Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph

Cong Fu, Chao Xiang, Changxu Wang, Deng Cai
The State Key Lab of CAD&CG, College of Computer Science, Zhejiang University, China
Alibaba-Zhejiang University Joint Institute of Frontier Technologies
*f*fc731097343, changxu.mail, dengcai*g*@gmail.com; chaoxiang@zju.edu.cn

ABSTRACT

The approximate nearest neighbor search (ANNS) is a fundamental problem in data base and data mining. A scalable ANNS algorithm should be both memory-saving and search-efficient. Some early graph-based approaches have shown attractive theoretical guarantees on search time complexity, but they all suffer from the problem of high indexing time complexity. Recently, some practical graph-based methods are proposed to reduce the indexing complexity by approximating the traditional graphs and have achieved revolutionary performance on public datasets. However, these works are mainly based on intuitions and lack of rigorous theoretical support. They fail to maximize the potential of graph-based search. In our observation, an ideal graph for ANNS should consider four aspects, (1) ensuring the connectivity of the graph; (2) lowering the average out-degree of the graph for fast traversing; (3) shortening the search path; and (4) avoiding additional index structures to reduce the index size. In this paper, we introduce a new graph structure called Monotonic Relative Neighborhood Graph (MRNG) which takes the four aspects into account simultaneously and guarantees very low search complexity (close to logarithmic time). To further lower the indexing complexity, we propose a novel graph structure named Navigating Spreading-out Graph (NSG) for practical large-scale ANNS problems by approximating the MRNG. Extensive experiments show that NSG outperforms all the existing algorithms significantly. What's more, NSG shows superior performance in the e-commercial search scenario of Taobao (Alibaba Group) and has been integrated into their search engine for billion-scale search.

XXXXX Reference Format:
Cong Fu, Chao Xiang, Changxu Wang, Deng Cai. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *XXXXX*, xx (xx): xxxx-yyyy, 2018.
DOI: https://doi.org/XXX

Corresponding author.

Proceedings of the XXXX, Vol. xx, No. xx

DOI: https://doi.org/XXX

1. INTRODUCTION

Approximate nearest neighbor search (ANNS) has been a hot topic over decades and provides fundamental support for many applications in data mining, data base and information retrieval [2, 8, 10, 19, 32, 37]. For sparse discrete data (like documents), the nearest neighbor search can be carried out efficiently on advanced index structures (*e.g.*, inverted index [30]). For dense continuous vectors, various solutions have been proposed such as tree-structure based approaches [6, 14, 20, 31], hashing-based approaches [15, 17, 34], quantization-based approaches [1, 16, 22] and graph-based approaches [3, 18, 28, 35]. Among them, graph-based methods have shown great potential recently, and have demonstrated the significant improvements over other methods experimentally [13, 23].

Nearest neighbor search via graphs has been studied for decades [3,11,21]. Given a set of points S in the d-dimensional Euclidean space E^d, a graph G is defined as a set of edges connecting these points (nodes). The edge pq defines a neighbor-relationship between node p and q. Various constraints are proposed on the edges to make the graphs suitable for ANNS problem. These graphs are now referred to as the *Proximity Graphs* [21]. Some proximity graphs like Delaunay Graphs (or Delaunay Triangulation) [4] and Monotonic Search Networks(MSNET) [11] ensures that, from any node p to another node q, there exists a path on which the intermediate nodes are closer and closer to q step by step [11]. However, the complexity to find such a path is not given. Other works like Randomized Neighborhood Graphs [3] guarantee polylogarithmic search time complexity. The average length of greedy-routing paths grows polylogarithmically with the data size on the Navigable Small-World Networks (NSWN) [7, 25], which is an empirical conclusion. However, the time complexity of building these graphs is very high (at least $O(n^2)$), which is impractical for massive problems.

The recent practical graph-based methods try to address this problem by designing approximations for those graphs. For example, GNNS [18], IEH [23], and Efanna [13] are based on the kNN graph, which is an approximation of the Delaunay Graph. NSW [28] approximates the NSWN, FANNG [5] approximates the Relative Neighborhood Graphs (RNG) [33], and Hierarchical NSW (HNSW) [29] is proposed to take the advantages of the Delaunay Graph, the NSWN, and the RNG. Moreover, a hierarchical structure is used in HNSW to enable multi-scale hopping on different layers of the graph.

These approximations are mainly based on intuitions and

lack of rigorous theoretical support. We cannot maximize the ANNS potential of graph-based algorithm unless we look into the essence of the search process on the graphs. Provided with a starting node p and a query q, the *search-on-graph* algorithm will find a path from p to q (or its nearest neighbor) in a greedy way. At each step, the algorithm always goes to the neighbor node of the current node, which minimizes the distance to the query q (given in Algorithm 1). The search time complexity is mainly determined by two factors, the length of the path (defined as the number of edges on the path) and the number of neighbors to be checked at each step. Through comprehensive theoretical analysis, we find that an ideal graph for search should take four aspects into consideration. (1) Ensure the connectivity of the graph to make sure the existence of the path; (2) Lower the average out-degree of the graph and (3) shorten the search path to lower the search time complexity; (4) Avoid extra index structures to further control the memory use. Works like IEH [23], Efanna [13], and HNSW [29], use hashing, randomized KD-trees and multi-layer graphs to accelerate the search. However, the contribution of these extra structures will be minor if the graph is efficient enough in the search. Removing these structures and reducing the memory use will increase the scalability of the method.

In this paper, we propose a new graph structure, Monotonic Relative Neighborhood Graph (MRNG) which guarantees a low average search time complexity (very close to logarithmic time complexity). To further reduce the indexing complexity for the large-scale ANNS problem, we propose the Navigating Spreading-out Graph (NSG), which is a good approximation of MRNG, inherits the low search complexity and takes all the four aspects into account. It is worthwhile to highlight our contributions as follows.

1. We dig deep into the critical factors influencing the search performance of graph-based ANNS methods and propose a novel graph, MRNG, which ensures a close-logarithmic search complexity in expectation.
2. We present comprehensive theoretical analysis on how to design an ideal graph structure for ANNS problem, which can be summarized as four points: ensuring the connectivity of the graph, lowering the average out-degree, shortening the search path, and avoiding additional index structures.
3. We design a close approximation of the MRNG, named as Navigating Spreading-out Graph (NSG), to address the four aspects simultaneously. The indexing complexity is reduced significantly compared to the MRNG and is practical for massive problems. Extensive experiments show that our approach outperforms the state-of-the-art methods in search performance with the smallest memory use among graph-based methods.
4. The NSG algorithm has been tested in the e-commercial search scenario of Taobao (Alibaba Group), and it showed superior performance and has been integrated into their search engine for billion-scale search.

2. PRELIMINARIES

We use E^d to denote the Euclidean space under the l_2 norm. The closeness of any two points p, q is defined as the l_2 distance, $\delta(p, q)$, between them.

Algorithm 1 Search-on-Graph(G, $\mathbf{p}$, $\mathbf{q}$, l)

Require: graph G, start node $\mathbf{p}$, query point $\mathbf{q}$, candidate pool size l
Ensure: k nearest neighbors of $\mathbf{q}$
1: i=0
2: candidate pool $S = \emptyset$
3: S.add($\mathbf{p}$)
4: **while** $i < l$ **do**
5: i =the index of the first unchecked node in S
6: mark $\mathbf{p_i}$ as checked
7: **for all** neighbor $\mathbf{n}$ of $\mathbf{p_i}$ in G **do**
8: S.add($\mathbf{n}$)
9: **end for**
10: sort S in ascending order of the distance to $\mathbf{q}$
11: remove the distant nodes in S to keep its size no larger
12: than l
13: **end while**
14: return the first k nodes in S

2.1 Problem Setting

Various applications in information retrieval and database management of high-dimensional data can be abstracted as the nearest neighbor search problem in high-dimensional space. The Nearest Neighbor Search (NNS) problem is defined as follows [17]:

DEFINITION 1 (**Nearest Neighbor Search**). *Given a finite point set S of n points in space E^d, preprocess S to efficiently answer the queries that, return a point p 2 S which is closest to a given query point q.*

This naturally generalizes to the K **Nearest Neighbor Search** when we require the algorithm to return K points ($K > 1$) which are the closest to the query point. The approximate version of the nearest neighbor search problem (ANNS) can be defined as follows [17]:

DEFINITION 2 (ϵ-**Nearest Neighbor Search**). *Given a finite point set S of n points in space E^d, preprocess S to efficiently answer the queries that, return a point p in S such that $\delta(p, q)$ $(1 + \epsilon)\delta(r, q)$, where r is the nearest neighbor of q in S.*

Similarly, this problem can generalizes to the approximate K **Nearest Neighbor Search (AKNNS)** when we require the algorithm to return K points ($K > 1$) such that $8i = 1, ..., K, \delta(p_i, q)$ $(1 + \epsilon)\delta(r, q)$. Due to the intrinsic difficulty of exact nearest neighbor search, most researchers turn to AKNNS. The main motivation is to trade a little loss in accuracy for much shorter search time.

For the convenience of modeling and evaluation, we usually don't calculate the exact value of ϵ and use another indicator to show the degree of the approximation, the *precision*. Suppose the point set returned by an AKNNS algorithm of a given query q is R' and the correct k nearest neighbor set of q is R, then the *precision* (accuracy) is defined as below [13].

$$precision(R') = \frac{jR' \setminus Rj}{jR'j} = \frac{jR' \setminus Rj}{K}. \quad (1)$$

A higher *precision* corresponds to a smaller ϵ, thus, a higher degree of approximation. In this paper, we use the *precision* as the evaluation metric.

2.2 Graph-Based ANNS Methods

Given a finite point set S in E^d, a graph is a structure amounting to a set of nodes (representing the points) and

edges which link some pairs of the nodes. A node p is called a neighbor of q if and only if there is an edge between p and q. The most popular way to solve the NNS problem via a graph can be converted to that, given a start node s and a query node q, find a path from s to p then return p, where p 2 S is the closest node to q. For some reason (*e.g.*, the property of a specific graph structure and the limit on response time), we may not expect to reach the exact nearest neighbor of q, but we aim to get close to q as much as possible. During the past decades, many graphs are designed for efficient ANNS. Here we will introduce several graph structures with appealing theoretical properties which have drawn so much interest in the NNS literature.

Delaunay Graphs (or Delaunay Triangulations) are defined as the dual graph of the Voronoi diagram [4]. It is proved to be a monotonic search network [26], but the time complexity of high-dimensional ANNS on a Delaunay Graph is high. According to Harwood *et al.* [5], Delaunay Graphs quickly becomes almost fully connected at high dimensionality. Thus the efficiency of the search reduces dramatically. GNNS [18] is based on the (approximate) kNN graph, an approximation of Delaunay Graphs. IEH [23] and Efanna [13] are also based on the (approximate) kNN graph. They use hashing and Randomized KD-trees to provide better starting position of Algorithm 1 for the kNN graph. Although they improve the performance, they suffer from large and complex indices.

Wen *et al.* [27] propose a graph structure called *DPG*, which is built upon an approximate kNN graph. They propose an edge selection strategy to cut off half of the edges from the prebuilt kNN graph and maximize the average angle among the remaining edges. Finally, they will make compensation on the graph to produce an undirected one. Their intuition is to make the edges among edges to distribute evenly around each node in terms of the angles, but it lacks theoretical support. According to our empirical study, the DPG suffers from a large index and inferior search performance.

Relative Neighborhood Graphs (RNG) [33] are not designed for the ANNS problem in the first place. However, RNG has shown great potential in ANNS area. The RNG adopts an interesting edge selection strategy to eliminate the longest edge in all the possible triangles on S. With this strategy, the RNG reduces its average out-degree to a constant $C_d + o(1)$, which is only related to the dimension d and usually very small [21]. However, according to Dearholt *et al.*'s study [11], the RNG doesn't have sufficient edges to be a monotonic search network due to the strict edge selection strategy. Therefore, there is no theoretical guarantee on the length of the path. Dearholt *et al.*propose a method to add edges to the RNG to turn it to a Monotonic Search Network (MSNET) with the minimal amount of edges, named as the minimal MSNET [11]. The algorithm is based on a prebuilt RNG, the indexing complexity of which is $O(n^{2-\frac{2}{1+d}+\epsilon})$, under the general position assumption [21]. The post-processing of building the minimal MSNET is of $O(n^2 \log n + n^3)$ complexity. The total indexing complexity of the minimal MSNET is huge for high-dimensional and massive databases. Recent practical graph-based methods like FANNG [5] and HNSW [29] adopt the RNG's edge selection strategy to reduce the degree of their graphs and improve the search performance. However, they did not provide a theoretical analysis.

