Hierarchical Satellite System Graph for Approximate Nearest Neighbor Search on Big Data

JIARU ZHANG, RUHUI MA, and TAO SONG, Shanghai Jiao Tong University, China
YANG HUA, Queen's University Belfast, UK
ZHENGUI XUE, Shanghai Jiao Tong University, China
CHENYANG GUAN, Jiangnan University, China
HAIBING GUAN, Shanghai Jiao Tong University, China

Approximate nearest neighbor search is a classical problem in data science, which is widely applied in many fields. With the rapid growth of data in the real world, it becomes more and more important to speed up the nearest neighbor search process. Satellite System Graph (SSG) is one of the state-of-the-art methods to solve the problem. However, with the further increase of the data scale of problems, SSG still needs a considerable amount of time to finish the search due to the limitation of step length and start point locations. To solve the problem, we propose Hierarchical Satellite System Graph (HSSG) and present its index algorithm and search algorithm. The index process can be distributed deployed due to the good parallelism of our designed hierarchical structure. The theoretical analysis reveals that HSSG decreases the search steps and reduces the computational cost and reduces the search time by searching on the hierarchical structure with a similar indexing time compared with SSG, hence reaches a better search efficiency. The experiments on multiple datasets present that HSSG reduces the distance computations, accelerates the search process, and increases the search precision in the real tasks, especially under the tasks with large scale and crowded distributions, which presents a good application prospect of HSSG.

CCS Concepts: • **Information Systems** → **Nearest-neighbor search**;

Additional Key Words and Phrases: Nearest neighbor search, hierarchical structure, approximate nearest neighbor search, big data, data science

ACM Reference format:

Jiaru Zhang, Ruhui Ma, Tao Song, Yang Hua, Zhengui Xue, Chenyang Guan, and Haibing Guan. 2022. Hierarchical Satellite System Graph for Approximate Nearest Neighbor Search on Big Data. *ACM/IMS Trans. Data Sci.* 2, 4, Article 32 (February 2022), 15 pages.
<https://doi.org/10.1145/3488377>

This work was supported in part by National NSF of China (NO. 61872234, 61732010, 61525204) and Shanghai Key Laboratory of Scalable Computing and Systems.

Authors' addresses: J. Zhang, R. Ma (corresponding authors), T. Song (corresponding authors), Z. Xue, and H. Guan, Shanghai Jiao Tong University, Dongchuan Rd 800, Shanghai, P.R.China; emails: {jiaruzhang, ruhuima, songt333}@sjtu.edu.cn, zhenguihue@sjtu.edu.cn, hbguan@sjtu.edu.cn; Y. Hua, Queen's University Belfast, University Road, Belfast Northern Ireland, UK, BT7 1NN; email: Y.Hua@qub.ac.uk; C. Guan, Jiangnan University, Wuxi, Jiangsu, P.R.China; email: 1033180320@stu.jiangnan.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.

2577-3224/2022/02-ART32 \$15.00

<https://doi.org/10.1145/3488377>

1 INTRODUCTION

With the coming of Big Data Era, searching from massive amounts of saved data to get the most similar items becomes necessary in many practical problems. Therefore, more and more applications use the **Nearest Neighbor Search (NNS)** technology, and they play an important role in many fields, e.g., Data Mining, Information Retrieval, Machine Learning, Database, Multimedia, Social Network, and Computer Vision [2, 8, 10, 29]. In a dataset with N data points (nodes), the time complexity of search accurately is $O(N)$. Therefore, when there is a huge amount of points in the data, it is remarkably time-consuming to perform accurate search. However, a precise search result is usually unnecessary and an approximate result is sufficient in many practical tasks. Therefore, the **Approximate Nearest Neighbor Search (ANNS)** problem is proposed and gets widespread attention. Researchers have proposed many algorithms to solve the problem, which can be divided into three categories: tree-based algorithms, hash-based algorithms, and graph-based algorithms. The tree-based algorithms and hash-based algorithms usually need to check more nodes to reach the target node when searching, which brings greater time consumption [12]. Therefore, compared with the other two types of algorithms, graph-based algorithms have better performance and are also more widely used in practical problems.

