Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement

Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin, *Fellow, IEEE*

Abstract—Nearest neighbor search is a fundamental and essential operation in applications from many domains, such as databases, machine learning, multimedia, and computer vision. Because exact searching results are not efficient for a high-dimensional space, a lot of efforts have turned to approximate nearest neighbor search. Although many algorithms have been continuously proposed in the literature each year, there is no comprehensive evaluation and analysis of their performance. In this paper, we conduct a comprehensive experimental evaluation of many state-of-the-art methods for approximate nearest neighbor search. Our study (1) is cross-disciplinary (i.e., including 19 algorithms in different domains, and from practitioners) and (2) has evaluated a diverse range of settings, including 20 datasets, several evaluation metrics, and different query workloads. The experimental results are carefully reported and analyzed to understand the performance results. Furthermore, we propose a new method that achieves both high query efficiency and high recall empirically on majority of the datasets under a wide range of settings.

Index Terms—Similarity search, approximate nearest neighbor search, high-dimensional space, metric space, dense vector

1 Introduction

Nearest neighbor search (NNS) finds an object in a reference database which has the smallest distance to a query object. It is a fundamental and essential operation in applications from many domains, including databases, computer vision, multimedia, machine learning and recommendation systems.

Despite much research on this problem, it is commonly believed that it is very costly to find the *exact* nearest neighbor in high dimensional euclidean space, due to the *curse of dimensionality* [1]. Experiments showed that exact methods can rarely outperform the brute-force linear scan method when dimensionality is high [2] (e.g., more than 20). Nevertheless, returning sufficiently nearby objects, referred to as *approximate nearest neighbor search* (ANNS), can be performed efficiently and are sufficiently useful for many practical problems, thus attracting an enormous number of research efforts.

1.1 Motivation

There are hundreds of papers published on algorithms for (approximate) nearest neighbor search, but there has been few systematic and comprehensive comparisons among these algorithms. In this paper, we conduct a comprehensive experimental evaluation on the state-of-the-art approximate nearest neighbor search algorithms in the literature, due to the following needs:

1. Coverage of Competitor Algorithms and Datasets from Different Areas. As the need for performing ANNS arises naturally in so many diverse domains, researchers have come up with many methods while unaware of alternative methods proposed in another area. In addition, there are practical methods proposed by practitioners and deployed in large-scale projects such as the music recommendation system at `spotify.com` [3]. As a result, it is not uncommon that important algorithms from different areas are overlooked and not compared with. For example, there is no evaluation among Rank Cover Tree [4] (from Machine Learning), Product Quantization [5], [6] (from Multimedia), SRS [7] (from Databases), and KGraph [8] (from practitioners). Moreover, each domain typically has a *small* set of commonly used datasets to evaluate ANNS algorithms; there are very few datasets used by all these domains.

In contrast, we conduct comprehensive experiments using carefully selected representative or latest algorithms from different domains, and test *all of* them on 20 datasets including those frequently used in prior studies in different domains. Our study confirms that there are substantial variability of the performance of all the algorithms across these datasets.

- *W. Li is with the School of Information Engineering, Nanjing Audit University, Nanjing 210017, China, and also with the Centre for Artificial Intelligence, University of Technology Sydney, Sydney, NSW 2007, Australia. E-mail: Wen.Li@uts.edu.au.*
- *Y. Zhang is with the Centre for Artificial Intelligence, University of Technology Sydney, Sydney, NSW 2007, Australia and Zhejiang Gongshang University, No. 18, Xuezheng Str., Xiasha University Town, Hangzhou, China, 310018. E-mail: Ying.Zhang@uts.edu.au.*
- *Y. Sun, W. Wang, W. Zhang, and X. Lin are with the School of Computer Science and Engineering, University of New South Wales, Sydney, NSW 2052, Australia. E-mail: {yifangs, weiw, zhangw, lxue}@cse.unsw.edu.au.*
- *M. Li is with the Centre for Artificial Intelligence, University of Technology Sydney, Sydney, NSW 2007, Australia. E-mail: Mingjie.Li@student.uts.edu.au.*

Manuscript received 10 Oct. 2016; revised 12 Dec. 2018; accepted 25 Mar. 2019. Date of publication 3 Apr. 2019; date of current version 7 July 2020. (Corresponding author: Mingjie Li.)
Recommended for acceptance by K. Yi.
Digital Object Identifier no. 10.1109/TKDE.2019.2909204

2. Overlooked Evaluation Measures/Settings. An ANNS algorithm can be measured from various aspects, including (i) search time complexity, (ii) search quality, (iii) index size, (iv) scalability with respect to the number of objects and the number of dimensions, (v)robustness against datasets, query workloads, and parameter settings, (vi) updatability, and (vii) efforts required in tuning its parameters.

Unfortunately, none of the prior studies evaluates these measures completely and thoroughly.

For example, most existing studies use a query workload that is essentially the same as the distribution of the data. Measuring algorithms under different query workloads is an important issue, but little result is known. In this paper, we evaluate the performance of the algorithms under a wide variety of settings and measures, to gain a complete understanding of each algorithm (c.f., Table 3).

3. Discrepancies in Existing Results. There are discrepancies in the experimental results reported in some of the notable papers on this topic.

For example, AGH was shown to perform better than SpectralHashing in the literature [9], while the study in [10] indicates otherwise. This situation also exists between SpectralHashing and DSH [10], [11]. While much of the discrepancies can be explained by the different settings, datasets and tuning methods used, as well as implementation differences, it is always desirable to have a maximally consistent result to reach an up-to-date rule-of-the-thumb recommendation in different scenarios for researchers and practitioners.

In this paper, we try our best to make a fair comparison of several algorithms, and test them on all 20 datasets. Finally, we will also publish the source code, datasets, and other documents so that the results can be easily reproduced.

We classify popular ANNS algorithms into three categories: *Hashing-based*, *Partition-based* and *Graph-based*. The key idea of each category of the method will be introduced in Section 3.

1.2 Contributions

Our principle contributions are summarized as follows.