Navigable Small-World Networks [7, 25] are suitable for the ANNS problem by nature. The degree of the nodes and the neighbors of each node are all assigned according to a specific probability distribution. The length of the search path on this graph grows polylogarithmically with the network size, $O(A[logN]^{\nu})$, where A and ν are some constants. This is an empirical estimation, which hasn't been proved. Thus the total search complexity is $O(AD[logN]^{\nu})$, D is the average degree of the graph. The degree of the graph needs to be carefully chosen, which has a great influence on the search efficiency. Like the other traditional graphs, the time complexity of building such a graph is about $O(n^2)$ in a naive way, which is impractical for massive problems. Yury *et al.* [28] propose NSW graphs to approximate the Navigable Small-World Networks and the Delaunay Graphs simultaneously. But soon they find that the degree of the graph is too high to be efficient and there also exist connectivity problems in it. They later propose HNSW [29] to address this problem. They stack multiple NSW into a hierarchical structure to solve the connectivity problem. The nodes in the upper layers are sampled through a probability distribution, and the size of the NSW shrinks from bottom to top layer by layer. Their intuition is that the upper layers ensure long-range short-cuts for fast locating of the destination neighborhood. Then they use the RNG's edge selection strategy to reduce the degree of their graphs. HNSW is the most efficient ANNS algorithm so far, according to some open source benchmarks on GitHub[1].

Randomized Neighborhood Graphs [3] are designed for ANNS problem in high-dimensional space. It is constructed in a randomized way. They first partition the space around each node with a set of convex cones, then they select $O(\log\ n)$ closest nodes in each cone as its neighbors. They prove that the search time complexity on this graph is $O((\log\ n)^3)$, which is very attractive. However, its indexing complexity is too high. To reduce the indexing complexity, they propose a variant, called RNG*. The RNG* also adopts the edge selection strategy of RNG and use additional structures (KD-trees) to improve the search performance. However, the time complexity of its indexing is still as high as $O(n^2)$ [3].

3. ALGORITHMS AND ANALYSIS

We will discuss directed graphs in the remaining parts of this paper. The search algorithm on a graph almost remains unchanged during the last decades (Algorithm 1), two most crucial factors influencing the search efficiency are how many hops it needs from the starting node to the destination and how much it costs to decide the next move at each step. In other words, the search time complexity on a graph can be written as $O(ol)$, where o is the average out-degree of the graph and l is the length of the search path. To design an ideal graph for ANNS, we should first ensure the existence of such a path. Otherwise, the target can never be reached. Next, we should minimize both o and l simultaneously, *i.e.*, shortening the search path and lowering the average out-degree.

Some recent graph-based methods, like IEH [23], Efanna [13], HNSW [29], use extra index structures to get improvement on the performance. The main purpose of using extra index structures is to obtain coarse search results quickly

[1]https://github.com/erikbern/ann-benchmarks

and use them as the initialization of the on-graph-search. However, if the traversing on the graph is fast enough, these extra indices will not be necessary. And avoiding using extra index structures can save the memory usage significantly.

In conclusion, the four criteria to design an ideal graph for ANNS is **(1) ensuring the connectivity of the graph, (2) lowering the average out-degree of the graph and (3) shortening the search path, and (4) avoiding extra index structures.**

3.1 Graph Monotonicity And Path Length

The speed of ANNS on graphs is mainly determined by two factors, the length of the search path and the average out-degree of the graph. Our goal is to find a graph with both low degree and short search path. We will begin our discussion with how to design a graph with very short search paths. Before we introduce our proposal, we will first present a detailed analysis of a category of graphs called *Monotonic Search Networks (MSNET)*, which are first discussed in [11] and have shown great potential in ANNS. Here we will present the definition of the MSNETs.

3.1.1 Definition And Notation

Given a point set S in E^d space, p, q are any two points in S. Let $B(p, r)$ denote an **open sphere** such that $B(p, r) = \{x | \delta(x, p) < r\}$. Let $\overrightarrow{pq}$ denote a directed edge from p to q.

First we will give a definition of the monotonic path in a given graph as follows:

DEFINITION 3 (**Monotonic Path**). *Given a finite point set S of n points in space E^d, p, q are any two points in S and G denotes a graph defined on S. Let $v_1, v_2, ..., v_k, (v_1 = p, v_k = q)$ denote a path from p to q in G,* i.e., *$\forall i = 1, ..., k - 1$, edge $v_i\overrightarrow{v_{i+1}} \in G$. This path is a monotonic path if and only if $\forall i = 1, ..., k - 1, \delta(v_i, q) > \delta(v_{i+1}, q)$.*

Then the monotonic search network can be defined as follows:

DEFINITION 4 (**Monotonic Search Network**). *Given a finite point set S of n points in space E^d, a graph defined on S is a monotonic search network if and only if there exists at least one monotonic path from p to q for any two nodes $p, q \in S$.*

3.1.2 Analysis On Monotonic Search Networks

The Monotonic Search Networks (MSNET) [11] are a category of graphs which can guarantee a monotonic path between any two nodes in the graph. MSNETs are strongly connected graphs by nature, which ensures the connectivity. When traveling on a monotonic path, we always make progress to the destination at each step. In an MSNET, Dearholt *et al.*thought that one can use Algorithm 1 (commonly used in graph-based search) to detect the monotonic path to the destination node, *i.e.*, no back-tracing is needed [11], which is a very attractive property. Back-tracing means, when the algorithm cannot find a closer neighbor to the query (*i.e.*a local optimal), we need to go back to the visited nodes and find an alternative direction to move on. This Monotonicity of the MSNETs makes the search behavior of Algorithm 1 on the graph almost definite and analyzable. However, Dearholt *et al.* [11] failed to provide a proof of this property. In this section, we will give a concrete proof of this property.

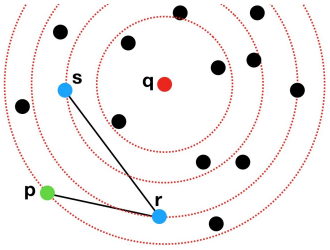


Figure 1: An illustration of the search in an MSNET. The query point is q and the search starts with point p. At each step, Algorithm 1 will select a node that is the closest to q among the neighbors of the current nodes. Suppose p, r, s is on a monotonic path selected by Algorithm 1. The search region shrinks from sphere $B(q, \delta(p, q))$ to $B(q, \delta(r, q))$, then to $B(q, \delta(s, q))$. The number of nodes in each sphere (may be checked) decreases by some ratio at each step until only the q is left in the final sphere.

THEOREM 1. *Given a finite point set S of n points, randomly distributed in space E^d and a monotonic search network G constructed on S, a monotonic path between any two nodes p, q in G can be found by Algorithm 1 without back-tracing.*

PROOF. We can convert this problem into proving the following proposition. For any two points $p, q \in S$, we treat q as the query and p as the search starting point. For any $0 < t < n$, after t iterations of Algorithm 1 on the MSNET without back-tracing, the current subpath $v_1, ..., v_t, (v_1 = p)$ found by the algorithm is monotonic about q, which means $\forall i \in \{1, ..., t - 1\}, \delta(v_i, q) > \delta(v_{i+1}, q)$.

If this proposition is true, then we can reach q from p after at most $n - 1$ iterations of Algorithm 1. Because after $n - 1$ iterations, we will get a sequence $\{\delta(v_1, q), ..., \delta(v_n, q)\}$. Because the path $v_1, ..., v_n$ is monotonic about q, we have $\forall i \in \{1, ..., n - 1\}, \delta(v_i, q) > \delta(v_{i+1}, q)$. If $\delta(v_n, q) > 0$, we have $\forall i \in \{1, ..., n\}, v_i \neq q$. Thus, we can conclude that $q \notin S$ and arrive at a contradiction. Therefore, we can reach q in at most $n - 1$ iterations.

Now we will prove the new proposition with mathematical induction.

(1) Suppose $t = 1$. There must exist at least one monotonic path between v_1 and q in the MSNET. We select one monotonic path from v_1 to q, there must be a node r on the monotonic path such that r is the neighbor of v_1 and $\delta(v_1, q) > \delta(r, q)$. If $r = v_2$, then it's monotonic from v_1 to v_2 about q. Otherwise, because v_2 is found by Algorithm 1 without back-tracing, v_2 is the closest node to q amongst v_1's neighbors. Therefore, $\delta(v_2, q) < \delta(r, q) < \delta(v_1, q)$. It's monotonic from v_1 to v_2 about q.

(2) Suppose the proposition is true when $t = m$, the path $v_1, ..., v_m$ found by the algorithm without back-tracing is monotonic about q. When $t = m + 1$, we select one monotonic path from v_m to q, there must be a node s on the monotonic path such that s is the neighbor of v_m and $\delta(v_m, q) > \delta(s, q)$. If $s = v_{m+1}$, then it's monotonic from v_1 to v_{m+1} about q. Otherwise, because v_{m+1} is found by Algorithm 1 without back-tracing, v_{m+1} is the closest node to q amongst v_m's neighbors. Therefore, $\delta(v_{m+1}, q) < \delta(s, q) < \delta(v_m, q)$.

It's monotonic from v_1 to v_{m+1} about q.

Therefore, the new proposition is true, and we can find a monotonic path between any two nodes p, q in G with Algorithm 1 without back-tracing. □

From Theorem 1, we know that we can reach the query q 2 S on a given MSNET with Algorithm 1 without back-tracing, Therefore, the number of the iterations in expectation is the same as the length expectation of a monotonic path in the MSNET. Before we discuss the length expectation of a monotonic path in a given MSNET, we first define the MSNETs from a different perspective, which will help with the analysis.

LEMMA 1. *Given a graph G on a set S of n points in E^d, G is an MSNET if and only if for any two nodes p, q, there is at least one edge $\overrightarrow{pr}$ such that r 2 $B(q, \delta(p, q))$.*

PROOF. (1)Sufficiency. If edge $\overrightarrow{pq}$ 2 G, we have $\overrightarrow{pq}$ 2 $B(q, \delta(p, q))$. If not, because G is an MSNET, for any two nodes p, q, there exists a monotonic path from p to q, denoted as $v_1, ..., v_k, (k > 2, v_1 = p, v_k = q)$. We have $\delta(v_1, q) > \delta(v_2, q)$. Therefore, v_2 2 $B(q, \delta(v_1, q))$. Because v_2 is connected to $p(v_1)$, the proposition is true.

(2) Necessity. If edge $\overrightarrow{pq}$ 2 G, there is a monotonic path between p, q with only one edge. Otherwise, we can prove that there exists at least one monotonic path from p to q with mathematical induction. This problem can be converted to that there must be a path $v_1, v_2, ..., v_k, (k \quad n, v_1 = p)$ such that the path $v_1, ..., v_k$ is monotonic about q, which means 8i 2 f1, ..., $k \quad$ 1g, $\delta(v_i, q) > \delta(v_{i+1}, q)$.

When $k = 2$, because there is at least one edge $\overrightarrow{v_1 v_2}$ such that v_2 2 $B(q, \delta(v_1, q))$, we have $\delta(v_2, q) < \delta(v_1, q)$.Thus, the path v_1, v_2 is monotonic about q.

Suppose the proposition is true when $k = m, m < n$. When $k = m+1$, because there is at least one edge $v_m \overrightarrow{v_{m+1}}$ such that v_{m+1} 2 $B(v_m, \delta(v_m, q))$, we have $\delta(v_{m+1}, q) < \delta(v_m, q)$. Therefore, when $k = m + 1$, the path $v_1, ..., v_{m+1}$ is monotonic about q.