Compared with other graph-based algorithms, a series of **Monotonic Search Networks (MSNETs)** methods [6], e.g., **Navigating Spreading-out Graph (NSG)** [11], achieves an obviously good performance. Since the monotonicity of the index graph is guaranteed in the MSNET methods, the query process can approximate to the target node in each step, which reduces a lot of useless access and greatly reduces the time overhead of the searching process. However, limited by the A*-like search algorithm used in the MSNET algorithms, the search process can only move to the neighbor points in a step. Consequently, the search process still needs relatively many steps to approach the target points, especially in the case of large data volume and dense data distribution, which obviously reduces the search efficiency. As shown in the left part of Figure 1, there are total 18 steps in the search process from the source node to the target node in the traditional MSNET method. Aiming at the above problem, a feasible improvement method is to randomly select multiple source nodes to construct an MSNET. Before the search process, the algorithm finds the nearest source node from the multiple source nodes and then starts to search from it. The expectation of the distance between the target node and the source node is reduced with it, and the problem of excessive search steps is alleviated to a certain extent. The current state-of-the-art method, **Satellite System Graph (SSG)** [10], uses this method, where 10 nodes are randomly selected from the whole dataset as start nodes. However, as the number of nodes is fixed in the method, it is difficult to find a suitable number of start nodes to quickly select from the starting nodes and search fast at the same time, especially in a large data situation. Therefore, there is still a large time consumption in the algorithm and the method does not solve the fundamental problem.

In this article, we propose a hierarchical method to build the MSNET hierarchically to solve the above problem. In our method, the nodes in the dataset are divided into different layers, and the MSNETs are built separately on each layer. The search process can skip a large distance when search on the high layer, which reduces the total search steps, especially in the large-data situations. As shown in the right part of Figure 1, the search process on MSNET with a simple two-layer structure only needs 10 steps. Concretely, we select the SSG as the graph used in each layer and build the **Hierarchical Satellite System Graph (HSSG)** in the whole dataset. Because the index processes in different layers are unrelated to each other once the nodes are determined, the index algorithm can be distributedly run, which reduces the time consumption of the index algorithm. When searching, we perform a faster coarse search on the upper layer with fewer nodes. After obtaining the coarse results, we recursively perform a fine search with higher precision in the lower layer. The final search on the bottom layer returns the accurate search result. The algorithm reaches

Fig. 1. The comparison between MSNET without hierarchical structure and MSNET with hierarchical structure. The search process on MSNET without hierarchical structure needs 18 steps to reach the terminal node, while search process on MSNET with a simple two-layer structure only needs 10 steps.

a better tradeoff between the search precision and speed. Therefore, the method to search from multiple start nodes is a special case of our method, where the number of middle layer is the same as the number of start nodes. The experimental results on multiple datasets show that our HSSG method presents an obviously better search performance, which presents a good applied prospect.

The main contributions of this article include the following four aspects:

- (1) Aiming at the problem that the search speed is slow under the condition of large amount of data on the MSNET, this article proposes a solution to solve the problem of using hierarchical structure on the MSNET. According to our theoretical analysis on the feasibility, the search time is reduced by reducing the number of search steps.
- (2) To use the method in practice, we design the **Hierarchical Satellite System Graph (HSSG)** method based on the **Satellite System Graph (SSG)** method. We describe the index algorithm and search algorithm in detail and theoretically analyze the time complexity on both the two algorithms. Due to the proposed hierarchical structure, the index algorithm can be distributedly run, which further expands its application prospects.
- (3) To comprehensively examine the performance of ANNS algorithms in more complex real situations, we build two ANNS datasets containing 5,000,000 nodes. These two datasets are released for further research purpose.
- (4) We verify the effectiveness and efficiency of HSSG on multiple datasets by experiments, especially on the condition of large dataset and densely distributed data.

The rest of this article is organized as follows: In Section 2, we give a formal definition of the **Approximate Nearest Neighbor Search (ANNS)** problem and review the various solutions of this problem and their advantages and disadvantages. Section 3 elaborates on the idea of hierarchical MSNET and the index and query algorithm of HSSG and analyzes the time complexity. In Section 4, we verify the performance of the HSSG in a variety of practical situations through experiments on multiple datasets. Section 5 summarizes the whole article.

2 BACKGROUND

In this section, we mainly introduce the definition of the ANNS problem and briefly describe the existing methods.

2.1 Problem Definition

In below text, we use E^d to represent the d -dimension Euclidean space and $\sigma(p, q)$ to represent the distance between two points p and q in the space. The definition of Nearest Neighbor Search problem is given below:

Definition 1 (Nearest Neighbor Search Problem). Given a finite point set S containing N points defined in E^d , preprocess S , such that for any query point q , the point p that is closest to point q in the point set S can be returned effectively.

