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

Cong Fu Changxu Wang Deng Cai
The State Key Lab of CAD&CG, College of Computer Science, Zhejiang University
Alibaba-Zhejiang University Joint Institute of Frontier Technologies
Hangzhou, China
{fc731097343,changxu.mail,dengcai}@gmail.com

ABSTRACT

The approximate nearest neighbor search (ANNS) is a fundamental problem in machine learning and data mining. An ANNS algorithm is required to be efficient on both memory use and search performance. Recently, graph-based methods have achieved revolutionary performance on public datasets. The search algorithm on a graph is a greedy algorithm. It can be applied to various graph structures and shows promising performance. Recent improvements of the graph-based methods mainly focus on two aspects: (1) Some provide better initial search position to prevent the search from being stuck in some local optima, which is far away from the correct answers. (2) Others try to construct better graphs for faster traversing and neighbor locating. In our observation, an ideal graph for search should consider four aspects, (1) lowering the average out-degree of the graph for fast traversing; (2) ensuring the connectivity of the graph; (3) avoiding detours; and (4) avoiding additional index structures to reduce the index size. None of the previous methods take all these four aspects into consideration simultaneously, which prevents them from achieving better performance. In this paper, we present a novel graph structure named Navigating Spreading-out Graph (NSG) to take the four aspects into account simultaneously. Extensive experiments show that our algorithm outperforms all the existing algorithms significantly. What's more, our algorithm outperforms the existing approach of Taobao (Alibaba Group) and has been integrated into their search engine for billion-scale search.

CCS CONCEPTS

• **Information systems → Top-k retrieval in databases**;

KEYWORDS

approximate nearest neighbor search, graph-based search, large-scale search, high dimensional data indexing

ACM Reference Format:
Cong Fu Changxu Wang Deng Cai. 2018. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. In *Proceedings of arXiv*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Approximate nearest neighbor search (ANNS) has been a hot topic over decades and provides fundamental support for many applications in data mining, machine learning and information retrieval[3, 5, 8, 10, 30, 35]. For sparse discrete data (like documents), the nearest neighbor search can be carried out efficiently on advanced index structures (*e.g.*, inverted index[28]). For dense continuous vectors, various solutions have been proposed such as tree-structure based approaches[6, 14, 19, 29], hashing-based approaches[15, 17, 32], quantization-based approaches[16, 20, 34] and graph-based approaches[1, 18, 26, 33]. Among them, graph-based methods have shown great potential recently, and have demonstrated the significant improvements over other methods experimentally [13, 21].

The search algorithm via graphs has been studied for decades[1, 18, 23, 31]. The basic idea is *"a neighbor of a neighbor is also likely to be a neighbor."* The algorithm starts from a random node in the graph, iteratively checks the neighbors of neighbors, and proceeds towards the nodes which are closer to the query. KGraph[11] improves this algorithm by maintaining a visited node set for backtracking. For convenience, we refer to this algorithm as **Search-on-Graph (SG)** algorithm. It's formally given in **Algorithm 1**.

Later improvements of the graph-based methods are mainly in two aspects. Some works focus on the initialization of the search algorithm. Works [13, 21] found that the SG algorithm easily gets trapped in local optima. They improved the algorithm by providing a better search start position with additional index structures like hash index[21] and KD-tree[13]. This leads to better search performance but much larger index size[13].

Other works try to find better graph structures for ANNS. Some works[1, 23, 31] are based on the relative neighborhood graph (RNG). The time complexity of SG algorithm on the RNG scales polylogarithmically with the data, which is quite low for the high dimensional ANNS problem. However, the indexing time complexity of the RNG is $O(N^2)$[1], which limits its scalability. The Fast Approximate Nearest Neighbor Graph (FANNG) proposed in [4] is an approximation of the RNG. They propose a heuristic indexing algorithm for lower time complexity. The kNN graph used in works[11, 13, 18, 21] is an approximation of the Delaunay Graph[24]. The Delaunay Graph can ensure an efficient exact nearest neighbor search at low dimension, but the performance decreases significantly when the dimensionality increases. Because it soon becomes fully connected[4]. SG algorithm on the kNN graph cannot ensure the exact search, but it has shown promising performance for ANNS. Recently, another graph structure, the navigable small-world network[7, 22], attracts the interests of the researchers. The traversing between any two nodes on a navigable small-world network [22] is of polylogarithmic time complexity, which makes it suitable for the ANNS problem. Navigable small-world graph (NSW)[26] and Hierarchical NSW (HNSW)[27] try to approximate the navigable small-world network for high-performance ANNS.

arXiv, February 10, 2018, Hangzhou, Zhejiang, China
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 $\mathbf{q}$

```
i=0
candidate pool S = ∅
S.add(p)
while i < l do
    i =the index of the first unchecked node in S
    mark p_i as checked
    for all neighbor n of p_i in G do
        S.add(n)
    end for
    sort S in ascending order of the distance to q
    remove the distant nodes in S to keep its size no larger
    than l
end while
return the first k nodes in S
```