- Comprehensive experimental study of state-of-the-art ANNS methods across several different research areas. Our comprehensive experimental study extends beyond past studies by: (i) comparing all the methods without adding any implementation tricks, which makes the comparison more fair; (ii) evaluating all the methods using multiple measures; and (iii) we provide rule-of-the-thumb recommendations about how to select the method under different settings. We believe such a comprehensive experimental evaluation will be beneficial to both the scientific community and practitioners, and similar studies have been performed in other areas (e.g., classification algorithms [12]).
- We group algorithms into several categories (Section 3), and then perform detailed analysis on both intra- and inter-category evaluations (Sections 5 and 6). Our *data-based*analyses provide confirmation of useful principles to solve the problem, the strength and weakness of some of the best methods, and some initial explanation and understanding of why some datasets are harder than others. The experience and insights we gained throughout the study enable us to engineer a new empirical algorithm, DPG (Section 4), that achieves both high query efficiency and high recall empirically on majority of the datasets under a wide range of settings.

The rest of the paper is organised as follows. Section 2 introduces the problem definition and some constraints in this paper. Section 3 presents the related research for ANNS problem and Section 4 presents our improved ANNS approach. In Sections 5 and 6, the comprehensive experiments and the analyses are reported. Finally, we conclude this paper in Section 7.

2 Background

2.1 Problem Definition

In this paper, we focus on the case where data points are d-dimensional vectors in $\mathbb{R}^d$ and the distance metric is the euclidean distance. Exact nearest neighbor search (NNS) for a given query point is defined as returning the data point with the smallest distance from the query. A generalization of the nearest neighbor search is called k-nearest-neighbor search (k-NNS), which aims to the k closest vectors for the query.

Due to the *curse of dimensionality*, much research efforts focus on the *approximate* solution for the problem of NNS and k-NNS on high dimensional data, which means we hope the algorithms (defined as approximate NNS (ANNS) and approximate k-NNS (k-ANNS) respectively) returns as many true neighbors as possible.

2.2 Scope

The problem of ANNS on high dimensional data has been extensively studied in various literature. Over hundreds of algorithms have been proposed to solve the problem from different perspectives, and this line of research remains very active in the above domains due to its importance and the huge challenges. To make a comprehensive yet focused comparison of ANNS algorithms, in this paper, we restrict the scope of the study by imposing the following constraints.

Representative and Competitive ANNS Algorithms. We consider the state-of-the-art algorithms in several domains, and omit other algorithms that have been dominated by them unless there is strong evidence against the previous findings.

No Hardware Specific Optimizations. Not all the implementations we obtained or implemented have the same level of sophistication in utilizing the hardware specific features to speed up the query processing. We pay more attention on the query technology itself, and weaken the implementation skills. Therefore, we modified several implementations so that no algorithm uses multiple threads, multiple CPUs, SIMD instructions, hardware pre-fetching, or GPUs.

Dense Vectors. We treat the input data as dense vectors, taking no account of the specific processing for sparse data.

Support the Euclidian Distance. The euclidean distance is one of the most widely used measure on high-dimensional datasets. It is also supported by most of the ANNS algorithms.

Exact k-NN as the Ground Truth. In several existing works, each data point has a label (typically in classification or clustering applications) and the labels are regarded as the ground truth when evaluating the recall of approximate k-NN algorithms. However, The labels mostly do not exist for large data sets. In this paper, we use the exact k-NN

points as the ground truth as this works for all datasets and majority of the applications.

Prior Benchmark Studies: There are two recent ANNS benchmark studies: [13] and ann – benchmark [14]. The former considers a large number of other distance measure in addition to the euclidean distance, and the latter does not disable general implementation tricks. In both cases, their studies are less comprehensive than ours, e.g., with respect to the number of algorithms and datasets evaluated.

3 Related Work

We classify the state-of-the-art ANNS algorithms into three main categories: *Hashing-based*, *Partition-based* and *Graph-based*.

3.1 Hashing-based Methods

The algorithms belonging to this class transform data point to a low-dimensional representation, so each point could be represented by a short code (called hash code). There are two main sub-categories in this class: locality sensitive hashing (LSH) and learning to Hash (L2H).

3.1.1 Locality Sensitive Hashing

Locality sensitive hashing (LSH) is data-independent hashing approach. The LSH methods rely on a family of locality sensitive hash functions that map similar input data points (distance $< r$) to the same hash codes with higher probability than dissimilar points (distance $> cr$), so LSH methods are initial designed to solve (r,c)-ANN problem. The designing of good locality sensitive hash functions is vital for LSH-related methods. For Euclidian distance measure, a great number of hash functions are proposed [15], [16], [17], [18], [19]. Random linear projections [15], [20], [21], [22] are the most commonly used hash function to generate hash code, in which the random projection parameters are chosen from a 2-stable distribution (e.g., Gaussian distribution).

In order to achieve good search precision, several hash functions are concatenated to form a hash table, thus decreasing the collision probability for dissimilar points. While it also reduces the collision probability of nearby points, so one usually requires to construct multiple hash tables, leading to large memory cost and long query time. Hence, some heuristic methods [23], [24], [25] are presented to check more hash buckets which may contain the nearest neighbor or the candidates near the query point, so as to increase the search quality without increasing the number of hash tables.

Because the hash tables are constructed before searching, the points collided with the query in a part of hash functions in one hash table are neglected although they are likely near. Hence, instead of using "static" compound hash functions to construct hash table before searching, some recent LSH-based methods (e.g., C2LSH [26], LazyLSH [27], QALSH [25]) employ dynamic collision counting scheme for more efficient searching.

LSH-based methods are widely studied by the theory community and enjoy the sound probabilistic theoretical guarantees on query result quality (based on distance ratios), efficiency, and index size even in the worst case. Note that the soundness of theoretical guarantees of LSH algorithms relies on the assumption that: given two data points, the hash functions are selected randomly and independently [28].

3.1.2 Learning to Hash (L2H)

Learning to Hash methods fully make use of the data distribution to generate specific hash functions, leading to higher efficiency at the cost of relinquishing the theoretical guarantees. The main methodology of Learning to Hash methods is similarity-preserving, so that the ANN relationships between the data points in the original space could be maximally preserved in the hash coding space.

According to the difference of the optimization objective design to preserve similarity, the learning to hash algorithms could be grouped into the following classes: pairwise-similarity persevering class [9], [29], [30], [31], [32], multiwise-similarity persevering class [33], [34], implicitly-similarity persevering class [35], [36] and quantization class [5], [37], [38]. More related references could be found in [39], [40], [41]. Besides similarity persevering criterion, most of the hashing methods require the codes to be balance and uncorrelated.