Since finding the accurate nearest point usually requires a lot of time, people are more concerned with the **Approximate Nearest Neighbor Search (ANNS)** problem. With the cost of an acceptable reduction in search precision, the algorithm can return an approximate solution with a faster speed. Formally, the ANNS problem is defined as follows:

Definition 2 (ϵ -Approximate Nearest Neighbor Search Problem). Given a finite point set S containing N points defined in E^d , preprocess S , such that for any query point q , a point p with

$$\sigma(p, q) < (1 + \epsilon)\sigma(p, r) \quad (1)$$

can be returned effectively, where r is the nearest point in set S .

The NNS and ANNS problems can be easily generalized to the condition where a set of points is returned, where multiple (approximate) nearest K points are searched. In practical applications, people usually do not calculate the specific value of ϵ in the ϵ -ANNS problem, but directly use the precision as the evaluation metric of algorithm performance. Formally, for a given query point q , the set consists of K points returned by a ANNS algorithm is R , while the set consists of K nearest points is R' , the precision can be calculated by

$$\text{precision}(R') = \frac{|R' \cap R|}{|R'|} = \frac{|R' \cap R|}{K}. \quad (2)$$

In this formula, a higher precision means that the ϵ in ANNS is smaller, which means that the search performance is better.

2.2 Related Work

The existing ANNS methods can be divided into two categories according to whether it depends on a graph structure. We will briefly introduce these two types of methods in this section.

2.2.1 Non-graph-based Method. Non-graph-based methods mainly include tree-based methods and hash-based methods. The tree-based methods are classic algorithms for ANNS task. The main idea is to partition the data space from top to bottom to build an index, e.g., grid-file [24], K-D-B-tree [26], KD-tree [27], and so on. Some other methods directly split the data from the bottom up for indexing, e.g., R-Tree [13], R*-Tree [4], and so on. Its main idea includes dividing the data space from top to bottom to build an index. Commonly, the tree-based methods face a serious dimensional disaster problem. With the dimension of data increases, the number of nodes that need to be visited during search increases rapidly, causing a decrease in the performance of the model.

The **Location-Sensitive Hashing (LSH)** algorithm is a classic hash-based algorithm to solve the ANNS problem [12]. This algorithm first encodes the input points and then maps the input points to the new space through LSH. Due to the characteristics of the LSH algorithm, compared to the nodes farther in the original space, the closer nodes have a greater probability of being mapped to a same position in the new space. Therefore, the approximate nearest neighbor of a query point can be found via multiple mappings. Based on a similar idea, there are many following methods proposed, e.g., **Spectral Hashing (SH)** [28], **Binary Reconstructive Embeddings**

(BRE) [17], **Kernelized locality-sensitive hashing (KLSH)** [18], **Random-Projection-based Locality Sensitive Hashing (RPLSH)** [5]. A novel data-sensitive indexing and query processing framework by optimizing the I/O efficiency is proposed in Reference [21] and improves the performance in I/O efficiency and search accuracy. Nevertheless, the hash-based method usually faces the problem of lower accuracy when the hash code is short, and it is difficult to use the hash table for pruning search when the hash code is long, so it is limited in practical situations.

2.2.2 Graph-based Method. Graph-based methods usually use the graph structure as the index of a given point set and then use a search method similar to the A^* algorithm to search on the graph. Therefore, the main difference of the graph-based algorithm is the difference in the index process. Generally speaking, the smaller the average degree of the constructed index graph is, the fewer the number of steps required to reach the target point, and the faster the search on the graph will be. Classical methods include Delaunay diagram [3, 19], monotonic search network [6], and so on. The search on these graphs can ensure that each step is close to the target point, so there is a good theoretical guarantee for the number of search steps. However, the requirements of these methods are too strict, leading to a high time complexity when building graphs. The current mainstream method is to use the optimized approximation of these algorithms to achieve a similar search performance in a shorter building time.

The thought behind the **k -Nearest Neighbor (kNN) Graph** [7] is “the neighbors of neighbors are also likely to be neighbors.” It connects each node directly to its k nearest neighbor nodes, which is an approximation to the Delaunay graph, which reduces the average degree in a high-dimensional space while keeping a few number of search steps. The hill-climbing search strategy can be used to optimize in searching [14], where the list of visited nodes can help the search process to find neighboring nodes that are closer to the target node. Based on it, Efanna [9] and IEH [16] algorithms have achieved further improvements. These two algorithms use the hash method and random KD-tree to find a better initial node when searching, thus speed up the search. However, it also leads to a larger index file and a more complicated indexing process.