Here we have proved that the path $v_1, .., v_k, (p = v_1)$ is monotonic about q. Similar to the proof in Theorem 1, we can finally reach q for some $k \quad n$. Therefore, G is an MSNET. □

Based on the perspective of Lemma 1, we can calculate the length expectation of the monotonic path in the MSNETs as follows.

THEOREM 2. *Given a finite point set S of n points, uniformly distributed in a finite subspace in E^d. Suppose the volume of the minimal convex hull containing S is V_S. The maximal distance between any two points in S is R. We impose a constraint on V_S such that when d is fixed, 9$\kappa, \kappa V_S$ $V_B(R)$, where κ is a constant independent of n, and $V_B(R)$ is the volume of the sphere with radius R. We define 4r as 4r = min f$j\delta(a, b) \quad \delta(a, c)$j, $j\delta(a, b) \quad \delta(b, c)$j, $j\delta(a, c)$ $\delta(b, c)$jg, for all possible non-isosceles triangle abc on S. 4r is a decreasing function of n.*

For a given MSNET defined on such S, the length expectation of a monotonic path from p to q, for any p, q 2 S, is $O(n^{1/d} log(n^{1/d})/$ 4$r)$.

PROOF. Please see the detailed proof in the Appendix. □

Theorem 2 is a general property for all kinds of MSNETs. The function 4r has no definite expression about n because it involves randomness. We have observed that, in practice, 4r decreases very slowly as n increases. In the experiments, we estimate the function of 4r on different public datasets, based on the proposed graph in this paper. We find that 4r is mainly influenced by the data distribution and data density. The results are shown in the experiment section.

Because $O(n^{\frac{1}{d}})$ increases very slowly when n increases in high dimensional space, thus, the length expectation of the monotonic paths in an MSNET, $O(n^{1/d} log(n^{1/d})/$ 4$r)$, has a low growth rate which is very close to $O(\log n)$. This is also verified in our experiments. Meanwhile, we can see that the growth rate of the length expectation of the monotonic paths is relatively lower when d is higher.

In Theorem 2, the assumption on the volume of the minimal convex hull containing the data points is actually a constraint on the data distribution. We try to avoid the extremely special shape of the data distribution, which may influence the conclusion. For example, if the data points are all distributed uniformly on a straight line, the length expectation of the monotonic paths on such a dataset will grows almost linearly with n.

In addition, though we assume a uniform distribution of the data points, the property still holds to some extent on other various distributions in practice. Except for some extremely special shape of the data distribution, we can always expect that, when the search sphere shrinks at each step of the search path, the amount of the nodes left in the sphere decreases by some ratio. The ratio is mainly determined by the data distribution, as shown in Fig. 1.

Besides the length expectation of the search path, another important factor that influences the search complexity is the average out-degree of the graph. The degree of some MSNETs, like the Delaunay Graphs, grows when n increases [4]. Because there is no unified geometrical description of the MSNETs, therefore, there is no unified conclusion about how the degree of the MSNETs scales.

Dearholt *et al.* [11] claim they have found a way to construct an MSNET with a minimal degree. However, there are two problems with this method. Firstly, they did not provide analysis on how the degree of their MSNET scales with n. This is mainly because the MSNET they proposed is to add edges to RNG and it lacks geometrical description for degree analysis. Secondly, the proposed MSNET construction method is of very high time complexity (at least $O(n^{2-\frac{2}{1+d}+\epsilon} + n^2 \log n + n^3)$) and is not practical in real large-scale scenarios. Below we will propose a new type of MSNET with lower indexing complexity and constant out-degree expectation (independent of n). In other words, the search complexity on this graph scales with n in the same speed as the length expectation of the monotonic paths.

3.2 Monotonic Relative Neighborhood Graph

In this section, we will propose an MSNET that is sparse enough and easy to construct. Dearholt *et al.* [11] think the RNG is very close to the MSNET, but the RNG does not have sufficient edges to be an MSNET due to the strict edge selection strategy. Therefore, there is no theoretical guarantee on the search path length on an RNG, and the search on an RNG may suffer from long detours.

Here is an example. Let $lune_{pq}$ denote a region such that $lune_{pq} = B(p, \delta(p, q)) \setminus B(q, \delta(p, q))$ [21]. Given a finite point set S of n points in space E^d, for any two nodes p, q 2 S, edge pq 2 RNG if and only if $lune_{pq} \setminus S = ;$. In Fig. 2, (a)

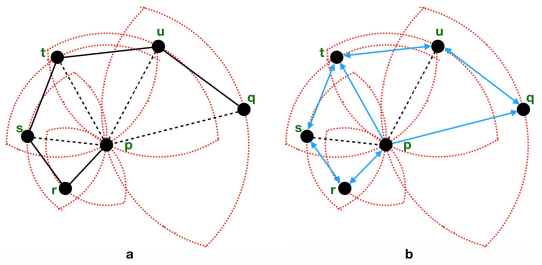


Figure 2: A comparison between the edge selection strategy of the RNG (a) and the MRNG (b). An RNG is an undirected graph, while an MRNG is a directed one. In (a), p and r are linked because there is no node in $lune_{pr}$. But $r \in lune_{ps}$, $s \in lune_{pt}$, $t \in lune_{pu}$, and $u \in lune_{pq}$. Therefore, there are no edges between p and s, t, u, q. In (b), p and r are linked because there is no node in $lune_{pr}$. p and s are not linked because $r \in lune_{ps}$ and $pr, sr \in MRNG$. Directed edge $\overrightarrow{pt} \in MRNG$ because $\overrightarrow{ps} \notin MRNG$. However, $\overrightarrow{tp} \notin MRNG$ because $\overrightarrow{ts} \in MRNG$. We can see that the MRNG is defined in a recursive way, and the edge selection strategy of the RNG is more strict than MRNG's. In the RNG(a), there is a monotonic path from p to q, but there is no monotonic path from q to p. In the MRNG(b), there is at least one monotonic path from any node to another node.

is an illustration of a non-monotonic path in an RNG. Node s is in $lune_{pr}$, so p, s is not connected. Similarly, t, u, q are not connected to p. When the search goes from p to q, the path is non-monotonic because $r, s, t \notin B(q, \delta(p, q))$.

We find that this problem is mainly due to RNG's strict edge selection strategy. Thus, instead of seeking the missing edges in the RNG with complicated operations as [11] does, we modify the edge selection strategy to enable monotonicity. The resulting graph may not be the minimal MSNET as [11] but is sparse enough. Based on the new strategy, we propose a novel graph structure called Monotonic Relative Neighborhood Graph (MRNG). Formally, an MRNG can be defined as following:

Definition 5 (MRNG). *Given a finite point set S of n points in space E^d, an MRNG is a directed graph with the set of edges satisfying the following property: for any edge $\overrightarrow{pq}$, $\overrightarrow{pq} \in MRNG$ if and only if $lune_{pq} \setminus S = \emptyset$ or $\forall r \in (lune_{pq} \setminus S), \overrightarrow{pr} \notin MRNG$.*

We avoid the ambiguousness in the following way when isosceles triangles appears. If $\delta(p, q) = \delta(p, r)$ and qr is the shortest edge in triangle pqr, we select the edge according to a predefined index, *i.e.*, we select $\overrightarrow{pq}$ if *index(q) < index(r)*. We can see that the MRNG is defined in a recursive way. In other words, Definition 5 implies that for any node p, we should select its neighbors from the closest to the farthest. The difference between MRNG's edge selection strategy and RNG's is that, for any edge $pq \in MRNG$, $lune_{pq} \setminus S$ is not necessarily the $\emptyset$. The difference can be seen in Fig. 2 clearly. Here we show that the MRNG is an MSNET.

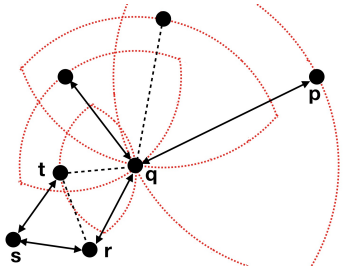


Figure 3: An illustration of the necessity that *NNG* $\subset$ *MRNG*. If not, the graph cannot be an MSNET. Path p, q, r, s, t is an example of non-monotonic path from p to t. In this graph, t is the nearest neighbor of q but not linked to q. We apply the MRNG's edge selection strategy on this graph. According to the definition of the strategy, t and r can never be linked. When the search goes from p to t, it must detour with at least one more step through s. This problem will be worse in practice.

Theorem 3. *Given a finite point set S of n points. An MRNG defined on S is an MSNET.*

Proof. Suppose $p, q \in S$ are any two points. If $\overrightarrow{pq} \in MRNG$, then there is a monotonic path from p to q and we are done. Otherwise, according to the definition of the MRNG, there must exist a node r such that $r \in lune_{pq}$, $\overrightarrow{pr} \in MRNG$. Therefore, we have $\delta(p, q) > \delta(r, q)$. Because for any two nodes p, q, there is at least one edge $\overrightarrow{pr}$ such that $r \in B(q, \delta(p, q))$ in an MRNG, we can conclude that MRNG is an MSNET by Lemma 1. □

Though different in the structure, MRNG and RNG share some common edges. First, we define the Nearest Neighbor Graph (NNG) as follows:

Definition 6 (NNG). *Given a finite point set S of n points in space E^d, an NNG is the set of edges such that, for any edge $\overrightarrow{pq}$, $\overrightarrow{pq} \in NNG$ if and only if q the closest neighbor of p in S.*

Similarly, we can remove the ambiguousness in the NNG by assigning a unique index for each node and linking the node to its nearest neighbor with the smallest index. Obviously, we have $MRNG \setminus RNG \subset NNG$ (if a node q is the nearest neighbor of p, we have $lune_{pq} \setminus S = \emptyset$). This is necessary for the MRNG's monotonicity. Fig. 3 shows an example of the non-monotonic path if we apply MRNG's edge selection strategy on some graph G but do not guarantee $NNG \subset G$. The edges in Fig. 3 satisfy the selection strategy of the MRNG except that q is forced not to be linked to its nearest neighbor t. Because t is the nearest neighbor of q, we have $\delta(q, r) > \delta(q, t)$. Because qt is the shortest edge in triangle qtr, and q, r is linked, rt must be the longest edge in triangle qtr according to the edge selection strategy of MRNG. Thus, r, t can not be linked and we can only reach t through other nodes (like s). Similarly, only when rt is the longest edge in triangle rst, edge rs and st can coexist in this graph. Therefore, when we go from p to t, we need a detour at least via node r, s. Because $\delta(q, r) > \delta(q, t)$, it is a non-monotonic path from p to t. If we don't guarantee $NNG \subset$

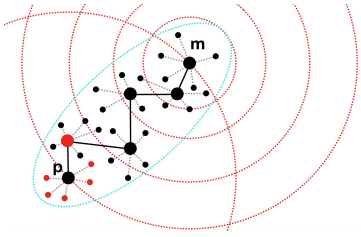


Figure 4: An illustration of the candidates of edge selection in NSG. Node p is the node to be processed, and m is the Navigating Node. The red nodes are the k nearest neighbors of node p. The big black nodes and the solid lines form a possible monotonic path contained in the candidates generated by the search-and-collect routine. The small black nodes are the nodes by the search-and-collect routine. All the nodes in the figure will be added to the candidate set of p.

MRNG, detours is unavoidable, which may be worse in practice. It's easy to verify that a similar detour problem will also appear if we perform RNG's edge selection strategy on G but do not guarantee that *NNG* G.

Here we will discuss the average out-degree of the MRNG. The MRNG has more edges than the RNG, but it's still very sparse because the angle between any two edges sharing the same node is at least 60° (by the definition of MRNG, for any two edges pq, pr *2 MRNG*, qr must be the longest edge in triangle pqr and qr *2 MRNG*).

LEMMA 2. *Given an MRNG in E^d, the max degree of the MRNG is a constant and independent of n.*

PROOF. Please see the detailed proof in the Appendix. □

Now according to Lemma 2, Theorem 1, Theorem 2, and Theorem 3, we have that the search complexity in expectation on an MRNG is $O(cn^{\frac{1}{d}} \log n^{\frac{1}{d}} / 4\,r)$, where c is the average degree of the MRNG and independent of n, $4\,r$ is a function of n, which decreases very slowly as n increases.