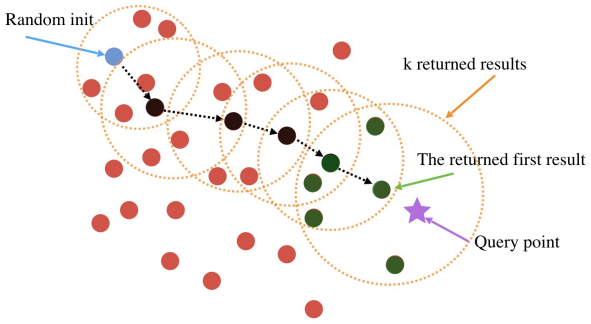


Figure 1: We show a typical search behavior of the Search-on-Graph (SG) algorithm on a 6-NN graph here. During the search, we need to calculate the distances between all the nodes in the orange circles and the query. If we reduce the edges within the circles and ensure the path is unobstructed, we can arrive at the answer sooner. What's more, this also reduces the size of the graph by lowering the degree.

In our observation, an ideal graph for search should take four aspects into consideration. The first is lowering the average out-degree of the graph as RNG[1], FANNG [4], and HNSW [27] did. The out-degree of the graph influences the efficiency of traversing in the graph significantly (please see **Figure.1**). A low average out-degree shorten the average stay on the nodes of the SG algorithm. The second is ensuring the connectivity of the graph [27]. In other words, it should be guaranteed that there is a path from the starting node to the destination node in each search. When one limits the degree of each node to a fixed value for fast traversing, the connectivity of the graph is hard to be guaranteed (see **Figure.2**). The third is avoiding detours. A node in the graph should be directly connected with its nearest neighbor to prevent the SG algorithm from making detours when the algorithm explores in a small neighborhood area of the node (see **Figure.3**). The last one is avoiding using additional index structures. Using additional index structures (IEH [21] and Efanna [13]) may improve the search performance but increase the index size, which limits the scalability of these algorithms. All the existing graph-based methods failed to consider all the four aspects simultaneously (please see **Table 1**).

In this paper, we propose a novel graph structure named Navigating Spreading-out Graph (NSG) to address all the four aspects simultaneously. Firstly, we select a navigating node as the fixed starting node of the search. And we ensure the connectivity from the navigating node to each of the other nodes in the graph. Secondly, we guarantee that almost all the nearest neighbors are connected with each other to avoid the detour problem. Thirdly, we lower the degree of the graph with the pruning strategy of RNG for efficient traversing. We also limit the maximum out-degree of the NSG for a smaller graph size. Finally, the NSG contains no extra structures except for a graph. It's worthwhile to highlight our contributions as follows.

(1) Based on summarizing previous works and our further exploration, we first propose four aspects that an ideal graph for search should consider: lowering the average out-degree, ensuring the connectivity of the graph, avoiding detours, and avoiding additional index structures.
(2) We introduce a novel graph structure NSG for high-performance ANNS to address the four aspects simultaneously, and we propose an efficient algorithm to build the NSG. Extensive experiments show that our approach outperforms the state-of-the-art methods in search performance with the smallest index size among graph-based methods.
(3) Our approach outperforms the existing baseline of Taobao (Alibaba Group), and it has been integrated into their search engine for billion scale search.

2 GRAPH-BASED ANNS METHODS

The search algorithm via the graphs (SG algorithm) has been studied for a long time[1, 18, 23, 31]. It's a simple greedy process on a graph. The algorithm starts from a random node. At each step, it checks the neighbors of the current node and moves to closer nodes to the query. The algorithm repeats this process until on closer nodes can be found. However, it's soon found that SG algorithm easily gets stuck in local optima. An improved version was introduced in KGraph[11]. Like the depth-first-search in trees, KGraph[11] provides a backtracking mechanism by maintaining a set of the nodes that have been visited. The set is sorted in the ascending order of the distance to the query. When the algorithm gets stuck in local optima, it will restart from a node in the set whose neighbors haven't been checked. Please see Algorithm.1 for the pseudo-code of this algorithm.

Considering the random initialization might be sub-optimal, IEH[21] and Efanna[13] were proposed to provide a better start position for the SG algorithm. The idea is to use additional index structures which are efficient at a low accuracy region, like hashing[21] or KD-tree[13]. They use them to generate better initial position with a small time cost. However, the additional index structures increase the index size a lot.

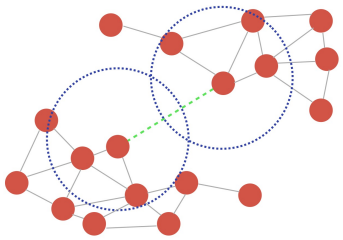


Figure 2: When we limit the degree of each node to a fixed value for efficient traversing and try to ensure links among nearest neighbors, the coverage of each node's neighborhood will be restricted to a small area. Thus the edges of the nodes at a sparse area between clusters tend to be oriented towards the respective cluster centers, the long edges between clusters tend to be discarded.