The idea of **Navigable Small World (NSW)** [22] is derived from the theory of small world networks in social network research. This algorithm simultaneously achieves an approximation of the small world networks and the Delaunay graphs. Because the graph retains some connections with remote nodes when retaining the edges of neighboring nodes, the search speed is accelerated to a certain extent. However, in this algorithm, the average degree in the graph is too high, and the connectivity of the graph cannot be guaranteed, which affects its practical performance. **Hierarchical Navigable Small World Graphs (HNSW)** [23] solves these problems. This algorithm combines multiple layers of NSW graphs into a hierarchical structure and solves the connectivity problem. According to the authors’ analysis, this algorithm achieves a better logarithmic time complexity than NSW. Furthermore, HNSW is proposed to run on GPU and the algorithm is further accelerated [30]. Based on HNSW, a training strategy to adaptively determines when terminate is proposed in Reference [20] and improves the searching speed. Although both our HSSG and HNSW algorithms use a hierarchical structure, there are still some differences between these two methods. First, HSSG uses the SSG as the mapping algorithm for each layer, which is faster than the NSW algorithm used by HNSW in searching for each layer [10]. Second, HNSW randomly assigns each point to each layer according to a certain probability when constructing the map, while HSSG directly calculates the number of nodes in each layer and determines the selected node, which makes the construction process of the HSSG more stable.

Based on the MSNET, Fu Cong et al. proposed the **Monotonic Relative Neighborhood Graph (MRNG)** [11], which simultaneously guarantees the average degree of the graph and the monotonicity of the search. To solve the problem of excessively high indexing time of this algorithm,

they further proposed the **Navigating Spreading-out Graph (NSG)** [11] as an approximation of MRNG in practical problems. In this approximation, it is only necessary to ensure that there is a monotonic search path from the starting point to all nodes, so the indexing time complexity is reduced from $O(N^2 \log N + N^2 c)$ to about $O(N \log N)$, where c is the average out degree, N is the number of nodes in the dataset. Furthermore, they proposed the **Satellite System Graph (SSG)** [10], which further improves the search speed by optimizing the edge selection strategy and reduces the time required for the index process. The SSG method is currently one of the state-of-the-art methods of the approximate nearest neighbor search problem.

3 METHODOLOGY

This article proposes to use a hierarchical method to construct an MSNET, which solves the problem of too many steps are required in search. The nodes in the dataset are divided into different layers to be searched in this method. The first layer contains all nodes. The number of nodes in each layer decreases as the number of layers increases, and nodes in the subsequent layers will definitely appear in the previous layers. Therefore, when searching, the algorithm only needs to begin the searching from the last layer and continue searching in the previous layer after reaching the nearest neighbor of the current layer, until the first layer is reached. In the search process of each layer, the algorithm performs the usual k approximate nearest neighbor search, which returns the k approximate nearest neighbors of the node. Therefore, the hierarchical structure can greatly reduce the quantity of search steps required, thereby speeding up the search. The search process of the SSG [10] uses multiple navigation nodes, selecting the nearest one to start the search, which can be regarded as a special case of using a two-layer hierarchical structure to build an MSNET. Since the SSG has been verified to be the current optimal monotonic search network method [10], in this article, we choose the SSG as the single-layer search graph construction method and propose the **Hierarchical SSG (HSSG)**.

3.1 Detailed Algorithm

3.1.1 Index Algorithm. As shown in Algorithm 1, the HSSG index algorithm is composed of a single-layer SSG index algorithm successively in multiple layers. Given the dataset D that needs to be constructed and the number of the total layers L , we first determine the number of nodes in each layer and then select the nodes. The first layer contains all the nodes in D , and the number of nodes in each subsequent layer decreases according to a certain proportional coefficient $ratio_l$. The nodes in the subsequent layers are selected randomly from the previous layer until the last layer (layer L), which contains only one node. In this article, we use the same $ratio_l$ for each layer, which is $ratio_l = \sqrt[l]{size(D)}$. After that, we build an index graph for each layer separately. The index graph is a directed complete graph if the layer contains few nodes ($N \leq k$), otherwise an SSG is built in the layer as the process shown below. The first step for building the SSG is to build an approximate k -nearest neighbor graph ($k < N$), where each node is connected to its K approximate nearest neighbors. After that, starting from any $node$ in the kNN graph, we put the l nodes with the nearest distance from its neighbors and the neighbors of its neighbors into the candidate set. For each node in the candidate set, check whether adding the node into the result graph will break the following property: For any nodes b and c in the result graph, the angle between edges $(node, b)$ and $(node, c)$ is larger or equal to a predefined angle m . Add the node into the result graph if it does not break the property. Repeat the process until all nodes in the kNN graph are checked. Since each layer of the HSSG is constructed using this method and is not related to each other, this algorithm naturally has good parallelism and hence can be distributed run to accelerate the index process of the HSSG.