3.3 MRNG Construction

The MRNG can be constructed simply by applying our edge selection strategy on each node. Specifically, for each node p, we denote the set of rest nodes in S as $R = Snfpg$. We calculate the distance between each node in R and p, then rank them according to the distance in ascending order. We denote the selected node set as L. We add the closest node in R to L to ensure *NNG* *MRNG*. Next, we fetch a node q from R and a node r from L in order, to check whether pq is the longest edge in triangle pqr. If pq is not the longest edge in triangle $pqr, 8r\ 2\ L$, we add q to L. We repeat this process until all the nodes in R are checked. This naive construction runs in $O(n^2 \log n + n^2 c)$ time, where c is the average out-degree of MRNG, which is much lower than that of the MSNET indexing method proposed in [11], which is at least $O(n^{2-\frac{2}{1+d}+\epsilon} + n^2 \log n + n^3)$ under the general position assumption.

3.4 NSG: A Practical Approximation For MRNG

Algorithm 2 NSGbuild(G, l, m)

Require: kNN Graph G, candidate pool size l for greedy search, max-out-degree m.
Ensure: NSG with navigating node **n**
1: calculate the centroid **c** of the dataset.
2: **r** =random node.
3: **n** =Search-on-Graph(G,**r**,**c**,l)%navigating node
4: **for all** node **v** in G **do**
5: Search-on-Graph(G,**n**,**v**,l)
6: E =all the nodes checked along the search
7: add **v**'s nearest neighbors in G to E
8: sort E in the ascending order of the distance to **v**.
9: result set $R = \emptyset$, $\mathbf{p}_0$ = the closest node to **v** in E
10: R.add($\mathbf{p}_0$)
11: **while** !E.empty() && R.size() $< m$ **do**
12: **p** = E.front()
13: E.remove(E.front())
14: **for all** node **r** in R **do**
15: **if** edge **pv** occluded by edge **pr** **then**
16: break
17: **end if**
18: **end for**
19: **if** no occlusion occurs **then**
20: R.add(**p**)
21: **end if**
22: **end while**
23: **end for**
24: **while** True **do**
25: build a tree with edges in NSG from root **n** with depth-first-search.
26: **if** not all nodes linked to the tree **then**
27: add an edge between one of the out-of-tree nodes and its closest in-tree
28: neighbor (by algorithm 1).
29: **else**
30: break.
31: **end if**
32: **end while**

Though MRNG can guarantee a very low search time, the indexing time is still not practical for large-scale problems. In this section, we will present a practical and close approximation of the MRNG based on the four criteria to designed a good ANNS graph. We name it the **Navigating Spreading-out Graph** (NSG). We first present the NSG construction algorithm and then give the detailed analysis.

i We build an approximate kNN graph with the current state-of-the-art method *nn*-descent [12].

ii We find the approximate medoid of the dataset. This can be achieved by the following steps. (1) Calculate the centroid of the dataset; (2) Treat the centroid as the query, search on the kNN graph with Algorithm 1, and take the returned nearest neighbor as the approximate medoid. This node is named as the Navigating Node because all the search will start with this node.

iii we generate a candidate neighbor set for each node and select neighbors for them on the candidate sets. This can be achieved by the following steps. For a given node p, (1) we treat it as a query and perform Algorithm 1 on the prebuilt kNN graph with the Navigating Node as the starting node. (2) During the search, each visited node q (*i.e.*, the distance between p and q is calculated) will be added to the candidate set (the distance is also recorded). (3) Select at most m neighbors for p within the candidate set through the edge selection strategy of MRNG.

iv We span a Depth-First-Search tree on the produced graph of previous steps. We treat the Navigating Node as the root. When the DFS terminates, and there are nodes which are not linked to the tree, we link them to their approximate nearest neighbors (got by Algorithm 1) and continue the DFS. Please see Algorithm 2 for more details.

Following is the design intent of the NSG construction algorithm. The ultimate goal is to build an approximation of MRNG with low indexing time complexity.

(i) MRNG ensures there exists at least one monotonic path between any two nodes, however, it is not a easy task. Instead, we just pick one node out and try to guarantee the existence of monotonic paths from this node to all the others. We name this node as the Navigating Node. When we perform the search, we always start from the Navigating Node, which makes the search on an NSG almost as efficient as on an MRNG.

(ii) In the edge selection strategy of the MRNG, it treats all the other nodes as the candidate neighbors of the current node, which causes the high time complexity. To speed up this process, we want to generate a small subset of candidates for each node. These candidates contains two parts: (1) As discussed above, the NNG is essential for the monotonicity. Because it is very time-consuming to get the exact NNG, we turn to the approximate kNN graph. A high quality approximate kNN graph usually contains a high quality approximate NNG. It is acceptable when only a few nodes are not linked to their nearest neighbors. (2) Because the search on the NSG always starts from the Navigating Node p_n, for a given node p, we only need to consider those nodes which are on the search path from the p_n to p. Therefore, we treat p as the query and perform Algorithm 1 on the prebuilt kNN graph. The nodes visited by the search and p's nearest neighbor in the approximate NNG are recorded as candidates. The nodes forming the monotonic path from the Navigating Node to p are very likely included in the candidates. When we perform MRNG's edge selection strategy on these candidates, it's very likely that the NSG inherits the monotonic path in the MRNG from the Navigating Node to p.

(iii) A possible problem of above approach is the degree explosion for some nodes. Especially, the Navigating Node and nodes in the dense area will act as the "traffic hubs" and have high out-degree. This problem is also discussed in HNSW [29], and they introduced a multi-layer graph structure to solve it, which increased the memory use significantly. Our solution is to limit the out-degree of all the nodes to a small value m n by abandoning the longer edges. The consequence is the connectivity of the graph is no longer ensured due to the edge elimination.

To address the connectivity problem, we introduce a new method based on the DFS spanning tree as described above. After this process, all the nodes are guaranteed at least a path spread out from the Navigating Node. Though the proposed method will sacrifice a little theoretical performance, the detours in the NSG will be minimized if we build a high-quality approximate kNN graph and choose a proper degree limitation m.

By approximating the MRNG, the NSG can ensure the search paths almost as **short** as the MRNG. Meanwhile, the degree constraint makes the graph very **sparse**, and the tree-spanning operation guarantees the **connectivity** of the NSG. According to our empirical studies, the NSG can achieve out-standing search performance **without any extra structures** (*e.g.*, hashing tables of IEH, KD-trees of Efanna, and hierarchical graphs of HNSW). Adding such structures does not improve the performance but slow down the search. This implies that the NSG has maximized the potential of the graph structure. We can see NSG considers all the four aspects simultaneously. In our experiments, NSGs use the least memory among all the graph-based methods. The detailed results will be presented in the experimental section later.

3.4.1 Indexing Complexity of NSG

The total indexing complexity of the NSG contains two parts, the complexity of the kNN graph construction and the post-processing steps. In the implementation of this paper, we use the nn-descent algorithm to build the kNN graph. Its empirical complexity is $O(n^{1.16})$ [12].

The post-processes includes the search-collect-select operation and the tree spanning. Because the kNN graph is an approximation of the Delaunay Graph (an MSNET), the search complexity on it is approximately $O(kn^{\frac{1}{d}} \log n^{\frac{1}{d}} / \boldsymbol{4}\, r)$. We search for all the nodes, so the total complexity is about $O(kn^{\frac{1+d}{d}} \log n^{\frac{1}{d}} / \boldsymbol{4}\, r)$. The complexity of the edge selection is $O(nlc)$, where l is the number of the candidates generated by the search and c is the maximal degree we set for the graph. Because c and l are usually very small in practice (*i.e.*, c n, l n), this process is very fast. The final process is the tree spanning. This process is very fast because the amount of the strongly connected components is usually far less than n. We only need to add a small number of edges to the graph. We can see that the most time-consuming part is the "search-collect" part. Therefore, the total complexity of the post-processes is about $O(kn^{\frac{1+d}{d}} \log n^{\frac{1}{d}} / \boldsymbol{4}\, r)$, which is verified in our experimental evaluation in later sections. We also find that $\boldsymbol{4}\, r$ is almost a constant and does not influence the complexity in our experiments.

In the implementation of this paper, the overall empirical indexing complexity of the NSG is $O(kn^{\frac{1+d}{d}} \log n^{\frac{1}{d}} + n^{1.16})$, which is much lower than $O(n^2 \log n + cn^2)$ of the MRNG.

3.5 Search On NSG

We use Algorithm 1 for the search on the NSG, and we always start the search from the Navigating Node. Because the NSG is a carefully designed approximation of the MRNG, the search complexity on the NSG is approximately $O(cn^{\frac{1}{d}} \log n^{\frac{1}{d}} / \boldsymbol{4}\, r)$ on average, where c is the maximal degree of the NSG, and d is the dimension. In our experiments, $\boldsymbol{4}\, r$ is about $O(n^{-\frac{\epsilon}{d}})$, $0 < \epsilon$ d. So the empirical average search complexity is $O(cn^{\frac{1+\epsilon}{d}} \log n^{\frac{1}{d}})$. Because $1 + \epsilon$ d, the complexity is very close to $O(\log n)$, which is verified in our experimental evaluation in later sections. Our code has been released on GitHub[2].

3.6 The Relation Between MRNG, NSG And Previous Works

The **MRNG** improves the edge selection strategy of RNG from the ANNS perspective, which enables the monotonicity and ensures very low search complexity. The MRNG is more efficient than the traditional graphs (Delaunay Graphs [4],

[2]https://github.com/ZJULearning/nsg

Table 1: We list the information of most of the existing graph-based methods about whether they take actions in the four aspects that a good graph should consider. $\surd$ means the method takes actions in this aspect. × means no actions.

algorithm	Graph connectivity	Lower the degree	Shorten the paths	No extra structures
GNNS [18]	×	×	$\surd$	$\surd$
Efanna [13]	×	×	$\surd$	×
IEH [23]	×	×	$\surd$	×
FANNG [5]	×	$\surd$	×	$\surd$
NSW [28]	×	×	$\surd$	$\surd$
HNSW [29]	$\surd$	$\surd$	×	×
DPG [27]	×	$\surd$	×	$\surd$
MRNG	$\surd$	$\surd$	$\surd$	$\surd$
NSG	$\surd$	$\surd$	$\surd$	$\surd$

the minimal MSNET [11], the RNG [33], the NSWN [7], the Randomized Neighborhood Graph [3]) in ANNS problem. The reasons are as follows:

1. The RNG is not an MSNET, and the search on it needs back-tracing. Its search time complexity is uncertain.

2. The Delaunay Graph is an MSNET but almost fully connected in high dimensional space [5].

3. The Randomized Neighborhood Graph and the NSWN guarantee the length of the search path grows polylogarithmically with n (an empirical estimation), but they all suffer from degree explosion (the average degree tends to be very high) on large-scale datasets.

4. There is no practical method to build the Randomized Neighborhood Graph and the minimal MSNET.

The recent practical graph-based methods (GNNS, IEH, NSW, Efanna, HNSW, FANNG) are all approximations of above traditional graphs. The NSG approximates the MRNG, and it is the only one which considers all the four aspects as shown in Table 1.

4. EXPERIMENTS

In this section, we will give a detailed analysis of extensive experiments on public and synthetic datasets to demonstrate the effectiveness of our approach.

4.1 Datasets

The experiments are conducted on four datasets. SIFT1M and GIST1M are in the well-known BIGANN public ANNS dataset collection[3], which are widely used in the literature [5,22]. RAND4M and GAUSS5M are two synthetic datasets. RAND4M are vectors sampled from the uniform distribution of range (-1, 1). GAUSS5M are vectors sampled from a Gaussian distribution $N(0, 3)$. Considering that the data may lie in a low dimensional manifold, we measure the local intrinsic dimension (LID) [9] to reflect the datasets' degree of difficulty better. See Table.2 for more details.