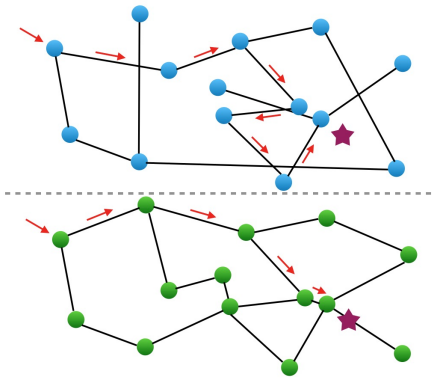


Figure 3: An illustration of the detour problem. Traversing on the graph above (not all nearest neighbors are directly connected with each other) takes more steps than the graph below (nearest neighbors are directly connected with each other).

Also, some works focus on improving the graph structure. Early works[1, 23, 31] use the RNG for ANNS. SG algorithm via the RNG is of polylogarithmic time complexity, which is an excellent property. However, the time complexity of building an RNG is $O(N^2)$[1], which is impractical for massive problems. Harwood *et al.* [4] propose a graph structure named as FANNG. It's actually an approximation of the RNG. Although they interpret their motivation from a different perspective, the occlusion rule used in FANNG is the same as the pruning strategy used in the RNG. FANNG uses a heuristic algorithm to speed up the indexing. However, it still needs many iterations to converge to a high-quality graph for ANNS.

The *k*NN graph used in [11, 13, 18, 21] is an approximation of the Delaunay Graph[24], which can be constructed by Delaunay triangulation algorithm. Delaunay Graphs can be explored in a deterministic way that is usually very efficient at low dimension and can guarantee that the absolute nearest neighbor to a query point will be found. Unfortunately, as the dimensionality of a dataset increases, the search efficiency of the Delaunay Graphs rapidly reduces, because they quickly become almost fully connected. The *k*NN graph cannot guarantee the exact search for high dimensional data, but it can be constructed efficiently with the nn-descent algorithm[12]. And they have shown promising results in the ANNS scenario.

The small-world phenomenon – the principle that most of us are linked by short chains of acquaintances – was first investigated as a question in sociology[7, 22]. This property makes it welcomed in the ANNS scenario. Work[2] has shown the possibility of using navigable small-world networks[7, 22] for finding the nearest neighbor with a greedy search algorithm. The algorithm in [2] approximates the small-world network by adding random long-range short-cuts among nodes and short links among closest neighbors. However, the performance relies heavily on the quality of the approximation, and it in principle only works in Euclidean space[26]. Malkov *et al.* [26] proposed NSW to approximate the navigable small-world network in global perspective by random inserting nodes into an empty graph. While for each newly inserted node, they greedily search for its nearest neighbors in the current graph with Algorithm.1 to approximate the Delaunay Graph in this local neighborhood. In their opinion, they can synthesize the advantages of the Delaunay Graph and the small-world network in the NSW.

Later Malkov *et al.* [27] find that there exist two problems with the NSW. The first one is the high average out-degree of the NSW. Since the SG algorithm checks all the neighbors of a node at each step, a high average out-degree of the graph will result in a long stay on a node as **Figure 1** shows. That's why recent state-of-the-art methods all try to prune their graphs from different perspectives. In HNSW[27], they leverage the pruning strategy used in RNG and FANNG to prune the NSW for efficient traversing.

The other one is the connectivity problem in the NSW. For fast traversing, we must **limit the maximum out-degree** of a graph to a small value. The out-degree of a node determines the diameter of the space it covers. Thus, there tends to be few or even no edges across the sparse areas between data clusters, as **Figure 2** illustrates. Malkov *et al.* try to solve this problem by building multiple **pruned NSW graphs** on the dataset (HNSW)[27]. The data points on each graph are sampled in different proportions. Then they stack them up like a pyramid. The upper layers cover fewer nodes, while the lower ones cover more nodes. The bottom layer of the HNSW is **the only layer containing all the data points**. They believe the edges in the upper layers can serve as long-range short-cuts, while those in lower layers can help with a fine-grained search. HNSW outperforms NSW significantly with the above two problems addressed[27]. However, their solutions cause a new problem which is the large-index-size problem. It's because the upper graphs can be regarded as extra index structures and occupy quite a lot of memory.

Meanwhile, there also exists the detour problem in previous works like NSW, FANNG, and HNSW. These methods all build their graphs from an empty graph. In the beginning, the edges are randomly inserted into the graphs. For each insertion, they use the SG algorithm to search on the **current incomplete graph** for nearest neighbors of the node being inserted. Then they add links between these nodes. Suppose node s is inserted into the graph at the time T. The nearest neighbors of node s are provided by the SG algorithm

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. √ means the method takes actions in this aspect. × means no actions.

algorithm	Pruning the edges	Graph connectivity	Avoiding detours	No extra index structures
RNG[1]	√	×	√	√
KGraph[11]	×	×	√	√
Efanna[13]	×	×	√	×
IEH[21]	×	×	√	×
FANNG[4]	√	×	×	√
NSW[26]	×	×	×	√
HNSW[27]	√	√	×	×
DPG[25]	√	×	√	√
NSG-Naive	√	×	√	√
NSG	√	√	√	√

approximately. The search accuracy of the SG algorithm on their "intermediate" indices is unsure. Thus, there is no guarantee that node s can be connected with its nearest neighbor inserted before time T or after time T. According to our experimental study, a large proportion of the nodes in their final indices are not connected to their nearest neighbor. This may cause the detour problem as **Figure 3** illustrates.