ALGORITHM 1: Hierarchical Satellite System Graph Indexing

Input: input dataset D , size of candidate pool l , max out-degree r , minimum angle α , number of layers L , k and other parameters for building k -nearest neighbor graph

Output: A Hierarchical Satellite System Graph G

```

1 .
2  $ratio = \sqrt[n]{size(D)}$ 
3 for  $layer$  in  $1, \dots, L$  do
4   if  $layer = 1$  then
5      $N = size(D)$ 
6      $candidateset = D$ 
7   end
8   else
9      $N = N \times ratio$ 
10     $candidateset = N$  nodes randomly selected from the previous layer
11  end
12  if  $k \geq N$  then
13    Build a directed complete graph  $G_L$  with  $candidateset$ 
14     $G[layer] = G_L$ 
15  end
16  else
17    Build an approximate  $kNN$  graph  $GL_{kNN}$ 
18     $GL = \phi$ 
19    for  $node$  in  $GL_{kNN}$  do
20       $P = \phi$  for  $neighbor\ ne$  of  $node$  do
21         $P.add(ne)$ 
22        for  $neighbor\ ne'$  of  $ne$  do
23           $P.add(ne')$ 
24        end
25        remove the redundant nodes in  $P$ 
26        if  $P.size() \leq l$  then
27          break
28        end
29      end
30      Sort  $P$  in the ascending order of the distance to  $node$ 
31      Select neighbors from  $P$  according to the definition of SSG to  $GL$ .
32    end
33     $G[layer] = GL$ 
34  end
35 end
36 for  $layer$  in  $1, \dots, L$  do
37   DFS-expanding from  $G[layer + 1]$ 
38 end
39 Return  $G$ 

```

To ensure the strong connectivity of the constructed HSSG, a depth-first search algorithm is needed to traverse the intermediate result graph as the final step. Different from the original SSG index algorithm, in HSSG, we do not need to ensure that the maps of each layer are strongly connected, but only need to ensure that all points in the next layer can be reached from the previous

ALGORITHM 2: Hierarchical Satellite System Graph Search

Input: HSSG G , query node q , dataset D , candidate pool size l
Output: k approximate nearest neighbor of q in D

```

1 for  $layer = n \dots 1$  do
2   if  $G[layer].size() < l$  then
3      $p = G[layer].nodes()$ 
4     continue
5   end
6   else
7      $results = Search-on-Graph(G[layer], p, q, 1)$ 
8      $p = results$ 
9   end
10 end
11 Return  $results$ 

```

layer. Therefore, the speed of index process can be improved without affecting the accuracy of searching.

3.1.2 Search Algorithm. The search on HSSG is shown in Algorithm 2. The search process starts from the node in the top layer. It finds the nearest nodes to the target node in the current layer and then finds the nearest nodes in the next layer with the result nodes from the last layer as the start nodes. This process is done recursively until reaching the bottom layer, where the result is exactly the result of the total process. The search algorithm in one layer uses the popularly used graph search algorithm similar to the A* algorithm [10, 11], as shown in Algorithm 3. Concretely, in the search process of each layer, we maintain a candidate pool of size l . When the algorithm runs into a layer, we initialize the candidate pool set with the start nodes. Then the following process is repeated: (a) find the nearest node of the target node in the candidate pool and put its neighbors into the candidate pool, (b) sort the nodes in the candidate pool and delete surplus nodes. Therefore, a path from the source node to the target node is generated in the process, and its length is exactly the steps needed. When all nearest neighbors of the target node are found, we cannot add any more nodes into the candidate pool, and the algorithm terminates.

3.2 Analysis on Complexity

Here, we denote the time cost of building a SSG with N nodes as $f(N)$. According to the index algorithm of HSSG, the time cost of building an L -layer HSSG is

$$f(N) + f\left(\frac{N}{ratio}\right) + f\left(\frac{N}{ratio^2}\right) + \dots + f(1), \quad (3)$$

where $ratio = \sqrt[k]{N}$. According to the analysis in Reference [10], when the parameter k in k -NN graph, size of candidate pool set, max-degree of SSG, and space dimension are fixed, the time complexity of building an SSG is $f(N) = O(N^{1.16})$. Therefore, for a given total layer L , the time complexity of building an HSSG is also $O(N^{1.16})$.