Many literature indicates that the quantization algorithms are more efficient than other learning to hash methods. The quantization-based methods seek to minimize the quantization distortion (equal to the sum difference of each data point and its approximation). Product Quantization (PQ) [5] is a popular methods for ANNS, which decomposes the original vector space into the Cartesian product of M lower dimensional subspaces, and performs vector quantization [42] in each subspace separately. A vector is then represented by a short code composed of its subspace quantization indices. Recently, there are many extensions are proposed to improve the performance of PQ for indexing step [6], [43], [44], [45], [46] and searching step [46], [47], [48], [49]. For example, Optimized Product Quantization (OPQ) [6] use pre-rotation to further minimize the quantization distortion. Additive Quantization (AQ) [50] and Composite Quantization (CQ) [51] are the generalization of PQ and represent a vector as the sum of M D-dimensional vectors where D is equal to the dimension of input data.

Benefiting from the development of deep neural network, deep hashing methods that employ deep learning are widely studied in recent years. As we doesn't use label information, we only introduce unsupervised deep hashing methods in this paper. More evaluation of supervised deep hashing algorithms could be found in [52]. Semantic hashing [53] is the first work on using deep learning techniques for hashing, which builds a multi-layers Restricted Boltzmann Machines (RBM) to learn compact binary codes for text and documents. In order to learn the binary codes, most of deep hashing methods design a sign activation layer to produce binary codes and minimize the loss between the compact real-valued code and the learned binary vector [54], [55], [56], [57]. Another solution is to reconstruct the original data. For example, [58], [59] use autoencoder as hidden layers. Thanh-Toan etc. [60] propose to constrain the penultimate layer to directly output the binary.

Because hashing methods must obtain the binary code from the output of hash functions, the binary constraint optimization problem is an NP-hard problem. To ease the optimization, most of the hashing methods adopt the following "relaxation + rounding" approach, which makes the binary codes are suboptimal. For handling this problem, some discrete optimization methods are developed [30], [61], [62].

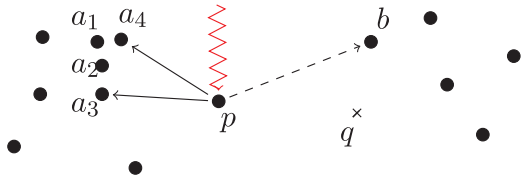


Fig. 1. Speedup with recall of 0.8.

3.2 Partition-based Methods

Methods in this category can be deemed as dividing the entire high dimensional space into multiple disjoint regions. Let the query q be located in a region r_q, then its nearest neighbors should reside in r_q or regions *near* r_q.

The partition process often carry out in a recursive way, so partition-based methods are best represented by a tree or a forest. Based on the way to partition, there are mainly three kinds: pivoting, hyperplane and compact partitioning schemes. Pivoting methods divide the points relying on the distances from the point to some pivots (usually randomly chosen). Algorithms in this class contain VP-Tree [63], Ball Tree [64] etc. Hyperplane partitioning methods recursively divide the space by the hyperplane with random direction (e.g., Annoy [3], Random-Projection Tree [65]) or axis-aligned separating hyperplane (e.g., Randomized KD-trees) [66], [67]. Compact partitioning algorithms either divide the data into clusters [68] or create possibly approximate Voronoi partitions [69], [70] to exploit locality.

3.3 Graph-based Methods

Graph-based methods construct a proximity graph where each data point corresponds to a node and edges connecting some nodes define the neighbor-relationship. The main idea of these methods is *a neighbor's neighbor is likely to also be a neighbor*. The search could be efficiently performed by iteratively expanding neighbors' neighbors in a best-first search strategy following the edges.

According to the difference of the graph structures, graph-based methods are divided into several classes. The first one tries to build an exact or approximate k-nearest neighbor graph that records the top-k nearest neighbors for each node. Especially for high dimensional space, the approximate k-nearest neighbor graph construction methods were widely studied recently [71], [72], [73], [74], [75], [76]. With the support of kNN graph, nearest neighbor search is conducted by hill-climbing strategy [77] and usually assigns some random data points as initial enter points, which is easy to get trapped in local optimal. In order to obtain better starting points, some schemes are proposed to locate some initial entries quickly. For example, Wei Dong uses LSH to generate the initial points in his thesis [78]. IEH [79] and Efanna [80] employ hashing based methods and Randomized KD-tree for initialization.

The second class is a proximity graph called navigable Small World graph (SW-graph) [81]. The SW-graph is an undirected graph, which contains an approximation of the Delaunay graph and has long-range links together with the small-world navigation property. Thus, it is more efficient for the nearest neighbor search. Yury et al. [82] proposed NSW method to build an SW-graph by iteratively inserted the points where each point is linked to some nodes selected by a greedy search algorithm from the graph in building. But the degree of NSW is too high to be efficient and there also exist connectivity problems in it. HNSW [83] is an Extension of NSW. It generates a multi-layer proximity graph with different scales and uses a heuristic to preferentially select the neighbors in diverse directions. HNSW is one of the most efficient ANNS algorithms so far.

4 Diversified Proximity Graph

The experience and insights we gained from this study enable us to engineer a new method, Diversified Proximity Graph (DPG), which constructs a different proximity graph to achieve better and more robust search performance.

4.1 Motivation

The construction of K-NN graph mainly consider the distances of neighbors for each data point, But intuitively we should also consider the *coverage* of the neighbors. As shown in Fig. 1, the two closest neighbors of the point p are a_3 and a_4, and hence in the 2-NN graph p cannot lead the search to the NN of q (i.e., the node b) although it is close to b. Since $a_1, \ldots, a_4$ are clustered, it is not cost-effective to retain both a_3 and a_4 in the K-NN list of p. This motivates us to consider the direction diversity (i.e., angular dissimilarity) of the K-NN list of p in addition to the distance, leading to the diversified K-NN graph. Regarding the example, including a_3 and b is a better choice for the K-NN list of p.