Since the nearest neighbors in a kNN graph are connected with each other, the kNN graph-based methods (KGraph, IEH, and Efanna) don't suffer from the detour problem. For efficiency in the indexing, they all turn to the approximate kNN graph with high quality. But according to our experimental assessments, they usually need a large enough k to get their best performance. It may be because a large k (out-degree) ensures a large diameter of space coverage. Thus, they all suffer from the inefficient traversing problem. Besides, a large average out-degree leads to large index size.

Although the Diverse Proximity Graph (DPG) [25] is a graph pruned out of a kNN graph, they also suffer from inefficient traversing. Because they only cut off half of the edges of the kNN graph to maximize the average angles among the edges. In addition, they turn the directed graph into an undirected one by adding reverse edges to the graph. The performance of the DPG is sensitive to the value of k[25]. According to our experimental study, they achieve their best performance when k is large. As a consequence, the resulting undirected graph is of a large average out-degree.

In summary, none of the previous graph-based methods consider all the four aspects simultaneously. The detailed information is given in **Table 1**.

3 NAVIGATING SPREADING-OUT GRAPH

3.1 Motivation

An ideal graph structure for the search should provide a good solution in four aspects simultaneously: lowering the average out-degree, ensuring the connectivity of the graph, avoiding detours, and avoiding using extra index structures.

Firstly, we should guarantee the connectivity of the graph. The graph connectivity is crucial to the performance of graph-based methods. Most previous methods (RNG, KGraph, DPG, IEH, Efanna, FANNG, and NSW) perform the SG algorithm with a query-related starting position on their graphs (random initialization or leveraging other structures). If there are no edges between clusters as **Figure 2** shows, the only chance for them to get the right answer is that the starting node and the destination node are within the same cluster. In our observation, it's easier to ensure the connectivity from a fixed node to all the others than between each pair of the nodes. If there exists a path from a node s to each of the others, we can generate a tree by depth-first-search with node s being the root. Therefore, we choose an approximate medoid of the dataset as the **navigating node**, where the SG algorithm always starts. Then we can ensure the connectivity from it to the others by building a tree from it and linking the out-of-tree nodes to the in-tree nodes close to them.

Secondly, we should avoid the detour problem. The behavior of the SG algorithm can be summarized as two stages. First is the **large-range traversing** from the starting node to the neighborhood area of the query. Second is the **small-range exploring** in the neighborhood area of the query for the k nearest neighbors. Graphs with the detour problem will suffer from inefficiency at the small-scale exploring stage (**Figure 3**). We can solve this problem by building a kNN graph and ensure the nearest neighbors are connected with each other in the final graph. However, it's too time-consuming to build an exact kNN graph. We can build a high-quality approximate kNN graph to alleviate this problem as much as possible. And there are very efficient algorithms proposed for it, such as nn-descent[12] and Efanna[13].

Thirdly, we should lower the average out-degree of the graph by removing "redundant" edges. As we discussed above, a low average-degree of the graph not only leads to smaller index size but also ensures more efficient traversing (**Figure 1**). We leverage the pruning strategy proposed in RNG[1] (*i.e.*, the occlusion rule in FANNG[4]), which can cut off "redundant" long edges and dramatically lower the average out-degree of the graph.

Here we give a formal definition of this strategy (the occlusion rule). For a given node s, we put the candidate nodes for pruning into a sequence Q. The sequence Q is sorted according to the distance to node s. The result neighbor set of node s is R. The first node in Q is removed from Q directly and added to R. Next for the rest nodes in Q, we take out the first node m and remove it from Q. We need to verify whether node m can be added to set R. The node m can be added to set R if and only if $|ms| < |mn|, \forall n \in R$, where $|mn|$ means the distance between node m and node n. We repeat this step until there is no node left in Q.

Since the search via NSG always starts from the navigating node, we should ensure the traversing efficiency along all the search paths (from the navigating node to each of the others). We can use the pre-built kNN graph to locate the search paths. We treat each node except for the navigating node as a query and perform the SG algorithm on the kNN graph. The nodes lying on the search path and the k nearest neighbors will be regarded as the candidates for pruning.

Finally, the NSG index contains no extra index structures except for a graph, and we limit the maximum out-degree of our graph to a small value. Therefore, the resulting NSG has very small size.

Below we present our construction algorithm in detail.