Because of the restriction on the max degree of SSG during building, the space cost of saving an SSG with adjacency matrix is $O(N)$. Therefore, the space cost of saving an HSSG with N nodes is

$$O(N) + O\left(\frac{N}{ratio}\right) + O\left(\frac{N}{ratio^2}\right) + \dots + O(1). \quad (4)$$

ALGORITHM 3: Search-on-Graph

Input: graph G , start nodes set p , query node q , candidate pool size l
Output: k approximate nearest neighbor of q in G

```

1   $i = 0$ 
2  candidate pool set  $S = p$ 
3  sort  $S$  in descending order according of the distance to  $q$ 
4  while  $i < l$  do
5       $i = \text{index of the first unchecked node in } S$ 
6      mark  $p_i$  as checked
7      for each neighbor } n \text{ of } p_i \text{ in } G do
8          if } n \text{ has not been compared then}
9               $S.add(n)$ 
10         end
11     end
12     sort  $S$  in descending order according of the distance to  $q$ 
13     if } S.size() > l then
14          $S.resize(l)$ 
15     end
16 end
17 Return the first  $k$  nodes in  $S$ .

```

For a given total layer L , the space complexity of saving an HSSG is also $O(N)$.

The time complexity of the search process on HSSG is greatly affected by the practical situation, hence it is difficult to draw accurate conclusions. According to the experimental observation presented in Reference [10], the time complexity of the search algorithm on SSG is $O(\log N)$. As the hierarchical structure reduces the needed search steps and saves time, it can be considered that the search complexity on the HSSG is also $O(\log N)$.

4 EXPERIMENTS

4.1 Datasets

To verify the effectiveness of HSSG in different situations, we do experiments on multiple datasets derived from images, text, and randomly generated data. The detailed information of the datasets is shown in Table 1. The SIFT1M and GIST1M are two popularly used ANNS datasets derived from images [15], and many literatures have performed performance evaluations on these two datasets [10, 11]. Crawl [1] is another dataset derived from image data. GloVe is a method to convert text data into vector representations, and GloVe-100 is an ANNS dataset extracted from text using this method [25]. Among the datasets, SIFT1M and GIST1M contain relatively few nodes (No. of base), and the other two datasets contain more nodes. GIST1M contains 1,000 query nodes, and the other three datasets have 10,000 query nodes.

To show the performance of ANNS algorithm in the situation of large-scale data, we build another two larger datasets containing 5,000,000 nodes within the 10-dimensional space. They are named as RAND5M and NORM5M. The nodes in RAND5M dataset are uniformly distributed and the nodes in NORM5M are normally distributed. We also randomly generated 10,000 query nodes and provide the accurate search results by brute force search in the datasets. These two datasets will be publicly released for the further research purpose.

Table 1. Information on Experimental Datasets

Datasets	Dimension	No. of base	No. of query
SIFT1M	128	1,000,000	10,000
GIST1M	128	1,000,000	1,000
Crawl	300	1,989,995	10,000
GLoVe-100	100	1,183,514	10,000
RAND5M	10	5,000,000	10,000
NORM5M	10	5,000,000	10,000

Table 2. The Indexing Time Comparison (s)

Datasets	SSG	HSSG						
		Layer = 3	4	5	6	7	8	9
SIFT1M	88.85	90.71	97.02	104.92	112.72	123.84	134.00	145.80
GIST1M	434.30	455.65	484.30	546.57	581.59	553.08	618.73	569.00
Crawl	1,079.01	1,194.49	1,236.80	1,312.67	1,394.40	1,485.23	1,594.39	1,721.88
GLoVe-100	437.12	423.82	394.94	382.69	389.26	408.75	459.43	497.20
RAND5M	300.64	298.34	305.36	319.88	337.24	365.89	390.13	417.41
NORM5M	333.09	316.22	326.32	345.46	365.50	388.43	418.49	442.71

4.2 Environment

All experiments in this article are performed on the same server. The hardware configuration of the server is shown as follows: two Xeon(R) E5-2680 v4 processors with a main frequency of 2.4 GHz, 128 GB memory; the software configuration is as follows: Ubuntu 16.04 (64-bit) Operating system, GNU g++ compiler. We use C++ language to implement the HSSG algorithm. To ensure the consistency and fairness of the comparison, we always use the -O3 compilation parameter in the experiments.

4.3 Experimental Results

In this section, we show in detail and analyze the indexing and query performance of HSSG on multiple datasets. Since the SSG used in practice can be regarded as a special case of the HSSG where $l = 2$, and the number of nodes in the top layer is 10, we continue to do experiment on the HSSG with more layers. The experimental results show that the HSSG with $l = 3$ is sufficient to achieve obvious performance superiority.