Now assume we have replaced edge (p, a_4) with the edge (p, b) (i.e., the dashed line in Fig. 1), but there is still another problem. As we can see that there is no incoming edge for p because it is relatively far from two clusters of points (i.e., p is not 2-NN of these data points). This implies that p is isolated, and two clusters are disconnected in the example. This is not uncommon in high dimensional data due to the phenomena of "hubness"[84] where a large portion of data points rarely serve as K-NN of other data points, and thus have no or only a few incoming edges in the K-NN graph. This motivates us to also use the reverse edges in the diversified K-NN graph; that is, we keep an bidirected diversified K-NN graph as the index, and we name it *Diversified Proximity Graph* (DPG).

4.2 Diversified Proximity Graph

The construction of DPG is a diversification of an existing K-NN graph, followed by adding reverse edges. Given a reference data point p, the similarity of two points x and y in p's K-NN list $\mathcal{L}$ is defined as the angle of $\angle xpy$, denoted by $\theta(x, y)$. We aim to choose a subset of κ data points, denoted by $\mathcal{S}$, from $\mathcal{L}$ so that the average angle between two points in $\mathcal{S}$ is maximized; or equivalently, $\mathcal{S} = \arg\min_{\mathcal{S}\subseteq\mathcal{N},|\mathcal{S}|=\kappa} \sum_{o_i,o_j\in\mathcal{S}} \theta(o_i, o_j)$.

The above problem is NP-hard [85]. Hence, we design a simple greedy heuristic. Initially, $\mathcal{S}$ is set to the closest point of p in $\mathcal{L}$. In each of the following $\kappa - 1$ iterations, a point is moved from $\mathcal{L} \setminus \mathcal{S}$ to $\mathcal{S}$ so that the average pairwise angular similarity of the points in $\mathcal{S}$ is minimized. Then for each data point u in $\mathcal{S}$, we include both edges (p, u) and (u, p) in the diversified proximity graph. The time complexity of the diversification process is $O(\kappa^2 Kn)$ where n is the number of data points, and there are totally at most $2\kappa n$ edges in the diversified proximity graph.

It is critical to find a proper K value for a desired κ in the diversified proximity graph as we need to find a good

trade-off between diversity and proximity. In our empirical study, the DPG algorithm usually achieves the best performance when $K = 2\kappa$. Thus, we set $K = 2\kappa$ for the diversified proximity graph construction. This greedy algorithm has the time complexity of $O(\kappa^2 Kn)$. We actually implemented a simplified version whose complexity is $O(K^2 n)$, which has only slightly worse performance than the full greedy version, but significantly fewer diversification time (with full details in Appendix D, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/TKDE.2019.2909204).

Note that the search process of the DPG is the same as that of KGraph.

5 Experiments

5.1 Experimental Setting

5.1.1 Compared Algorithms

We test 19 representative existing ANNS algorithms from three categories and our proposed diversified proximity graph (DPG) method. The detailed description of the test algorithm can be found in A. All of the modified source code used in this paper are public available on GitHub [86].

(1) LSH-based Methods. We evaluate Query-Aware LSH [87] (QALSH[1], PVLDB'15), SRS [7] (SRS[2], PVLDB'14) and FALCONN [19] (FALCONN[3],NIPS'15) in this class.

(2) L2H-based Methods. We evaluate Scalable Graph Hashing [88] (*SGH*[4], IJCAI'15), Anchor Graph Hashing [9] (*AGH*[5], ICML'11) and Neighbor-Sensitive Hashing [10] (*NSH*[6], PVLDB'15) from binary-encoded learning to hash methods. In order to do non-exhaustive search, We organize the hash codes with the hierarchical clustering tree [67]. The comparisons with linear scan search are given in Appendix E, available in the online supplemental material.

Selective Hashing [89] (*SH*[7], KDD'15) is an extension of LSH, which considers the local density of each point. Selective Hashing performs hash table lookup to search. We classify Permutation-based methods into L2H-based Methods. Permutation-based methods encode each point based on the order to some random selected pivots. Neighborhood APProximation index [13] (NAPP[8] PVLDB'15) is one of the most efficient Permutation-based methods, which relies on inverted index for searching.

We also evaluate Optimal Product Quantization[6] (OPQ[9], TPAMI'14) with non-exhaustive search, which is implemented by the inverted multi-indexing [10] [47].

Composite Quantization [51] (*CQ*[11], ICML'11) conducts linear scan search and a lookup table is used to pre-compute the distances from the query to the codewords of each subquantizer.

(3) Partition-based Methods. We evaluate FLANN[12][67] (TPAMI'14), *FLANN-HKM*, *FLANN-KD*, Annoy[13] and an advanced Vantage-Point tree [90] (VP − tree[8], NIPS'13) in this class.

(4) Graph-based Methods. We evaluate Small World Graph [82] (SW[8], IS'14), Hierarchical Navigable Small World [83](HNSW[8], CoRR'16), K-NN graph [8], [74] (KGraph[14], WWW'11), Rank Cover Tree [4] (RCT[15], TPAMI'15), and our Diversified Proximity Graph (DPG[15]). Note that though the tree structure is employed by RCT, the key idea of RCT belongs to connect the nearest nodes between each adjacent layer. Thus, we divide RCT into Graph-based Methods.

We use the implementation of NAPP, VP − tree, SW and HNSW from the Non-Metric Space Library (NMSLIB). We carefully tuned the hyper-parameters for each algorithm and each dataset. More detailed could be found in Appendix C, available in the online supplemental material. Considering of the comparison fairness, we would like to focus on the algorithm perspective of the existing methods. we turned off all hardware-specific optimizations in the implementations of the methods. Specifically, we disabled distance computation using SIMD and multi-threading in KGraph, `-ffast-math` compiler option in Annoy, multi-threading in FLANN, and distance computation using SIMD, multi-threading, prefetching technique implemented in the NMSLIB, i.e., SW, NAPP, VP − tree and HNSW). In addition, we disabled the optimized search implementation used in HNSW .

Computing Environment. All C++ source codes are complied by g++ 4.7, and MATLAB source codes (only for index construction of some learning to hash algorithms) are compiled by MATLAB 8.5. All experiments are conducted on a Linux server with Intel Xeon 8 core CPU at 2.9 GHz, and 32 G memory.