4.2 Compared Algorithms

The algorithms we choose for comparison cover various types such as tree-based, hashing-based, quantization-based and graph-based approaches. The codes of most algorithms

[3]http://corpus-texmex.irisa.fr/

Table 2: Information on experimental datasets. D stands for dimension, and LID stands for local intrinsic dimension [9]. Base stands for the number of base vectors, and query stands for the number of query vectors.

dataset	D	LID	base	query
SIFT1M	128	9.3	1,000,000	10,000
GIST1M	960	18.9	1,000,000	1,000
RAND4M	128	62.1	4,000,000	10,000
GAUSS5M	128	12.3	5,000,000	10,000

are available and well optimized on the GitHub. For those who did not release their code, we implement their algorithm according to their papers with equal optimization. All of them are implemented in C++, compiled with g++4.9 and the same optimization option. The experiments of SIFT1M and GIST1M are carried out on a machine with i7-4790 CPU and 32G memory. The experiments of RAND4M and GAUSS5M are carried out on a machine with Xeon E5-2630 CPU and 96G memory.

Because not all algorithms support **inner-query parallelizing**, for all the search experiments, we only evaluate the algorithms on a single thread. Given that all the compared algorithms have the parallel versions for their index building algorithms, for time-saving, we construct all the indices with eight threads.

1. **Tree-Based Methods. Flann**[4] is a well-known ANNS library based on randomized KD-tree, K-means trees, and composite tree algorithm. We use its randomized KD-tree algorithm for comparison. **Annoy**[5] is based on a binary search forest.

2. **Hashing-Based Methods.** FALCONN[6] is a well-known ANNS library based on multi-probe locality sensitive hashing.

3. **Quantization-Based Methods. Faiss**[7] is recently released by Facebook. It contains well-implemented code for state-of-the-art product-quantization-based methods on both CPU and GPU. The CPU version is used here for a fair comparison.

4. **Graph-Based Methods. KGraph**[8] is based on a kNN Graph. **Efanna**[9] is based on a composite index of randomized KD-trees and a kNN graph. **FANNG** is base on a kind of graph structure proposed in [5]. They did not release their codes. Thus, we implement their algorithm according to their paper and with the same optimization as our codes. **HNSW**[10] is based on a hierarchical graph structure, which was proposed in [29]. **DPG**[11] is based on an undirected graph whose edges are selected from a kNN graph. According to an open source benchmark[12], **HNSW is the fastest ANNS algorithm on CPU before NSG.**

[4]https://github.com/mariusmuja/flann
[5]https://github.com/spotify/annoy
[6]https://github.com/FALCONN-LIB/FALCONN
[7]https://github.com/facebookresearch/faiss
[8]https://github.com/aaalgo/kgraph
[9]https://github.com/ZJULearning/efanna
[10]https://github.com/searchivarius/nmslib
[11]https://github.com/DBWangGroupUNSW/nns_benchmark
[12]https://github.com/erikbern/ann-benchmarks

5. **NSG** is the method proposed in this paper. It contains only one graph with a navigating node where the search always starts.

6. **NSG-Naive** is a designed baseline to demonstrate the necessity of NSG's search-collect-select operation and the guarantee of the graph connectivity. We directly perform the edge selection strategy of MRNG on the edges of the kNN graph to get NSG-Naive. There is no navigating node, and we use Algorithm 1 with random initialization on NSG-Naive.

4.3 Results

We randomly sample one percentage points out of each train set as the **validation sets**. Since it's essential to be fast in the high-precision region in real scenarios, we focus on the performance of all the algorithms in the high-precision area. We run all the algorithms on the four datasets, tune their indices on the validation set to get the best performance in the high-precision region. Then we record the time and the corresponding precision for each algorithm to get the curves of precision-against-queries-per-second. And the results are shown in **Fig. 5**.

The total memory use of different graph-based indices are reported in **Table.3**. Further, we investigate some other critical features of the graphs. We count how many edges of the NNG are included in a given graph (**NN-percentage**) for all the compared graph-based methods, which are also shown in **Table 3**. It's an indicator that whether the graph suffers from the detour problem discussed in Section. 3.2. Because HNSW contains multiple graphs in its index and the upper graphs do not cover all the points, we only report the average out-degree, maximum out-degree, and the NN-percentage of its bottom-layer graph, which contains all the points of the dataset.

Because the graph-based methods are far better than the others, the performance gaps amongst the algorithms make it difficult to draw all the curves smoothly within one figure. We use the discontinuous axis to show the results in **Fig. 5**. Many interesting points can be found as follows:

1. In **Fig. 5**, the NSG outperforms all the other state-of-the-art methods significantly on datasets of different scales and distributions. It's usually harder to search on datasets with higher local intrinsic dimension due to the "curse of the dimensionality". As the local intrinsic dimension increases, the performance gap between NSG and the other algorithms is widening. The superior performance of NSG owes to that NSG approximates the MRNG, which enables logarithmic search complexity. In other words, NSG considers the four aspects simultaneously during its construction.

2. Both HNSW and FANNG use RNG's edge selection strategy to lower the degree of the graph. However, they don't have any guarantee that their graphs contain the NNG. Specifically, both HNSW and FANNG build their graphs from a randomly linked graph then refine the graphs iteratively. Their indexing algorithms do not guarantee that the nearest neighbors will be linked to each other in the final graphs. This is verified in our experiments: a large proportion of nearest neighbors are not connected to each other (**Table 3**).

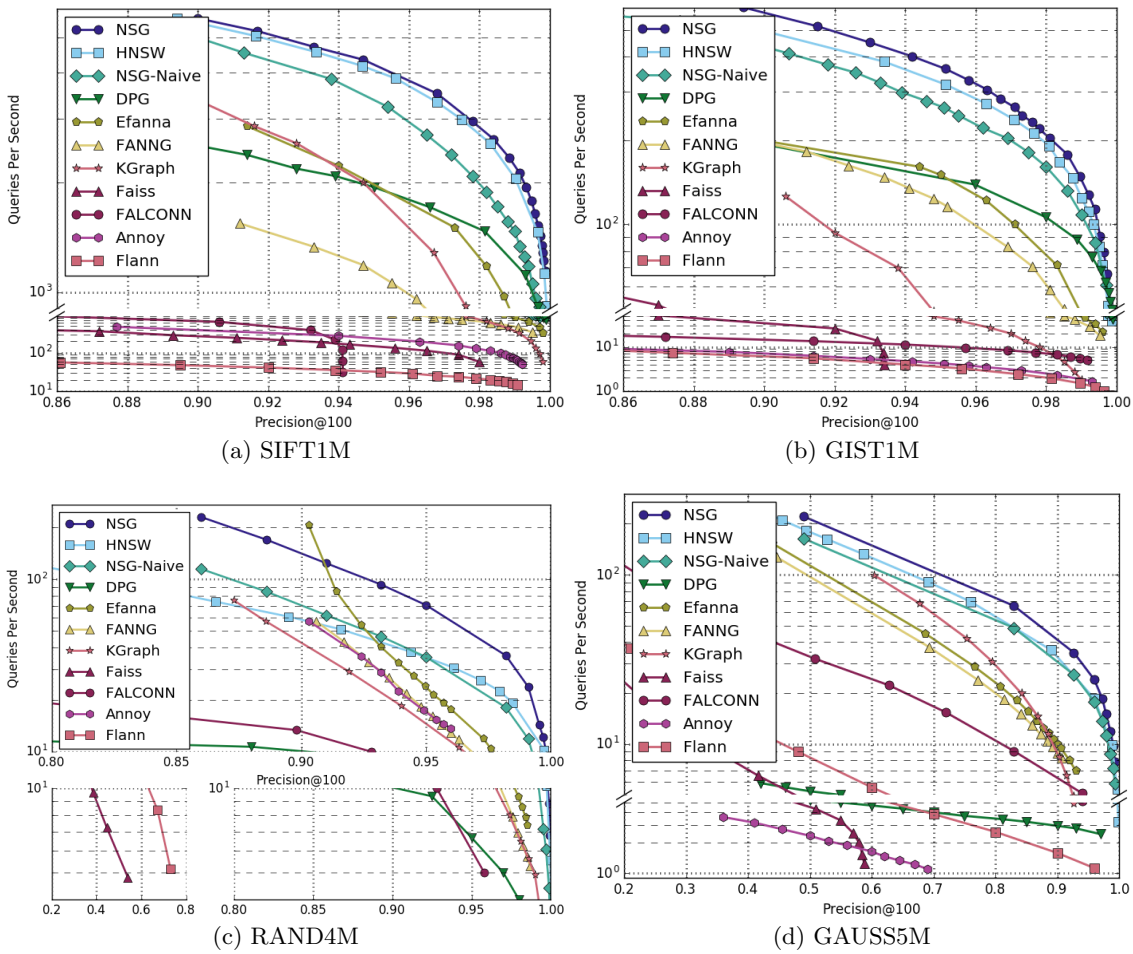
Table 3: Information of the graph-based indices involved in all of our experiments. AOD means the Average Out-Degree. MOD means the Maximum Out-Degree. The NN(%) means the percentage of the nodes which are linked to their nearest neighbor. Because HNSW contains multiple graphs, we only report the AOD, MOD, and NN(%) of its bottom-layer graph here.

dataset	algorithms	memory(MB)	AOD	MOD	NN(%)
SIFT1M	NSG	**153**	25.9	50	99.3
	HNSW	451	32.12	50	66.3
	FANNG	374	30.2	98	60.4
	Efanna	1403	300	300	99.4
	KGraph	1144	300	300	99.4
	DPG	632	165.08	1260	99.4
GIST1M	NSG	**267**	26.3	70	98.1
	HNSW	667	23.87	70	47.5
	FANNG	1526	29.2	400	39.9
	Efanna	2154	400	400	98.1
	KGraph	1526	400	400	98.1
	DPG	741	194.29	20899	98.1
RAND4M	NSG	$\mathbf{2.7 \times 10^3}$	174.011	220	96.4
	HNSW	6.7×10^3	160.995	220	76.5
	FANNG	5.0×10^3	181.176	327	66.7
	Efanna	6.3×10^3	400	400	96.6
	KGraph	6.1×10^3	400	400	96.6
	DPG	4.7×10^3	246.415	5309	96.6
GAUSS5M	NSG	$\mathbf{2.6 \times 10^3}$	146.223	220	94.3
	HNSW	6.7×10^3	131.857	220	57.6
	FANNG	5.2×10^3	152.16	433	53.4
	Efanna	7.8×10^3	400	400	94.3
	KGraph	7.6×10^3	400	400	94.3
	DPG	3.7×10^3	193.987	15504	94.3

The edge selection strategy of the RNG will result in a non-monotonic graph, thus, the search on the HNSW and FANNG will suffer from detours. Moreover, a large number of edges between the nearest neighbors are missing in the HNSW and FANNG, which will cause more severe detour problems as we have discussed in Section. 3.2 and shown in **Fig. 2, 3**. As a result, the search performance of FANNG is much worse than the NSG. HNSW is the second best-performing algorithm. It is because HNSW provides solutions to ensure the connectivity of the graph and enables fast short-cuts via multi-layer graphs. However, this results in very large index size.

3. NSG is the most memory-efficient algorithm among the graph-based methods (**Table 3**) since NSG best approximates the MRNG and limit the maximum out-degree to a small value. The traversing on the NSG is so fast that we do not need any extra indices to improve the search performance. It is important to note that, the memory occupations of NSG, HNSW, FANNG, Efanna's graph, and KGraph are all determined by the maximum out-degree. Although different nodes have different out-degrees, each node is allocated the same memory based on the maximum out-degree of the graphs to ensure the continuous memory access (for better search performance).

   The bottom layer of HNSW has the same maximum out-degree with that of NSG, HNSW requires larger memory because it has multiple graphs in the upper layers. DPG cannot use the continuous-memory-access technique since the maximum out-degree of DPG is too large. Considering both the search performance



(a) SIFT1M (b) GIST1M

(c) RAND4M (d) GAUSS5M

Figure 5: ANN search results of graph-based algorithms on the four datasets. All the algorithms use their best performing indices in high-precision region. (top right is better)

and the memory use, we can see that NSG outperform all the other graph-based methods significantly. Especially the index size of NSG is about 1/2-1/3 of the HNSW, which is the previous best performing algorithm[13].