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**

```
calculate the centroid c of the dataset.
r =random node.
n =Search-on-Graph(G,r,c,l)%navigating node
for all node v in G do
    Search-on-Graph(G,n,v,l)
    E =all the nodes checked along the search
    add v's nearest neighbors in G to E
    sort E in the ascending order of the distance to v.
    result set R = ∅, p_0 = the closest node to v in E
    R.add(p_0)
    while !E.empty() && R.size() < m do
        p = E.front()
        E.remove(E.front())
        for all node r in R do
            if edge pv occluded by edge pr then
                break
            end if
        end for
        if no occlusion occurs then
            R.add(p)
        end if
    end while
end for
while True do
    build a tree with edges in NSG from root n with depth-first
    -search.
    if not all nodes linked to the tree then
        add an edge between one of the out-of-tree nodes
        and its closest in-tree node (by SG algorithm).
    else
        break.
    end if
end while
```

3.2 NSG Construction Algorithm

The main steps to build an NSG are as follows:

(1) Build a high-quality approximate kNN graph with efficient algorithm provided by [11, 13].
(2) Choose the approximate medoid as the navigating node. Specifically, we take the centroid of the dataset as the query. Then we use Algorithm.1 to search on the kNN graph to locate its nearest node as the navigating node.
(3) For each node, we take it as a query and perform Algorithm.1 on the kNN graph with the navigating node as the starting node. The nodes along the search path are regarded as the candidates. Meanwhile, we add the k nearest neighbors of each node in the kNN graph into the candidate set.
(4) We maintain a result set for this node. The edge that connects this node and its nearest neighbor will be added to the set immediately. For each edge left in the candidate set, we add them to the result set in ascending order of length. Then we use the occlusion rule for pruning.
(5) Finally, we will build a tree by the depth-first-search on the graph and use the navigating node as the root. If there exist nodes that are not linked to the tree after the tree building, we add links from unconnected nodes to its closest leaf nodes (by using SG algorithm on the current graph). Then we continue building the tree with the depth-first-search. When all the nodes are connected to the tree, we can ensure there exists a path from the navigating node to each of the other nodes.

Please see the pseudo code in the Algorithm 2 for more details.

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

3.3 Search With NSG

The search via the NSG is quite simple. For any query, we just apply Algorithm.1 with the navigating node as the starting node.

Our code has been released at GitHub[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

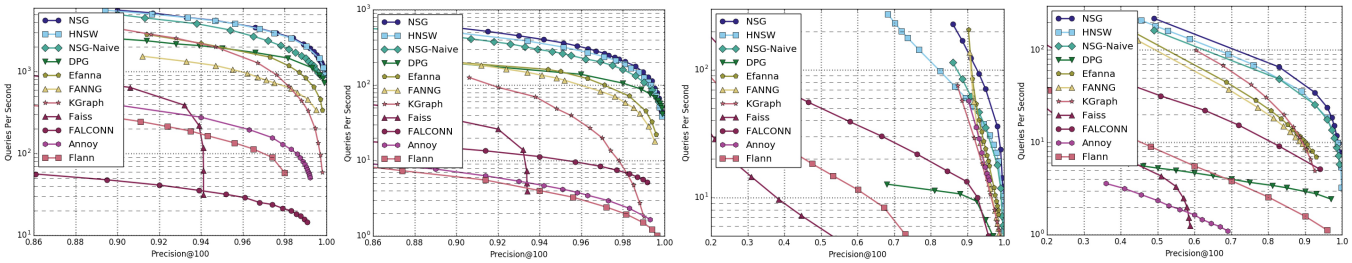
The experiments are conducted on four datasets. SIFT1M and GIST1M are in the well-known BIGANN public ANNS dataset collection[2], which are widely used in the literature[4, 20]. RAND4M and GAUSS5M are two synthetic datasets. RAND4M are vectors sampled from the uniform distribution of range (-1, 1). GAUSS5M are vectors sampled from a Gaussian distribution $\sim N(0, 3)$. 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 Evaluation Metrics

For ANNS problem, an algorithm is expected to return K points. We need to verify how many of them are correct K nearest neighbors (ground truth). Therefore, we use the *precision* defined in [13] as our evaluation protocol. Suppose the returned point set of a given query q is R' and the correct k nearest neighbor set of q is R, then the *precision* is defined as below.

$$precision(R') = \frac{|R' \cap R|}{|R'|} = \frac{|R' \cap R|}{K}. \quad (1)$$

[1] https://github.com/ZJULearning/nsg
[2] http://corpus-texmex.irisa.fr/



(a) SIFT1M (b) GIST1M (c) RAND4M (d) GAUSS5M

Figure 4: ANN search results of all compared algorithms on the four datasets. All the algorithms use their best performing indices at high precision. (top right is better)

Because $|R'| = |R| = K$ in our experiments, the *precision* here is the same as the *recall* used in [20]. We will report the performance with $K = 100$ throughout all the experiments.