5.1.2 Datasets and Query Workload

We deploy 18 real datasets used by existing works which cover a wide range of applications including image, audio, video and text. We also use two synthetic datasets. Table 1 summarizes the characteristics of the datasets including the number of data points, dimensionality, *Relative Contrast* (RC [91]), *local intrinsic dimensionality* (LID [92]), and data type. RC indicates the ratio of the mean distance and NN distance for the data points, and smaller RC value implies harder dataset. LID calculates the local intrinsic dimensionality and a higher LID value implies harder dataset. We mark the first four datasets in Table 1 with asterisks to indicate that they are "hard" datasets compared with others according to their RC and LID values.

Query Workload. Following the convention, we randomly remove 200 data points as the query points for each dataset. The average performance of the k-NN searches is reported. In this paper, k is equal to 20 by default. We evaluate the impact of k in Appendix H, available in the online supplemental material.

1. http://ss.sysu.edu.cn/~fjl/qalsh/qalsh_1.1.2.tar.gz
2. https://github.com/DBWangGroupUNSW/SRS
3. https://github.com/FALCONN-LIB/FALCONN
4. http://cs.nju.edu.cn/lwj
5. http://www.ee.columbia.edu/ln/dvmm/downloads
6. https://github.com/pyongjoo/nsh
7. http://www.comp.nus.edu.sg/~dsh/index.html
8. https://github.com/searchivarius/nmslib
9. http://research.microsoft.com/en-us/um/people/kahe
10. http://arbabenko.github.io/MultiIndex/index.html
11. https://github.com/hellozting/CompositeQuantization
12. http://www.cs.ubc.ca/research/flann
13. https://github.com/spotify/annoy
14. https://github.com/aaalgo/kgraph
15. https://github.com/DBWangGroupUNSW/nns_benchmark

TABLE 1
Dataset Summary

Name	n ($\times 10^3$)	d	RC	LID	Type
Nus*	269	500	1.67	24.5	Image
Gist*	983	960	1.94	18.9	Image
Rand*	1,000	100	3.05	58.7	Synthetic
Glove*	1,192	100	1.82	20.0	Text
Cifa	50	512	1.97	9.0	Image
Audio	53	192	2.97	5.6	Audio
Mnist	69	784	2.38	6.5	Image
Sun	79	512	1.94	9.9	Image
Enron	95	1,369	6.39	11.7	Text
Trevi	100	4,096	2.95	9.2	Image
Notre	333	128	3.22	9.0	Image
Yout	346	1,770	2.29	12.6	Video
Msong	922	420	3.81	9.5	Audio
Sift	994	128	3.50	9.3	Image
Deep	1,000	128	1.96	12.1	Image
Ben	1,098	128	1.96	8.3	Image
Gauss	2,000	512	3.36	19.6	Synthetic
Imag	2,340	150	2.54	11.6	Image
UQ-V	3,038	256	8.39	7.2	Video
BANN	10,000	128	2.60	10.3	Image

5.2 Evaluation Measures

For each algorithm, we retrieve N points based on its searching process and then rerank these candidates using original features. The search quality is measured using recall, precision, accuracy and mAP. The *recall* is the ratio of the true nearest items in the retrieved N items to k. The *precision* is defined as the ratio of the number of retrieved true items to N. F-score (F1 score) is the harmonic mean of precision and recall: $F1 = 2 * precision * recall/(precision + recall)$. Then *mean average precision(mAP)* is computed as the mean of average precisions over all the queries. Accuracy is equal to $\sum_{i=0}^{i=k} \frac{dist(q,kANN(q)[i])}{dist(q,kNN(q)[i])}$, where q is a query, $kNN(q)[i]$ is q's ith true nearest neighbor, and $kANN(q)[i]$ is ith nearest neighbor estimated by one ANNS algorithm.

The search efficiency is usually measured as the time taken to return the search results for a query. For most of the algorithms (expect for Graph-based methods), we could vary the number of the retrieved points N to get different pair of recall/precision/accuracy and its corresponding search time. Since exact k-NN can be found by a brute-force linear scan algorithm, we use its query time as the baseline and define the *speedup* as $\frac{\bar{t}}{t'}$, where $\bar{t}$ is query time for linear scan and t' is the search time at a specific recall or N. For instance, we come up with a speedup 10 if an algorithm takes 1 second while the linear scan takes 10 seconds.

In addition to evaluating the search performance, we also evaluate other aspects such as index construction time, index size, index memory cost and scalability.

5.3 Comparison with Each Category

In this subsection, we evaluate the trade-offs between speedup and recall of all the algorithms on Sift and Notre data in each category. Given the large number of algorithms in the hashing-based category, we evaluate them in LSH-based and learning to hash-based subcategories separately. The goal of this round of evaluation is to select several algorithms from each category as the representatives in the second round evaluation (Section 5.4).

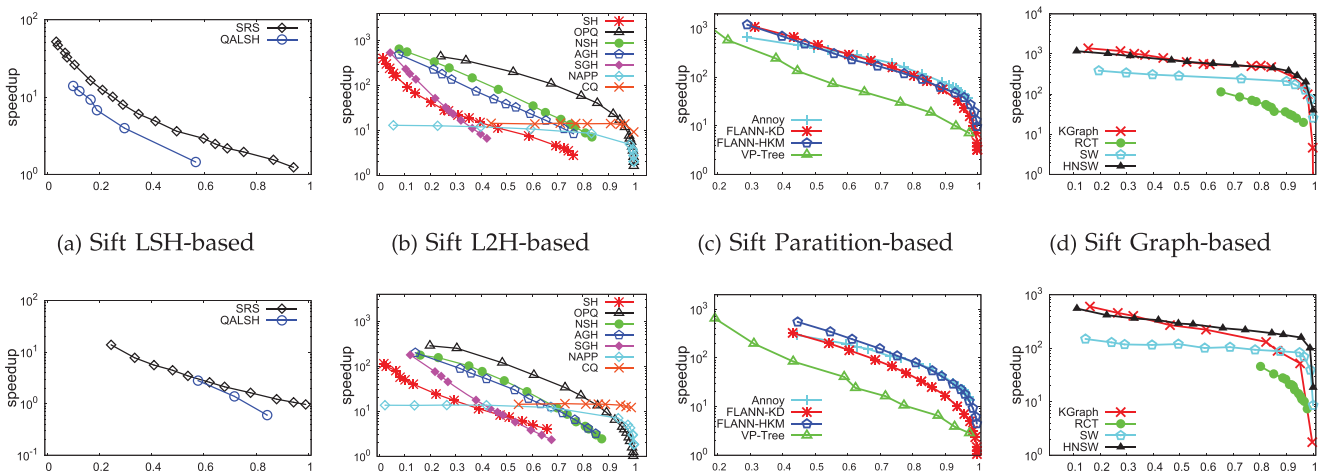
5.3.1 LSH-based Methods

Figs. 2a and 2e plot the trade-offs between the speedup and recall of two most recent data-independent algorithms SRS and QALSH on Sift and Notre . Note that, as FALCONN doesn't provide theoretical guarantee on L2 distance, we evaluate it in the second round.