4.3.1 Index Performance. To verify the index performance of HSSG, we have performed the indexing time and space comparison of HSSG and the ordinary SSG on multiple datasets. The results are shown in Table 2 and 3. Experimental data shows that when the number of layers is small, the additional time and space overhead brought by the hierarchical structure is extremely small, and even in some special cases, the indexing time is less than that of SSG. When the number of layers is large, the additional time and space consumption of building a HSSG is relatively obvious, but it usually does not exceed twice the time required to build a single-layer SSG. In practice, the index only needs to be built once for long-term use, so this slight increase in index overhead caused by the hierarchical structure is acceptable.

4.3.2 Search Performance. Figure 2 shows the performance comparison of search on 3–9 layer HSSG on multiple datasets. In the figure, the horizontal axis represents the precision of the query, and the vertical axis is the number of queries processed per second, representing the speed of the

Table 3. The Index Space Comparison (MB)

Datasets	SSG	HSSG						
		Layer = 3	4	5	6	7	8	9
SIFT1M	154	160	167	178	191	206	221	238
GIST1M	95	100	104	111	119	128	138	148
Crawl	145	186	195	208	225	242	261	281
GLoVe-100	73	76	79	84	90	97	104	112
RAND5M	712	787	808	846	896	953	1,015	1,080
NORM5M	706	758	804	826	865	915	974	1,103

query. Therefore, the curve corresponding to the method in the figure is closer to the upper right, indicating that the method has a better tradeoff between accuracy and speed, that is, the better search performance. In each dataset, the search process are carried out multiples with different candidate pool size l from 150 to 1,500. The larger the candidate set size is, the slower the search speed is, but at the same time the higher the precision of the search results is, which represents the tradeoff between search search precision and search speed. It is noted that the curves corresponding to RAND5M and NORM5M are nearly vertical, which means the algorithm can reach a very high search precision with a small cost of search speed. The search on the HSSG with a layer number of 3 can usually achieve good search performance. A possible reason of the phenomenon is that the extra layer in the three-layer HSSG is crucial and important compared with the two-layer HSSG (i.e., original SSG). It is a qualitative change from two layer ($10 - N$) to three layer ($1 - \sqrt{N} - N$). On this basis, in the current actual tasks, the performance improvement obtained by continuing to increase the number of layers is small. A possible reason is that the sizes of the datasets are not large enough to show the advantages of HSSG with more layers. Taking into account the increase in indexing time caused by the increase in the number of layers (as shown in Table 2), in the follow-up comparison experiment, we uniformly use the three-layer hierarchical satellite system map.

Figure 3 shows the performance comparison between the three-layer HSSG and SSG on multiple datasets. The results show that on the two datasets SIFT1M and GIST1M with a relatively small amount of data ($n = 1,000,000$), HSSG method does not show a clear time advantage. With the gradual increase in the amount of data ($1,000,000 < n < 2,000,000$), the results on the Crawl and GLoVe-100 datasets show that the HSSG method has shown obvious advantages over the SSG method. On the RAND5M and NORM5M datasets with larger data volume ($n = 5,000,000$) and lower dimensional, the data distribution is more dense. At this time, the SSG method requires a long time consumption ($< 10^3 \text{ queries/sec}$) to achieve approximately accurate ($\text{precision} > 0.99$) query results. However, HSSG method can achieve more accurate ($> 3 \times 10^3 \text{ queries/sec}$) query results at several times the speed ($\text{precision} > 0.999$).

To further explore the reasons for the superior performance of the HSSG on the densely distributed RAND5M and NORM5M datasets and to verify that HSSG reduces the total amount of calculation, we separately record the the number of average required distance calculations of search on these two datasets on three-layer HSSG and SSG. The experimental results are shown in Figure 4. The results show that, compared to SSG, HSSG requires less distance calculation times in all situations of different candidate set pool sizes. As the size of candidate pool set increases, the distance calculation time also keeps expanding. This phenomenon indicates that the hierarchical structure used in HSSG greatly reduces the amount of calculation when searching, which is an important reason for the faster search speed on HSSG.

Fig. 2. Experiments of the comparison between HSSG with different layers on six datasets. Top right is better.

