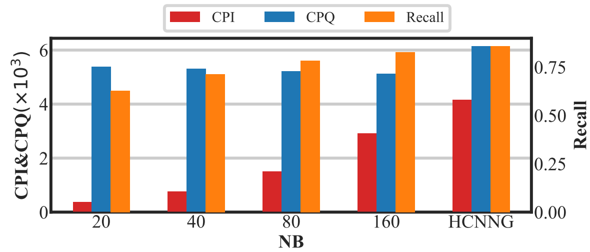


Figure 8: Performance of HCNNG and its LSH Extension

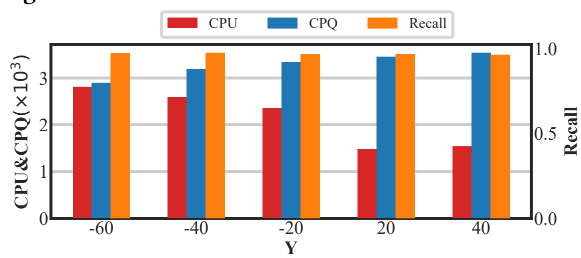


Figure 9: Performance on DEEP1M when Updating

8.6 Evaluation of Updating

we follow [18] and evaluate the update performance of LSH-APG via a batch insertion or deletion. Given the graph index of LSH-APG, $\mathcal{I}_G = (V_0, E_0)$, we denote a batch insertion (resp. deletion) of the graph as $Y\%$ update where $Y = \frac{|V|-|V_0|}{|V_0|}$ and $|V|$ is the number of points in the graph after insertion or deletion. In this set of experiments, we study the the effect of Y on DEEP1M with $V_0 = 600K$ when varying Y in $\{-60, -40, -20, 20, 40\}$ ($Y < 0$ denotes the deletion operation) and report the query performance and the cost per update (CPU) on Figure 9. As for the updating cost, the insertion cost is obvious smaller than the deletion cost and 40% insertion has the higher cost than 20% insertion. The deletion cost increases slightly with $|Y|$, which can be explained that the total number of points in the graph decreases and The probability that a vertex connecting to the points to be deleted increases, and thus it takes more time to add the new edges. For the query performance, we find the recall is steady and CPQ gradually increases with Y since the number of the points increases. It indicates that the update operations do not damage the 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] 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.

[6] T. Tony Cai, Jianqing Fan, and Tiefeng Jiang. 2013. Distributions of angles in random packing on spheres. *J. Mach. Learn. Res.* 14, 1 (2013), 1837–1864.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

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

[13] 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

[14] 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.

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

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

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

[18] 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.

[19] 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.

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

[21] 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.

[22] NL Johnson, S Kotz, and N Balakrishnan. 1994. Chi-squared distributions including chi and Rayleigh. *Continuous univariate distributions* 1 (1994), 415–493.

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

[24] Cornelius Lanczos. 1964. A precision approximation of the gamma function. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* 1, 1 (1964), 86–96.

[25] 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.

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

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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).

[33] Technical Report. 2022. *Available at:*. https://www.mit.edu/~andoni/LSH

[34] 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.

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

[36] 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.

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

[38] 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.

[39] Edwin B Wilson and Margaret M Hilferty. 1931. The distribution of chi-square. *proceedings of the National Academy of Sciences of the United States of America* 17, 12 (1931), 684.

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

[41] 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.