As both algorithms are originally external memory based approaches, we evaluate the speedup by means of the total number of pages of the dataset divided by the number of pages accessed during the search. It shows that SRS consistently outperforms QALSH, and the similar trend is observed on other datasets. Thus, SRS is chosen as the representative in the second round evaluation where a cover-tree based in-memory implementation will be used.

5.3.2 Learning to Hash-based Methods

We evaluate seven learning to hash based algorithms including OPQ, NAPP, SGH, AGH, NSH, SH and CQ. Figs. 2b and



(a) Sift LSH-based (b) Sift L2H-based (c) Sift Paratition-based (d) Sift Graph-based

(e) Notre LSH-based (f) Notre L2H-based (g) Notre Partition-based (h) Notre Graph-based

Fig. 2. Speedup versus recall on $Sift$ and $Notre$ for each category.

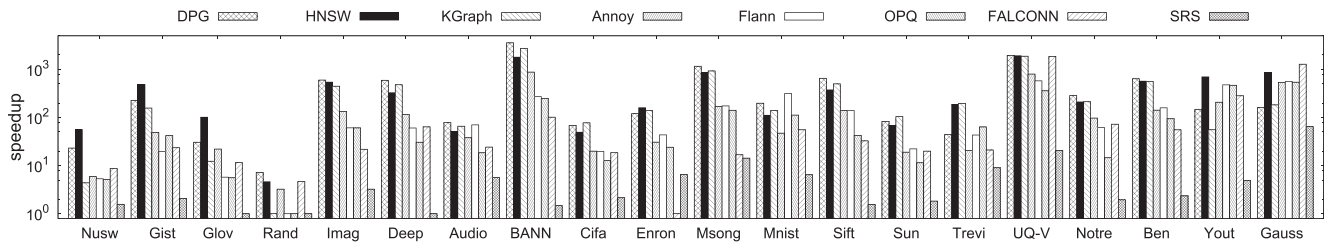


Fig. 3. Speedup with recall of 0.8.

2f demonstrate that, of all methods, the search performance of OPQ beats other algorithms by a big margin. Due to the linear scan search employed in the CQ, it shows a poor performance in terms of speedup@recall. In fact, CQ is very competitive in recall@N, but the indexing time of CQ is extremely high. So its hard to apply in practice.

For most of datasets, SelectiveHashing has the largest index size because it requires multiple long hash tables to achieve high recall. The index time value of OPQ has a strong association with the length of the sub-codeword and dimension of the data point. Nevertheless, the index construction time of OPQ still turns out to be very competitive compared with other algorithms in the second round evaluation. Therefore, we choose OPQ as the representative of the learning to hash based methods.

5.3.3 Partition-based Methods

We evaluate three algorithms in this category: FLANN, Annoy and $VP-tree$. To better illustrate the performance of FLANN, we report the performance of both randomized kd-trees and hierarchical k-means tree, namely $FLANN-KD$ and $FLANN-HKM$, respectively. Note that among 20 datasets, the randomized kd-trees method ($FLANN-KD$) is chosen by FLANN in five datasets: Enron, Trevi, $UQ-V$, BANN and Gauss . The linear scan is used in the hardest dataset: Rand, and the hierarchical k-means tree ($FLANN-HKM$) is employed in the remaining 14 datasets.

Figs. 2c and 2g show that Annoy and $FLANN-HKM$ have good performance on both datasets. For all datasets, Annoy, $FLANN-HKM$ and $FLANN-KD$ can obtain the highest performance on different datasets.

As the search performance of $VP-tree$ is not competitive compared to FLANN and Annoy under all settings, it is excluded from the next round evaluation.

5.3.4 Graph-based Methods

In the category of Graph-based methods, we evaluate four existing techniques: KGraph, SW, HNSW and RCT . Figs. 2d and 2h show that the search performance of KGraph, SW and HNSW substantially outperforms that of RCT on Sift and Notre . Because HNSW is an improvement version of SW and can achieve better performance on all datasets, we discard SW from the next round evaluation.

Although the construction time of KGraph and HNSW are relatively large, due to the outstanding search performance, we choose them as the representatives of the graph-based methods. Note that we delay the comparison of DPG to the second round.

5.4 Second Round Evaluation

In the second round evaluation, we conduct comprehensive experiments on *eight* representative algorithms: SRS, FALCONN, OPQ, FLANN, Annoy, HNSW, KGraph, and DPG .

5.4.1 Comparison of Search Quality and Time

In the first set of experiments, Fig. 3 reports the speedup of eight algorithms when they reach the recall around 0.8 on 20 datasets. Note that the speedup is set to 1.0 if an algorithm cannot outperform the linear scan algorithm. Among eight algorithms, DPG and HNSW have the best overall search performance and KGraph follows. It is shown that DPG enhances the performance of KGraph, especially on hard datasets: Nusw, Gist, Glove and Rand . As reported thereafter, the improvement is also more significant on higher recall. For instance, DPG is ranked after KGraph on four datasets under this setting (recall 0.8), but it eventually surpasses KGraph on higher recall. Overall, DPG and HNSW have the best performance for different datasets. Not surprisingly, SRS is slower than other competitors with a huge margin as it does not exploit the data distribution. Similar observations are reported in Fig. 4, which depicts the recalls achieved by the algorithms with speedup around 50.