5 CONCLUSION

The MSNET method represented by SSG is one of the optimal methods for the ANNS problem. This method requires a large number of steps in the search process, which increases the computational cost and reduces the search speed. Aiming at the problem, this article proposes to use a

Fig. 3. Experiments of the comparison between three-layer HSSG and SSG on six datasets. Top right is better.

hierarchical method to construct an MSNET. Our theoretical analysis shows that this method can significantly reduce the number of steps required in searching, hence reduce the amount of calculation and makes reaching the node to be queried faster. To use this method in practical problems, this article proposes and implements the **Hierarchical Satellite System Graph (HSSG)** method. We describe in detail the index building algorithm and search algorithm of this method and analyze its time complexity. Experimental results on multiple datasets show that compared to the SSG

Fig. 4. Comparison of the numbers of average distance calculation times between three-layer HSSG and SSG for search on RAND5M and NORM5M.

method, the HSSG method reduces the amount of distance calculation during search and achieves a better search speed and accuracy, which shows a good prospect for practical applications of HSSG methods.

REFERENCES

- [1] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.* 12, 5 (2019), 461–474.
- [2] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67.
- [3] Franz Aurenhammer. 1991. Voronoi diagrams—A survey of a fundamental geometric data structure. *ACM Comput. Surv.* 23, 3 (1991), 345–405.
- [4] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*.
- [5] Deng Cai. 2021. A revisit of hashing algorithms for approximate nearest neighbor search. *IEEE Trans. Knowl. Data Eng.* 33, 6 (2021), 2337–2348.
- [6] D. W. Dearholt, N. Gonzales, and G. Kurup. 1988. Monotonic search networks for computer vision databases. In *ACSSC*, Vol. 2. IEEE, 548–553.
- [7] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*. 577–586.
- [8] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. 2001. Approximate nearest neighbor searching in multimedia databases. In *ICDE*.
- [9] Cong Fu and Deng Cai. 2016. Efanna: An extremely fast approximate nearest neighbor search algorithm based on KNN graph. *arXiv preprint arXiv:1609.07228* (2016).
- [10] Cong Fu, Changxu Wang, and Deng Cai. 2019. Satellite system graph: Towards the efficiency up-boundary of graph-based approximate nearest neighbor search. *arXiv preprint arXiv:1907.06146*.
- [11] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.* 12, 5 (2019), 461–474.
- [12] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*.
- [13] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*.
- [14] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. 2011. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *IJCAI*.
- [15] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2010), 117–128.
- [16] Zhongming Jin, Debing Zhang, Yao Hu, Shiding Lin, Deng Cai, and Xiaofei He. 2014. Fast and accurate hashing via iterative nearest neighbors expansion. *IEEE Trans. Cyber.* 44, 11 (2014), 2167–2177.

- [17] Brian Kulis and Trevor Darrell. 2009. Learning to hash with binary reconstructive embeddings. In *NeurIPS*, Vol. 22. Citeseer, 1042–1050.
- [18] Brian Kulis and Kristen Grauman. 2009. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*.
- [19] Der-Tsai Lee and Bruce J. Schachter. 1980. Two algorithms for constructing a Delaunay triangulation. *Int. J. Comput. Inf. Sci.* 9, 3 (1980), 219–242.
- [20] Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. 2020. Improving approximate nearest neighbor search through learned adaptive early termination. In *SIGMOD*.
- [21] Mingjie Li, Ying Zhang, Yifang Sun, Wei Wang, Ivor W. Tsang, and Xuemin Lin. 2020. I/O efficient approximate nearest neighbour search based on learned functions. In *ICDE*.
- [22] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Inf. Syst.* 45 (2014), 61–68.
- [23] Yu A. Malkov and Dmitry A. Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2018), 824–836.
- [24] Jürg Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. 1984. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.* 9, 1 (1984), 38–71.
- [25] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *EMNLP*.
- [26] John T. Robinson. 1981. The KDB-tree: A search structure for large multidimensional dynamic indexes. In *SIGMOD*.
- [27] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *CVPR*.
- [28] Yair Weiss, Antonio Torralba, Robert Fergus, et al. 2008. Spectral hashing. In *NeurIPS*, Vol. 1. Citeseer.
- [29] Pei-Sen Yuan, Chao-Feng Sha, Xiao-Ling Wang, and Ao-Ying Zhou. 2012. C-approximate nearest neighbor query algorithm based on learning for high-dimensional data. *Ruanjian Xuebao/J. Softw.* 23, 8 (2012), 2018–2031.
- [30] Weijie Zhao, Shulong Tan, and Ping Li. 2020. SONG: Approximate nearest neighbor search on GPU. In *ICDE*.

Received March 2021; revised June 2021; accepted September 2021