4. The difference between NSG-Naive and NSG is that NSG-Naive does not follow the "search-collect-select" routine, doesn't select Navigating Node, and doesn't ensure the connectivity of the graph. As we discussed in Section. 3.4, there is no guarantee in the NSG-Naive of the existence of a path from the starting nodes to the targets. And the probability to reserve the monotonic paths is smaller due to the small neighborhood coverage. Though NSG-Naive uses the same edge selection strategy of MRNG, the degree of the approximation is inferior to NSG, which leads to inferior search performance.

5. DPG, KGraph, and Efanna suffer from poor search performance because the average out-degrees of them are much higher (**Table 3**). As we discussed above, the degree and the length of the search path should be minimized simultaneously to produce better efficiency. The kNN graph used in KGraph and Efanna is an approximation of the Delaunay Graph. When the k of the kNN graph is sufficiently large, the monotonicity may be best approximated, but the high degree damages the performance of KGraph and Efanna significantly. There is no evidence that DPG is monotonic, and the search complexity of DPG is unclear.The poor performance of the DPG may be due to the lack of theoretical support of such a design.

[13]https://github.com/erikbern/ann-benchmarks

4.4 The Search and Indexing Complexity of NSG

We estimate the search and indexing complexity of the NSG on SIFT1M and GIST1M. NSG's search complexity on SIFT1M is $O(n^{\frac{1}{9.3}} \log n^{\frac{1}{9.3}})$ for both 1-NN and 100-NN search. NSG's search complexity on GIST1M is about $O(n^{\frac{2}{18.9}} \log n^{\frac{1}{18.9}})$ for both 1-NN and 100-NN search. 9.3 and 18.9 are the local intrinsic dimension of SIFT1M and GIST1M respectively (Table 2), which agrees with our theoretical analysis. We can see that NSG is a good approximation of the MRNG.

We find that the $\boldsymbol{4}r$ on SIFT1M is almost a constant and has little impact on the search complexity, while the $\boldsymbol{4}r$ on

GIST1M is about $O(n^{-\frac{1}{18.9}})$. It decreases very slowly as n increases on GIST1M. This agrees with agrees with our theoretical analysis.

We also estimate how the search complexity scales with K, the number of neighbors required. It's about $O(K^{0.46})$ or $O((logK)^{2.7})$ on both datasets.

In our experimental study, the indexing complexity of NSG's post-processes is $O(n^{1+\frac{1}{9.3}} \log n^{\frac{1}{9.3}})$ on SIFT1M and $O(n^{1+\frac{1}{18.9}} \log n^{\frac{1}{18.9}})$ on GIST1M. The degree of the kNN graph is included in the constant factors. This agrees with our theoretical analysis.

Please see the Appendix for the figures and the detailed analysis.

4.5 Parameters

There are two parameters in the NSG indexing algorithm, l and m. In our experiments, we find that there exists an optimal parameter combination for given data distribution. The reason may be as follows: (1) l controls the search precision on the pre-built kNN graph. When the search result is accurate enough (*i.e.*, the generated candidates well cover the search path and the neighborhood of the current node), increasing l will not lead to better index. (2) m controls the max degree of the graph. There exists an optimal average out-degree for the NSG, which produces the best balance between the length of the search path and the degree, and leads to an optimal search complexity expectation.

Meanwhile, we find that the optimal parameter choice is insensitive to the data scale. This may be because if the feature vectors are produced by a fixed feature descriptor (*e.g.*, SIFT, GIST, and Deep Neural Networks), the data distribution tends to be fixed and independent of the scale. As a consequence, we can search for the best parameters on a small subset and apply to large datasets.

As for the search, the pool size l controls the search precision. The larger, the more accurate, but the slower. Please see our Github for the detailed parameters for each dataset.

4.6 Search In E-commercial Scenario With NSG

We have collaborated with Taobao on the billion-scale high-dimensional ANNS problem in the e-commercial scenario. In the e-commercial application, the response time of the search request is critical for the user experience. One search request from the user must be responded within 2 seconds, in which only 10 ms is available for the ANNS on the high dimensional vectors. We evaluate the performance of the NSG on some subsets of the e-commercial data (feature vectors of user and goods, produced by deep neural networks) and compare it with the original search engine (a well-optimized implementation of the IVFPQ algorithm [22]). The results are shown in **Table. 4**.

We compare NSG with the baseline on two datasets sampled from the same large-scale e-commerce database. The E10M dataset is used to test the performance on a single thread, while the E45M dataset is used to test the **Distributed Search** performance in a simulation environment. Search in such an environment involves data partition, local search, communication among computing nodes, and result merging. In our experiments, we randomly partition the dataset evenly into 12 subsets and build 12 NSGs, one for each subset. For each query, we search on 12 subgraphs in parallel and merge the results. We can see that our algorithm outperforms the baseline on both settings significantly.

Table 4: Results on the e-commerce dataset. E10M has 10 million vectors, and E45M has 45 million vectors. The dimension is 128. NT is the number of the threads. SQR98 means Single-Query-Response time to retrieve 100 neighbors at 98% precision. QB is the quantization-based baseline.

data set	algorithm	NT	SQR98 (ms)
E10M	NSG	1	2.3
E10M	IVFPQ	1	10
E45M	NSG	12	1
E45M	IVFPQ	12	10

In practice, the demand is much more complicated. There are billions of merchandise in the online platform of Taobao. Hundreds of millions of them are hot items which are often retrieved. The rest are long-tail items which are seldom retrieved. Moreover, the search system needs to add millions of new items and remove millions of outdated items every day. Although the NSG is very efficient in search, the indexing of it is slower than some algorithms like LSH or IVFPQ. To meet the demand of fast indexing, we don't build an NSG on the whole dataset. Instead, we take advantage of the distributed search engine of the platform and partition the dataset into several subsets on different computation node. To reduce the waiting time of the search request of hot items, we duplicate the indices on different computation node. We can update the NSG index by treating the new item as a query and search on the NSG. Then we use the same "search-collect-select" procedure described in Section 3.4 to link the new point into the NSG. For deletion, we adopt a lazy strategy, which is marking the deleted node and avoid returning them as the results. When a deleted node is included in the result, we continue the search until we find another non-deleted node. When the number of deleted nodes is so large that affects the response time, we rebuild the NSG. The indexing and updating are all controlled to be completed within a few hours by limiting the size of the subsets. When a subset grows too large, we repartition it into smaller one. The NSG is now integrated into the search engine and the response time of one ANNS request is now controlled to be within 5 ms.

5. CONCLUSIONS

In this paper, we present a new monotonic search network MRNG, which ensures approximately logarithmic search complexity. We propose four aspects (ensuring the connectivity, lowering the average out-degree, shortening the search paths, and avoiding additional index structures) for an ideal graph structure. Based on the four aspects, we propose NSG, which is a practical approximation of the MRNG and considers the four aspects simultaneously. Extensive experiments show the NSG outperforms the other state-of-the-art algorithms significantly over the graph-based approaches. Moreover, the NSG outperforms the baseline method of Taobao (Alibaba Group) and has been integrated into their search engine for billion-scale search.

6. REFERENCES

[1] F. André, A.-M. Kermarrec, and N. Le Scouarnec. Cache locality is not enough: high-performance

nearest neighbor search with product quantization fast scan. *Proceedings of the VLDB Endowment*, 9(4):288–299, 2015.

[2] A. Arora, S. Sinha, P. Kumar, and A. Bhattacharya. Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. *PVLDB*, 11(8):906–919, 2018.

[3] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, pages 271–280, 1993.

[4] F. Aurenhammer. Voronoi diagramsa survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[5] H. Ben and D. Tom. FANNG: Fast approximate nearest neighbour graphs. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, pages 5713–5722, 2016.

[6] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[7] M. Boguna, D. Krioukov, and K. C. Claffy. Navigability of complex networks. *Nature Physics*, 5(1):74–80, 2009.

[8] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2005.

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

[10] A. P. de Vries, N. Mamoulis, N. Nes, and M. Kersten. Efficient k-nn search on vertically decomposed data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 322–333. ACM, 2002.

[11] D. Dearholt, N. Gonzales, and G. Kurup. Monotonic search networks for computer vision databases. In *Signals, Systems and Computers, 1988. Twenty-Second Asilomar Conference on*, volume 2, pages 548–553. IEEE, 1988.

[12] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international Conference on World Wide Web*, pages 577–586, 2011.

[13] C. Fu and D. Cai. Efanna : An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv:1609.07228*, 2016.

[14] K. Fukunaga and P. M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computers*, 100(7):750–753, 1975.

[15] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi. Dsh: data sensitive hashing for high-dimensional k-nnsearch. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1127–1138. ACM, 2014.

[16] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2946–2953. IEEE, 2013.

[17] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, 1999.

[18] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *IJCAI 2011, Proceedings of the International Joint Conference on Artificial Intelligence*, volume 22, pages 1312–1317, 2011.

[19] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *PVLDB*, 9(1):1–12, 2015.

[20] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems (TODS)*, 30(2):364–397, 2005.

[21] J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517, 1992.

[22] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.

[23] Z. Jin, D. Zhang, Y. Hu, S. Lin, D. Cai, and X. He. Fast and accurate hashing via iterative nearest neighbors expansion. *IEEE transactions on cybernetics*, 44(11):2167–2177, 2014.

[24] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv:1702.08734*, 2017.

[25] J. M. Kleinberg. Navigation in a small world. *Nature*, 406(6798):845–845, 2000.

[26] G. D. Kurup. Database organized on the basis of similarities with applications in computer vision. *doctoral thesis*, 1992.

[27] W. Li, Y. Zhang, Y. Sun, W. Wang, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement (v1. 0). *arXiv:1610.02455*, 2016.

[28] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.

[29] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *arXiv:1603.09320*, 2016.

[30] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[31] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[32] G. Teodoro, E. Valle, N. Mariano, R. Torres, W. Meira, and J. H. Saltz. Approximate similarity search for online multimedia services on distributed cpu–gpu platforms. *The VLDB Journal*,

23(3):427–448, 2014.

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

[34] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in neural information processing systems*, pages 1753–1760, 2009.

[35] Y. Wu, R. Jin, and X. Zhang. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1139–1150. ACM, 2014.

[36] A. C.-C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

[37] Y. Zheng, Q. Guo, A. K. Tung, and S. Wu. Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2023–2037. ACM, 2016.

APPENDIX

A. PROOF FOR THEOREM 2

PROOF. (1) When p and q are linked in the MSNET, the path length between p and q is 1.

(2) When p and q are not linked in the MSNET, we can select a monotonic path in the MSNET, denoted as $p = v_1, v_2, ..., v_{k-1}, q$. Thus, we have a monotonically decreasing sequence $Z = \{\delta(v_1, q), \delta(v_2, q), ..., \delta(v_{k-1}, q)\}$. Because v_{k-1} may not be the nearest neighbor of q, we add a element $\delta(v_k, q)$ to the end of the sequence, where v_k is the nearest neighbor of q and $\delta(v_{k-1}, q) \quad \delta(v_k, q)$. The length of the new sequence still corresponds strictly to the length of the path (always longer by 1 for all possible paths), so it won't affect the correctness of the conclusion. We denote the new sequence as $Z = \{\delta(v_1, q), \delta(v_2, q), ..., \delta(v_k, q)\}$.

We can use the elements in Z to construct a sequence of concentric open spheres $\{B(q, \delta(v_1, q)), ..., B(q, \delta(v_k, q))\}$. Let $V_B(B(q, \gamma))$ denote the volume of the sphere $B(q, \gamma)$. Let

$$\eta_i = \frac{V_B(B(q, \delta(v_{i+1}, q)))}{V_B(B(q, \delta(v_i, q)))}, i \in \{1, .., k \quad 2\}$$
$$\eta_i = (\frac{\delta(v_{i+1}, q)}{\delta(v_i, q)})^d, i \in \{1, .., k \quad 2\}$$

Let R be the maximal distance between any two points in S. We have $\forall i \in \{1, .., k\}, \delta(v_i, q) \quad R$.