Fig. 5 illustrates speedup of the algorithms on eight datasets with recall varying from 0 to 1. It further demonstrates the superior search performance of DPG on high recall. The overall performance of HNSW, KGraph and Annoy are also very competitive, followed by FLANN . It is shown that the

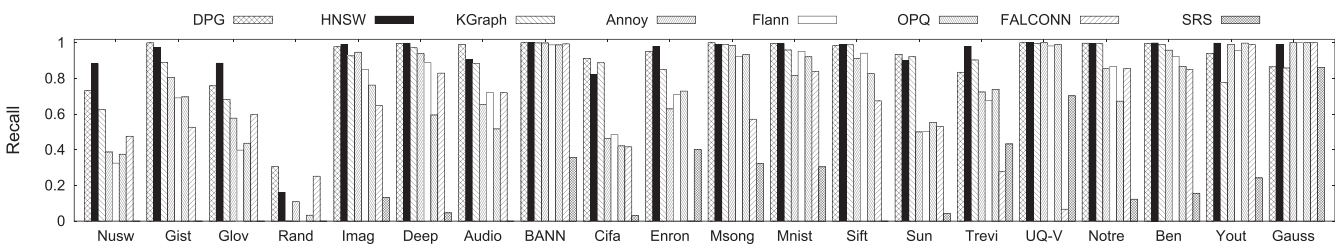


Fig. 4. Recall with Speedup of 50.

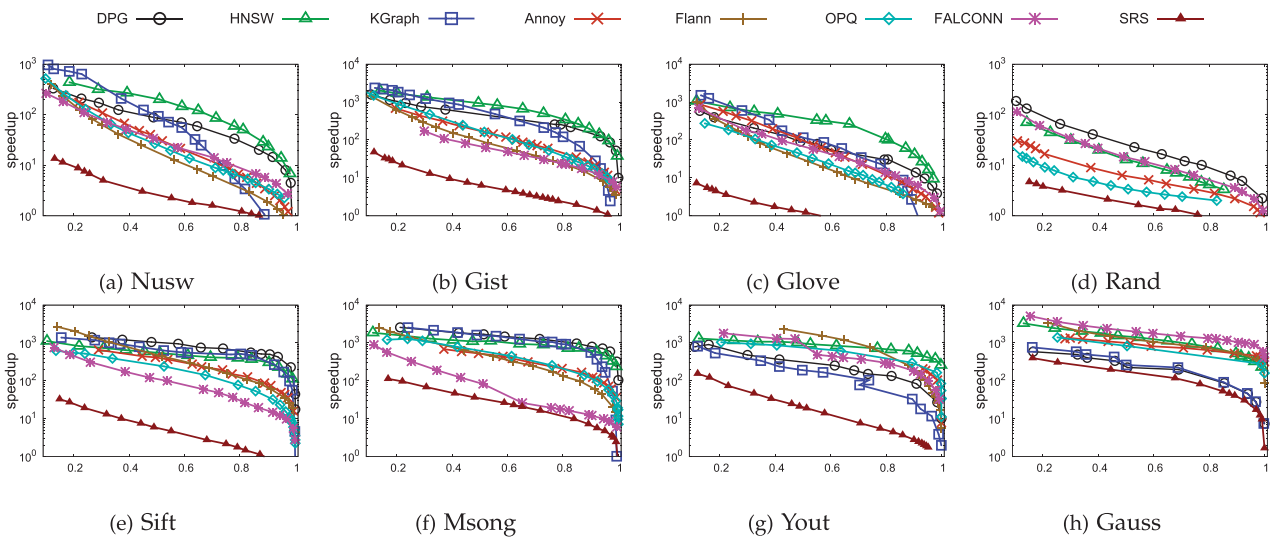


Fig. 5. Speedup versus recall on different datasets.

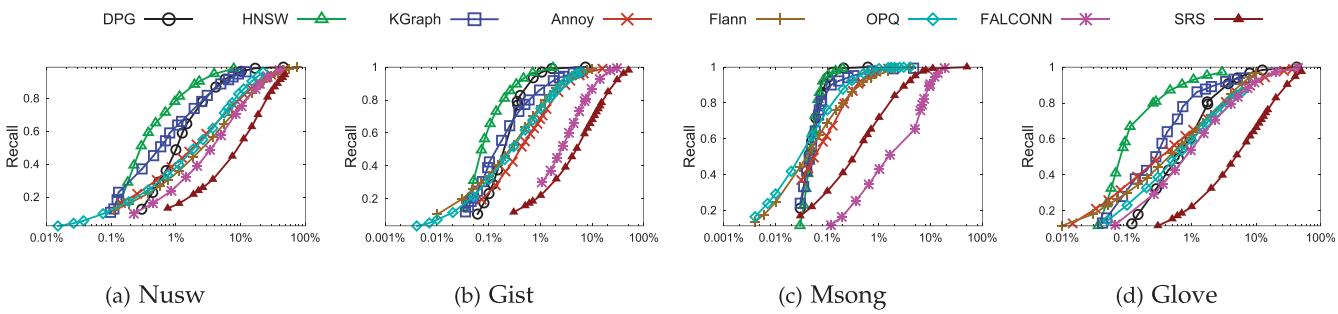


Fig. 6. Recall versus percentage of data points accessed.

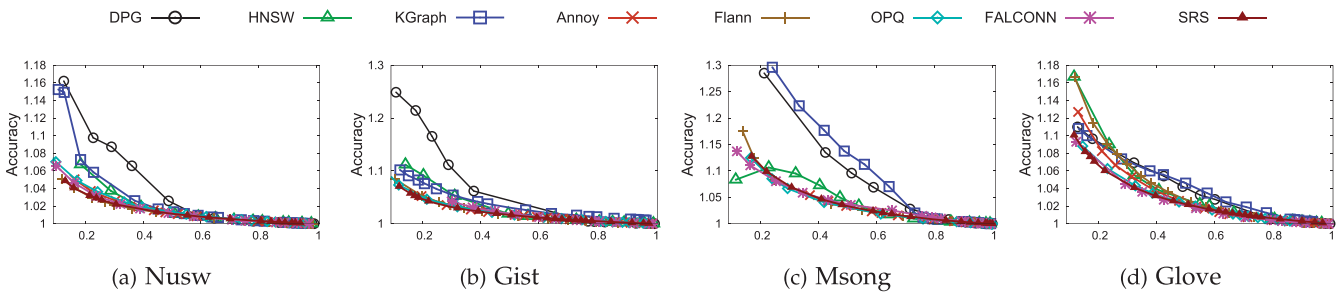


Fig. 7. Accuracy versus recall.

performance of both DPG and KGraph are ranked lower than that of HNSW, Annoy, FLANN and OPQ in Fig. 5h where the data points are clustered. As thereafter discussed in Section 6, Annoy, FLANN and OPQ essentially use the variants of k-means approach, and hence can well handle the clustered data. HNSW uses a heuristic to increase the probability of building the links between the clusters. FALCONN significantly outperforms SRS on all datasets, and it also surpasses the tree-based methods and learning to hash methods on some hard datsets, such as Glove, Nusw and random. It is worth noting that FALCONN even outperforms graph-based methods for Gauss dataset.