4.3 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 are available and well optimized on the GitHub. For those who didn't 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**[3] 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**[4] is based on a binary search forest.
(2) **Hashing-Based Methods.** FALCONN[5] is a well-known ANNS library based on multi-probe locality sensitive hashing.
(3) **Quantization-Based Methods. Faiss**[6] 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**[7] is based on a kNN Graph. **Efanna**[8] 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 [4]. They didn't release their codes. Thus, we implement their algorithm according to their paper and with the same optimization as our codes. **HNSW**[9] is based on a hierarchical graph structure, which was proposed in [27]. **DPG**[10] is based on an undirected graph pruned from a kNN graph. According to an open source benchmark[11], **HNSW is the fastest ANNS algorithm on CPU before NSG.**
(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-and-prune operation and the guarantee of the graph connectivity. We directly prune the kNN graph with the occlusion rule[4] to get NSG-Naive. There is no navigating node, and we use the SG algorithm with random initialization on NSG-Naive.

4.4 Results

We randomly sample one percentage points out of each base set as their corresponding **validation sets**. Since it's essential to be fast at 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 at 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 **Figure.4, 5**. The detailed information of the graph-based indices is given in **Table.3**. It's important to point out that the index size is **in-memory size**.

The percentages of the nodes which are connected with its nearest neighbors (**NN-percentage**) in all the graphs are shown in **Table 4**. Because HNSW contains multiple graphs in its index, we only report the average out-degree, maximum out-degree, and the NN-percentage of its bottom-layer graph. The bottom-layer graph of HNSW is the only layer which contains all the points of the dataset.

Figure 4 shows the performance of all the algorithms. As we can see, the graph-based methods are far better than the others, we redraw the curves in **Figure 5** for all the graph-based methods to

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

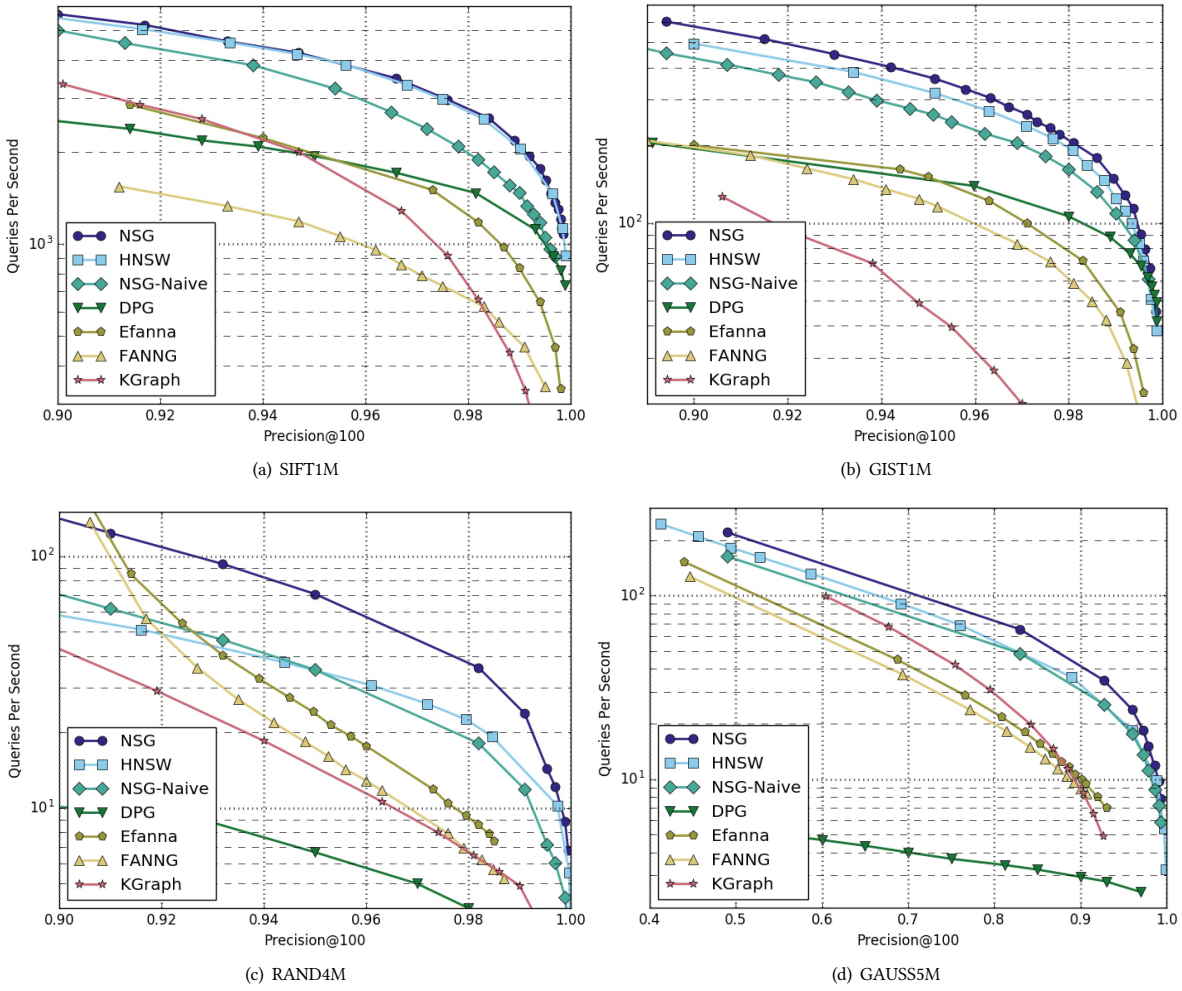


Figure 5: ANN search results of graph-based algorithms on the four datasets. All the algorithms use their best performing indices at high precision. (top right is better)

better exhibit the difference. Many interesting points can be found as follows:

(1) In **Figure 4**, 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 considers the four aspects simultaneously.