Suppose a, b, c are any three points in S such that triangle abc is not an isosceles triangle. Let $4r = \min\{j\delta(a, b) \quad \delta(a, c)j, j\delta(a, b) \quad \delta(b, c)j, j\delta(a, c) \quad \delta(b, c)j\}$. We have $\forall i \in \{1, .., k \quad 2\}, j\delta(v_i, q) \quad \delta(v_{i+1}, q)j \quad 4r$.

Therefore, we have

$$\eta_i = (\frac{\delta(v_{i+1}, q)}{\delta(v_i, q)})^d \quad (\frac{R \quad 4r}{R})^d, \forall i \in \{1, .., k \quad 2\}$$
$$\eta_{k-1} = (\frac{\delta(v_k, q)}{\delta(v_{k-1}, q)})^d \quad 1$$

and

$$V_{B_1} \quad V_B(B(q, R)).$$

Let $V_{B_i} = V_B(B(q, \delta(v_i, q)))$. We have

$$V_{B_k} = V_{B_1} \frac{V_{B_2}}{V_{B_1}} ... \frac{V_{B_k}}{V_{B_{k-1}}}$$
$$= V_{B_1} \quad \eta_1 ... \eta_{k-1}$$

Because $V_B(B(q, \delta(v_1, q))) \quad V_B(B(q, R))$, we have

$$V_{B_k} \quad V_B(B(q, R)) \quad \eta_1 ... \eta_{k-2}\eta_{k-1}$$
$$V_B(B(q, R)) \quad ((\frac{R \quad 4r}{R})^d)^{k-2}$$
$$\frac{V_{B_k}}{V_B(B(q, R))} \quad ((\frac{R \quad 4r}{R})^d)^{k-2}$$

Let $\hat{\eta} = (\frac{R - \triangle r}{R})^d$, we have

$$k \quad 2 \quad \log_{\hat{\eta}} \frac{V_{B_k}}{V_B(B(q, R))}$$
$$= \log_{\hat{\eta}} (\frac{\delta(v_k, q)}{R})^d$$
$$= d \log_{\hat{\eta}} \delta(v_k, q) \quad d \log_{\hat{\eta}} R$$

We calculate the expectation of above inequality and have

$$\mathbb{E}(k) \quad 2 \quad d\mathbb{E}(\log_{\hat{\eta}}(\delta(v_k, q))) \quad d\mathbb{E}(\log_{\hat{\eta}}(R))$$

Because R and $4r$ are constants (independent of the choice of p, q) when the point set S and the MSNET are determined, we have $\hat{\eta} = (\frac{R - \triangle r}{R})^d$ to be independent of the choice of p, q. Therefore, we have

$$\mathbb{E}(k) \quad 2 \quad d\mathbb{E}(\log_{\hat{\eta}}(\delta(v_k, q))) \quad d\mathbb{E}(\log_{\hat{\eta}}(R))$$
$$= \frac{d\mathbb{E}(\log \delta(v_k, q))}{\log \hat{\eta}} \quad \frac{d \log R}{\log \hat{\eta}}$$

Because $\log \mathbb{E}(x) \quad \mathbb{E}(\log x)$, we have

$$\mathbb{E}(k) \quad 2 \quad \frac{d \log \mathbb{E}(\delta(v_k, q))}{\log \hat{\eta}} \quad \frac{d \log R}{\log \hat{\eta}}$$
$$= \frac{d \log \mathbb{E}(\delta(v_k, q)) \quad d \log R}{d \log(R \quad 4r) \quad d \log R}$$
$$= \frac{\log \mathbb{E}(\delta(v_k, q)) \quad \log R}{\log(R \quad 4r) \quad \log R}$$

Let

$$f(R) = \frac{\log \mathbb{E}(\delta(v_k, q))) \quad \log R}{\log(R \quad 4r) \quad \log R}$$

Because $\mathbb{E}(\delta(v_k, q)))$ and $4r$ is independent of R, we can calculate the derivative of $f(R)$ about R and get

$$f'(R) = \frac{\frac{1}{R}(\log R \quad \log(R \quad 4r))}{(\log(R \quad 4r) \quad \log R)^2}$$
$$+ \frac{(\log R \quad \log(\mathbb{E}(\delta(v_k, q))))(\frac{1}{R - \triangle r} \quad \frac{1}{R})}{(\log(R \quad 4r) \quad \log R)^2}$$

By the definition of $4r$ and R, we have $4r < R$. Because $\delta(v_k, q)$ is the distance between q and v_k, which is the nearest neighbor of q in S, we have $\forall q \in S, R \quad \delta(v_k, q)$. Further, $R \quad \mathbb{E}(\delta(v_k, q))$. Therefore, we have $\forall R > 0, f'(R) > 0$.

We define the density of the data points to be the number of the points in a unit volume. Because the data points are uniformly distributed in the space, the density A of the data points is a constant. We have $n = A \quad V_S$. Because $\kappa V_S \quad V_B(B(q, R))$, we have

$$V_B(B(q, R)) \quad \kappa V_S$$
$$V_B(B(q, R)) \quad \kappa \frac{n}{A}$$
$$C_B R^d \quad \kappa \frac{n}{A}$$
$$R \quad (\frac{\kappa n}{A C_B})^{\frac{1}{d}},$$

where C_B is a constant that is only related to d. For convenience, we merge the constant factors and rewrite above inequality as $R \quad (\frac{n}{C})^{\frac{1}{d}}$. Because $\forall R > 0, f'(R) > 0$, we have $f(R) \quad f((\frac{n}{C})^{\frac{1}{d}})$, *i.e.*, we have

$$\mathbb{E}(k) \quad 2 \quad f(R) \quad f((\frac{n}{C})^{\frac{1}{d}})$$
$$= \frac{\log \mathbb{E}(\delta(v_k, q)) \quad \log(\frac{n}{C})^{\frac{1}{d}}}{\log((\frac{n}{C})^{\frac{1}{d}} \quad 4r) \quad \log(\frac{n}{C})^{\frac{1}{d}}}$$

Now we are going to evaluate growth rate of the right side of the above inequality about n.

Because $\delta(v_k, q)$ is the distance between q and its nearest neighbor. The expectation $\mathbb{E}(\delta(v_k, q))$ over all possible q is

only related to the density of the data points, thus, it's a constant which is independent of n. Let $l = \log \mathbb{E}(\delta(v_k, q))$ and $r(n) = (\frac{n}{C})^{\frac{1}{d}}$, then we have

$$\mathbb{E}(k) \quad 2 \quad \frac{l \quad \log r(n)}{\log(r(n) \quad 4\,r) \quad \log r(n)}$$

Let

$$f_1(n) = \frac{l \quad \log r(n)}{\log(r(n) \quad 4\,r) \quad \log r(n)}$$
$$f_2(n) = \frac{r(n) \log r(n)}{4\,r}$$

We are going to prove that $f_1(n)$ and $f_2(n)$ are of the same order of growth.

Suppose $S_n = \{p_1, ..., p_n\}$. By the definition of $4\,r$, we can find that, when n increases by 1, there will be a new point p_{n+1} added to S. And there will be C_n^2 triangles that can be formed by $p_i, p_j \in S_n$ and p_{n+1}. Amongst the C_n^2 triangles, it's possible to obtain a $4\,r'$ which is smaller than the $4\,r$ of the original point set S_n. Otherwise, the $4\,r$ on the new set S_{n+1} will remain unchanged. Thus, $4\,r$ is a monotonically decreasing function of n, and when $n > 1$, $4\,r > 0$. Because $r(n) = (\frac{n}{C})^{\frac{1}{d}}$, we have $r(n) > 1$ when $n > 1$.

Because n is discontinuous ($n \in \mathbb{N}^+$), for the convenience of the discussion of limit, we assume n is a continuous variable. And we assume $r(n)$ and $4\,r$ are continuously differentiable about n, which won't influence the correctness of the results.

We have

$$\frac{f_1(n)}{f_2(n)} = \frac{\frac{l - \log r(n)}{\log(r(n) - \triangle r) - \log r(n)}}{\frac{r(n) \log r(n)}{\triangle r}} = \frac{\frac{l}{\log r(n)} \quad 1}{\frac{r(n)(\log(r(n) - \triangle r) - \log r(n))}{\triangle r}}$$

Because when $n > 1$, we have

$$\frac{l}{\log r(n)} \quad 1 > \quad 1$$

Let

$$f_3(n) = \frac{r(n)(\log(r(n) \quad 4\,r) \quad \log r(n))}{4\,r}$$

Because $\lim_{n\to\infty} f_3(n)$ is not obvious, we rewrite $f_3(n)$ as follows:

$$f_3(n) = \frac{\log(r(n) \quad 4\,r) \quad \log r(n)}{\frac{\triangle r}{r(n)}}$$

Let

$$f_4(n) = \log(r(n) \quad 4\,r) \quad \log r(n),$$
$$f_5(n) = \frac{4\,r}{r(n)}$$

We have $\lim_{n\to\infty} f_4(n) = 0$ and $\lim_{n\to\infty} f_5(n) = 0$. According to L'Hôpital's rule, we have

$$\begin{aligned}
\lim_{n\to\infty} f_3(n) &= \lim_{n\to\infty} \frac{f_4(n)}{f_5(n)} = \lim_{n\to\infty} \frac{f_4'(n)}{f_5'(n)} \\
&= \lim_{n\to\infty} \frac{\frac{r'(n) - \triangle r'}{r(n) - \triangle r} \quad \frac{r'(n)}{r(n)}}{\frac{r(n)\triangle r' - r'(n)\triangle r}{r^2(n)}} \\
&= \lim_{n\to\infty} \frac{\frac{r'(n)\triangle r - r(n)\triangle r'}{(r(n) - \triangle r)r(n)}}{\frac{r(n)\triangle r' - r'(n)\triangle r}{r^2(n)}} \\
&= \lim_{n\to\infty} \frac{r^2(n)}{(r(n) \quad 4\,r)r(n)} \\
&= \lim_{n\to\infty} \frac{1}{1 \quad \frac{\triangle r}{r(n)}} \\
&= \quad 1
\end{aligned}$$

Further, we have

$$\begin{aligned}
\lim_{n\to\infty} \frac{f_1(n)}{f_2(n)} &= \lim_{n\to\infty} \frac{\frac{l}{\log r(n)} \quad 1}{f_3(n)} \\
&= \frac{1}{1} \\
&= 1
\end{aligned}$$

Therefore, we have proved that $f_1(n)$ and $f_2(n)$ are of the same order of growth. We have

$$\begin{aligned}
\mathbb{E}(k) \quad 2 &= O(f_2(n)) \\
&= O(\frac{n^{\frac{1}{d}} \log n^{\frac{1}{d}}}{4\,r})
\end{aligned}$$

We have the expectation length of a monotonic path in the given MSNET is $\mathbb{E}(length_{path}) = \mathbb{E}(k \quad 1) = O(n^{\frac{1}{d}} \log n^{\frac{1}{d}} / 4\,r)$.

□

B. PROOF FOR LEMMA 2

PROOF. Yao [36] prove that, for any $0 < \varphi < \pi$, one can cover the space E^d with finite convex cones sharing the same apex such that the angular diameter of each cone is smaller than φ. Let $\mathbb{C} = \{C_1, C_2, ..., C_k\}$ be such a set of convex cones constructed by the algorithm proposed by Yao. We have $sup\{\Theta(C_i) | C_i \in \mathbb{C}\} < \varphi$, where $\Theta(C)$ denotes the angular diameter of convex cone C. Let u be the apex of all the cones and ua be a ray. We define two convex cones C_i, C_j to be *adjacent* if $\exists ua, ua \in (C_i \setminus C_j)$.

Let $\{uv_1, uv_2, ..., uv_{k+1}\}$ be any $k+1$ rays with the same initial point u. Because the k convex cones in $\mathbb{C}$ cover the whole E^d space, according to the Pigeonhole Principle, we have that there are at least two rays are in the same cone. Let uv and uw be any two rays with the same initial point. Let $\angle vuw$ be the angle between uv and uw. Because $sup\{\Theta(C_i) | C_i \in \mathbb{C}\} < \varphi$, we have, if uv and uw are inside of the same cone, then $\angle vuw \quad \varphi$. Let $\varphi < 60^\circ$ and we construct a finite set of cones covering the whole E^d. Suppose the number of the cones is K_d. If we want the angle between any two rays sharing the same initial point to be no smaller than 60°, we have that the number of the rays must be smaller than $K_d + 1$, where K_d is only related to d.

Because S is a finite point set, the direction to which the out-edges of each node pointing is discontinuous. Because

the angles between any two out-edges of a given node is no smaller than 60° in the MRNG, the max out-degree of each node is much smaller than $K_d + 1$. Let C_d be the maximal value of the average out-degree of any MRNG, we have $C_d < K_d + 1$. C_d is only related to the dimension d and independent of n. $\square$

C. EXPERIMENTS ON SEARCH AND INDEXING COMPLEXITY

We estimate the search and indexing complexity of the NSG on SIFT1M and GIST1M. NSG's search complexity on SIFT1M is $O(n^{\frac{1}{9.3}} \log n^{\frac{1}{9.3}})$ for both 1-NN (**Fig. 6**) and 100-NN search (**Fig. 7**). NSG's search complexity on GIST1M is $O(n^{\frac{2}{18.9}} \log n^{\frac{1}{18.9}})$ for both 1-NN (**Fig. 6**) and 100-NN search (**Fig. 7**). 9.3 and 18.9 is the local intrinsic dimension of SIFT1M and GIST1M respectively (in Table 2 of the paper). The complexity of the search on the MRNG is $O(cn^{\frac{1}{d}} \log n^{\frac{1}{d}} / \boldsymbol{4}\, r)$, where c is the average degree of the MRNG.

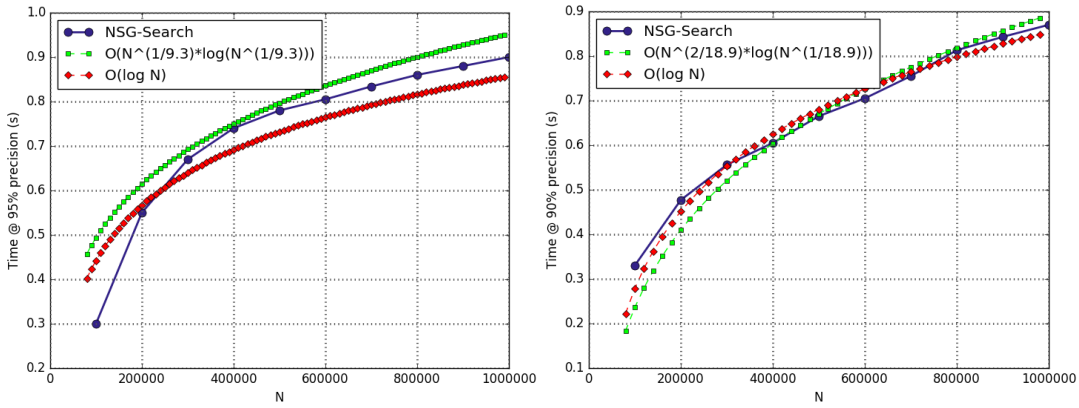
According to the results, we find that the $\boldsymbol{4}\, r$ on SIFT1M is almost a constant and has little impact on the search complexity. The $\boldsymbol{4}\, r$ on GIST1M is about $O(n^{-\frac{1}{18.9}})$. This may be mainly due to the difference in their data distribution. The numerical values in each dimension of SIFT vectors are integers ranging from 0 to 255, while the numerical values in each dimension of GIST vectors are real numbers ranging from 0 to 1.5. The density of the points in GIST1M is much larger than in SIFT1M. Nevertheless, $O(n^{-\frac{1}{18.9}})$ decreases very slow when n increases on GIST1M. The search complexity increases very slowly with n. These results agree with our theoretical analysis, and we can see that NSG is a good approximation of the MRNG.

We also estimate how the search complexity scales with K, the number of neighbors required. Because there is no theoretical analysis on how the search complexity may scale with K. We try to fit the figure with $O(K^x)$ and $O(\log^x K)$, and we find that $O(K^{0.46})$ or $O((logK)^{2.7})$ all fit the curves well on both datasets (**Fig. 8**).

The indexing of the NSG includes the approximate kNN graph construction, the "search-collect-select" procedure and the tree spanning procedure. The approximate kNN graph construction method is replaceable when there is a more efficient approach. In this paper, we use the *nn*-descent algorithm. Its empirical complexity is $O(n^{1.16})$ [12]. The tree spanning procedure is usually very fast and is of about $O(n)$ complexity because it is simply a DFS on the graph and it is rare that a node is not connected to the tree unless the data points are highly clustered. Usually, we just scan the dataset one time when spanning the tree. The reconnecting operation is seldom needed.

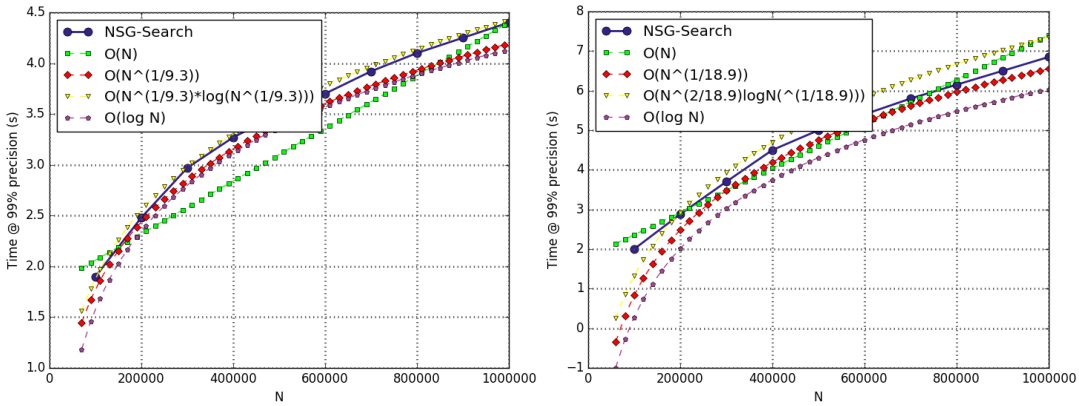
The "search-collect-select" is the most time-consuming part. We first search on the prebuilt kNN graph, which is an approximation of the Delaunay Graph. The Delaunay Graph is an MSNET. The search on the kNN graph can be estimated approximately through the search complexity on an MSNET. Because we search for all the nodes in the dataset and select among the nodes on the search path, the complexity of the "search-collect-select" should be about $O(\epsilon n^{1+\frac{1}{d}} \log n^{\frac{1}{d}})$ in total, where ϵ is some constant.

The experimental results are in **Fig. 9**. We can see that $O(\epsilon n^{1+\frac{1}{d}} \log n^{\frac{1}{d}})$ fit the indexing time curves well on both datasets, which agrees with our theoretical analysis.



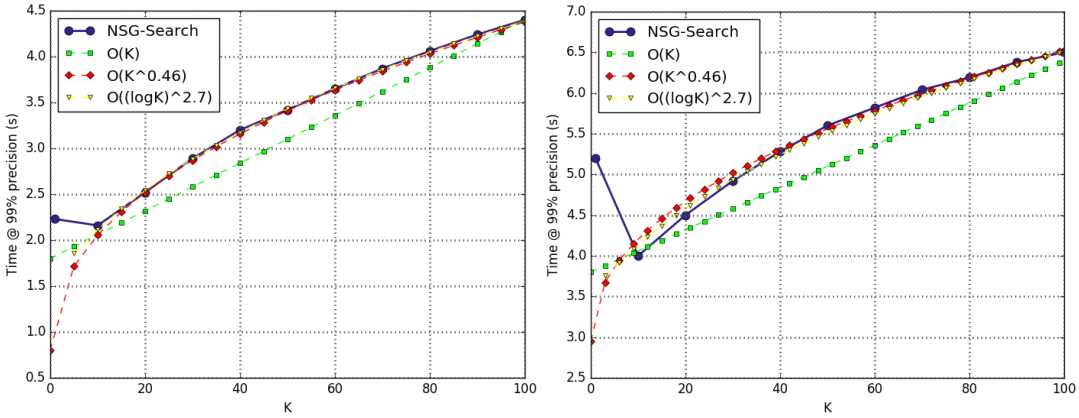
(a) SIFT1M (b) GIST1M

Figure 6: The experiments of how the 1-NN search time scales with the data size on SIFT1M and GIST1M. The search time is recorded at 95% precision and 90% precision on them respectively. The complexity of 1-NN search on NSG is about $O(N^{\frac{1}{9.3}} \log N^{\frac{1}{9.3}})$ on SIFT1M, and is a little higher than $O(\log N)$. 9.3 is the intrinsic dimension of SIFT1M. The complexity of 1-NN search on NSG is about $O(N^{\frac{2}{18.9}} \log N^{\frac{1}{18.9}})$ on GIST1M, and is also a little higher than $O(\log N)$. 18.9 is the intrinsic dimension of GIST1M.



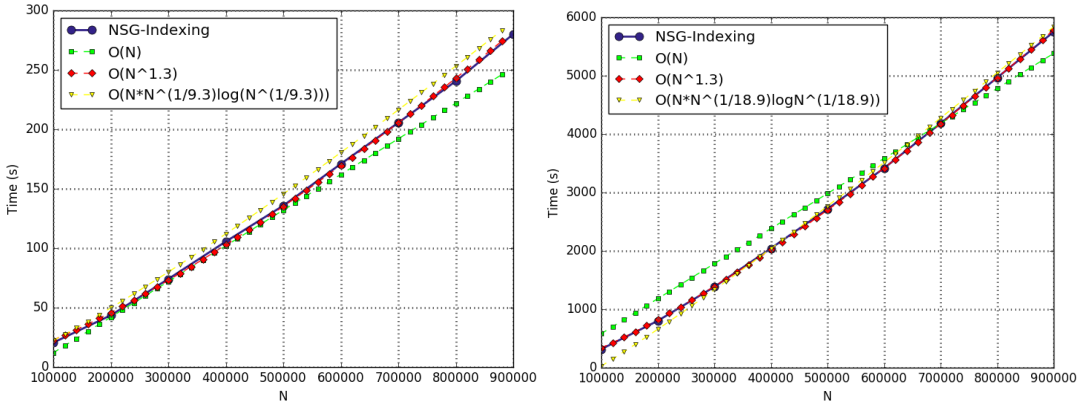
(a) SIFT1M (b) GIST1M

Figure 7: The experiments of how the 100-NN search time scales with the data size on SIFT1M and GIST1M. The search time is recorded at 99% precision. The complexity of 100-NN search on NSG is about $O(N^{\frac{1}{9.3}} \log N^{\frac{1}{9.3}})$ on SIFT1M, and is a little higher than $O(\log N)$ and $O(N^{\frac{1}{9.3}})$. 9.3 is the intrinsic dimension of SIFT1M. The complexity of 100-NN search on NSG is about $O(N^{\frac{2}{18.9}} \log N^{\frac{1}{18.9}})$ on GIST1M, and is a little higher than $O(\log N)$ and $O(N^{\frac{1}{18.9}})$. 18.9 is the intrinsic dimension of GIST1M.



(a) SIFT1M (b) GIST1M

Figure 8: The experiments of how the K-NN search time scales with K on SIFT1M and GIST1M. The search time is recorded at 99% precision. The complexity of K-NN search on NSG is about $O(K^{0.46})$ or $O(\log^{2.7} K)$ on both datasets.



(a) SIFT1M (b) GIST1M

Figure 9: Indexing time complexity estimation of the NSG on SIFT1M and GIST1M datasets. N is the amount of the data points. The reported time is the processing time of the later steps of the NSG, including the "search-collect-select" procedure and the tree spanning procedure. The indexing complexity of NSG is about $O(N^{1+\frac{1}{9.3}} \log N^{\frac{1}{9.3}})$ on SIFT1M, and is close to $O(N^{1.3})$. 9.3 is the intrinsic dimension of SIFT1M. The indexing complexity of NSG is about $O(N^{1+\frac{1}{18.9}} \log N^{\frac{1}{18.9}})$ on GIST1M, and is also close to $O(N^{1.3})$. 18.9 is the intrinsic dimension of GIST1M.