(2) HNSW is the second best-performing algorithm. It is because HNSW provides excellent solutions to ensure the connectivity of the graph and lower the average out-degree. And NSG takes a further step to address the detour problem. This is why NSG can outperform HNSW. We can conclude that low average out-degree, the connectivity of the graph, and avoiding detours are crucial to the performance of the graph-based methods. As a bonus, NSG doesn't need extra index structures to achieve better performance.

(3) NSG is the most memory-efficient algorithm among the graph-based methods since NSG has the smallest maximum out-degree. The memory occupations of NSG, HNSW, FANNG, Efanna, 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 (better search performance). Although 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. However, the index size of DPG is still larger than NSG due to its large average out-degree. From **Figure 5** and **Table 3** we can see that NSG

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. Because HNSW contains multiple graphs, we only report the AOD and MOD of its bottom-layer graph here.

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

has a superior search performance with the smallest index among all the graph-based methods. Especially the index size of NSG is about 1/2-1/3 of the HNSW, which is the previous best performing algorithm[12].

(4) The difference between NSG-Naive and NSG is that NSG-Naive doesn't guarantee the **connectivity of the graph** as NSG does. Besides, their candidates for the pruning are different. The candidates for the pruning of NSG cover the search paths from the navigating node to each of the others, while the candidates of NSG-Naive only contains the k nearest neighbors of each node. The significant improvement of NSG over NSG-Naive shows that the essentiality of our indexing steps.

(5) **Table 4** shows the NN-percentages (the percentage of the nodes which are connected with their nearest neighbor) of all the graphs. We can see the NN-percentages of HNSW and FANNG are much lower than the other graph-based methods, which indicates HNSW and FANNG suffer from the detour problem. NSG outperforms FANNG and HNSW significantly. It mainly owes to that NSG addresses the **detour problem**. KGraph, DPG, and Efanna also don't suffer from the detour problem. However, no guarantee of the connectivity of the graph and large average out-degrees lead to their inferior performance. KGraph and Efanna achieve their best performance when the k of their kNN graphs is large (a large k ensures large coverage around a node). KGraph and Efanna don't involve pruning in the indexing and suffer from inefficient traversing. DPG has a large average out-degree due to their improper pruning strategy. They only cut off half of the edges of the kNN graph and then turn it into an undirected graph by adding reverse edges. Given that the original kNN graph uses a large k (better space coverage), the resulting DPG still has a large average out-degree and inferior traversing speed.

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

Table 4: The NN-percentages (the percentage of the nodes which are linked to their nearest neighbor) of all the graphs. HNSW_0 means the bottom-layer graph of the HNSW index.

dataset	algorithm	NN(%)	algorithm	NN(%)
SIFT1M	NSG	99.3	HNSW_0	66.3
	FANNG	60.4	DPG	99.4
	KGraph	99.4	Efanna	99.4
GIST1M	NSG	98.1	HNSW_0	47.5
	FANNG	39.9	DPG	98.1
	KGraph	98.1	Efanna	98.1
RAND4M	NSG	96.4	HNSW_0	76.5
	FANNG	66.7	DPG	96.6
	KGraph	96.6	Efanna	96.6
GAUSS5M	NSG	94.3	HNSW_0	57.6
	FANNG	53.4	DPG	94.3
	KGraph	94.3	Efanna	94.3

4.5 The Search and Indexing Complexity of NSG

NSG is an approximation of the RNG, which has polylogarithmic time complexity. According to our experimental study, the search time complexity of NSG also scales polylogarithmically with the data (about $O((\log N)^2)$).

As for the index construction, the complexity has two parts, approximate kNN graph construction (nn-descent algorithm, $O(n^{1.14})$[12]) and later steps. The later steps adopt the "search-and-prune" routine. In our experimental assessment, the total time complexity of NSG's indexing procedure is about $O(n^{1.3})$.

4.6 Experiments on E-commerce Data

We have collaborated with Taobao of Alibaba Group on billion-scale ANNS problem. Before NSG, they already have a baseline approach which is a quantization based method. Like IVFPQ[20], this approach has a two-level hierarchical structure. A K-means based inverted index serves as the coarse search index, and the product quantization within each cluster serves as the fine-grained search index. They further optimize the index structure to support large-scale distributed search.

We compare NSG with the baseline on two datasets sampled from the same large-scale e-commerce database with all the goods are represented by 128 dimension vectors. The results are shown in Table.5. On the dataset with 10 million vectors, we test the performance of the algorithms on a single thread. While on the dataset

Table 5: 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. OQL98 means One-Query-Latency to retrieve 100 neighbors at 98% precision. It measures the average time interval, from the time that one query arrives at the server to the time when 100 neighbors are returned. QB is the quantization-based baseline.

data set	algroithm	NT	OQL98 (ms)
E10M	NSG	1	2.3
E10M	QB	1	10
E45M	NSG	12	1
E45M	QB	12	10

with 45 million vectors, we test the **inner-query parallelism** performance of the algorithms on 12 threads. To support the inner-query parallelism, we random divide the dataset into 12 subsets and build 12 graphs, one for each subset. For each query, we search 12 subgraphs in parallel and merge the results.

We can see that our algorithm outperforms the baseline on both settings significantly, and the NSG has been integrated into their search engine for billion-scale search.

5 CONCLUSIONS

In this paper, we present a novel graph index, NSG, to provide efficient approximate nearest neighbor search. It takes the four aspects (lowering the average out-degree, ensuring the connectivity, avoiding detours and avoiding additional index structures) into consideration simultaneously. Extensive experiments show NSG outperforms the other state-of-the-art algorithms with the smallest index size among 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.

REFERENCES

[1] Sunil Arya and David M Mount. 1993. Approximate Nearest Neighbor Queries in Fixed Dimensions.. In *SODA*, Vol. 93. 271–280.

[2] Olivier Beaumont, Anne-Marie Kermarrec, and Étienne Rivière. 2007. Peer to peer multidimensional overlays: Approximating complex structures. *Principles of Distributed Systems* (2007), 315–328.

[3] Jeffrey S. Beis and David G. Lowe. 1997. Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. In *1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*. 1000–1006.

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

[5] Kristin P Bennett, Usama Fayyad, and Dan Geiger. 1999. Density-based indexing for approximate nearest-neighbor queries. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 233–243.

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

[7] Marian Boguna, Dmitri Krioukov, and Kimberly C Claffy. 2009. Navigability of complex networks. *Nature Physics* 5, 1 (2009), 74–80.

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

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

[10] Arjen P de Vries, Nikos Mamoulis, Niels Nes, and Martin Kersten. 2002. Efficient k-NN search on vertically decomposed data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM, 322–333.

[11] Wei Dong. 2014. KGraph, an open source library for K-NN graph construction and nearest neighbor search. *www.kgraph.org* (2014).

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

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

[14] Keinosuke Fukunaga and Patrenahalli M. Narendra. 1975. A Branch and Bound Algorithm for Computing k-Nearest Neighbors. *IEEE Trans. Comput.* 100, 7 (1975), 750–753.

[15] Jinyang Gao, Hosagrahar Visvesvaraya Jagadish, Wei Lu, and Beng Chin Ooi. 2014. DSH: data sensitive hashing for high-dimensional k-nnsearch. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 1127–1138.

[16] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization for approximate nearest neighbor search. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE, 2946–2953.

[17] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*. 518–529.

[18] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. 2011. Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph.. In *IJCAI 2011, Proceedings of the International Joint Conference on Artificial Intelligence*, Vol. 22. 1312–1317.

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

[20] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2011), 117–128.

[21] Zhongming Jin, Debing Zhang, Yao Hu, Shiding Lin, Deng Cai, and Xiaofei. He. 2014. Fast and Accurate Hashing Via Iterative Nearest Neighbors Expansion. *IEEE transactions on cybernetics* 44, 11 (2014), 2167–2177.

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

[23] Philip M Lankford. 1969. Regionalization: theory and alternative algorithms. *Geographical Analysis* 1, 2 (1969), 196–212.

[24] Der-Tsai Lee and Bruce J Schachter. 1980. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer & Information Sciences* 9, 3 (1980), 219–242.

[25] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Wenjie Zhang, and Xuemin Lin. 2016. Approximate Nearest Neighbor Search on High Dimensional Data—Experiments, Analyses, and Improvement (v1. 0). *arXiv:1610.02455* (2016).

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

[27] Yury A. Malkov and D. A. Yashunin. 2016. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *arXiv:1603.09320* (2016).

[28] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. 2008. *Introduction to information retrieval*. Vol. 1. Cambridge university press Cambridge.

[29] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition*. 1–8.

[30] George Teodoro, Eduardo Valle, Nathan Mariano, Ricardo Torres, Wagner Meira, and Joel H Saltz. 2014. Approximate similarity search for online multimedia services on distributed CPU–GPU platforms. *The VLDB Journal* 23, 3 (2014), 427–448.

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

[32] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *Advances in neural information processing systems*. 1753–1760.

[33] Yubao Wu, Ruoming Jin, and Xiang Zhang. 2014. 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*. ACM, 1139–1150.

[34] Ting Zhang, Chao Du, and Jingdong Wang. 2014. Composite Quantization for Approximate Nearest Neighbor Search.. In *ICML*. 838–846.

[35] Yuxin Zheng, Qi Guo, Anthony KH Tung, and Sai Wu. 2016. Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2023–2037.