In Fig. 6, we evaluate the recalls of the algorithms against the percentage of data points accessed. As the search of most Graph-based methods starts from random entrance points and then gradually approaches the results while other algorithms initiate their search from the most promising candidates, the search quality of Graph-based methods is outperformed by Annoy, FLANN and even OPQ at the beginning stage. While, benefiting from HNSW's hierarchical structure, it could continue the search from the element which is the local optimum in the previous layer. The entries of each layer are carefully selected to ensure it could locate the closer points of the query quickly.

Fig. 7 shows the range search quality for a specific recall. Smaller accuracy indicates the closer of the results and the query, so the search quality is better. SRS and FALCONN which are designed for c-ANN search outperform all other algorithms.

In Figs. 8 and 9, we evaluate the precision@recall and F1@recall for each algorithm and report the mAP in Table 2. The results further verify our observations in Fig. 6. Tree-

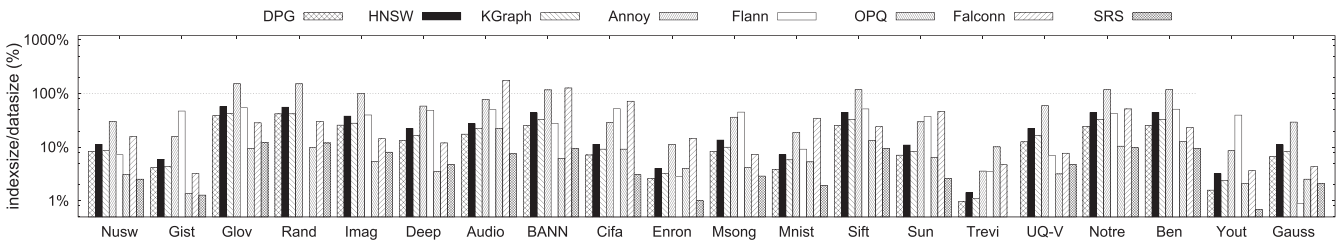


Fig. 8. Precision versus recall.

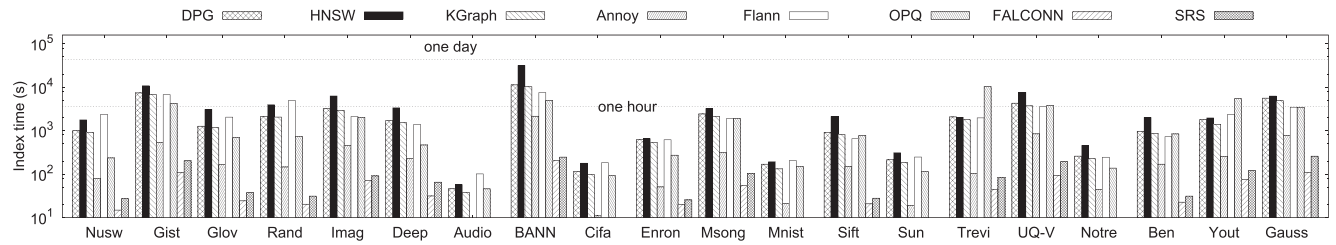


Fig. 9. F1 score versus recall.

based methods and HNSW could find the closer neighbors after retrieving small points.

5.4.2 Comparison of Indexing Cost

In addition to search performance, we also evaluate the index size, memory cost and construction time. Fig. 10 reports ratio of the index size (exclude the data points) and the data size. Except Annoy, the index sizes of all algorithms are smaller than the corresponding data sizes. The index sizes of DPG, KGraph, HNSW, SRS and FALCONN are irrelevant to the dimensionality because a fixed number of neighbor IDs and projections are kept for each data point. Consequently, they have a relatively small ratio on data with high dimensionality (e.g., Trevi). Overall, OPQ and SRS have the smallest index sizes, less than 5 percent among most of the datasets, followed by FALCONN, HNSW, DPG, KGraph and FLANN . It is shown that the rank of the index size of FLANN varies dramatically over 20 datasets because it may choose three possible index structures. Annoy needs to maintain a considerable number of trees for a good search quality, and hence has the largest index size.

Fig. 11 reports the index construction time on 20 datasets. FALCONN has the smallest index construction time among all the test algorithms. SRS ranks the second. The construction time of OPQ is related to the dimensionality because of the calculation of the sub-codewords (e.g., Trevi). HNSW, KGraph and DPG have similar construction time. Compared with KGraph, DPG doesn't spend large extra time for the graph diversification. Nevertheless, they can still build the indexes within one hour for 16 out of 20 datasets.

Fig. 12 reports the index memory cost of the algorithms on 20 datasets. OPQ needs less memory resource to build the index, so it is very efficient for large-scale datasets.

TABLE 2
mAP for Each Algorithm

Name	Nusw	Gist	Msong	Glove
DPG	0.008386	0.008586	0.01876	0.004906
HNSW	0.013122	0.012627	0.024439	0.012414
KGraph	0.008147	0.007161	0.017455	0.005867
Annoy	0.010082	0.006304	0.025659	0.019304
Flann	0.012074	0.026154	0.069939	0.044106
OPQ	0.013121	0.004076	0.072857	0.003116
FALCONN	0.001661	0.002457	0.002444	0.002584
SRS	0.010264	0.005244	0.022778	0.002837

5.5 Summary

Table 3 ranks the performance of the eight algorithms from various perspectives including search performance, index size, index construction time, index memory cost, and scalability. We also indicate that SRS is the only one with theoretical guarantee of searching quality, and it is very easy to tune the parameters for search quality and search time.

Below are some recommendations for users according to our comprehensive evaluations.

- When there are sufficient computing resources (both main memory and CPU) for the off-line index construction, and sufficient main memory to hold the resulting index, *DPG* and *HNSW* are the best choices for ANNS on high dimensional data due to their outstanding search performance in terms of robustness to the datasets, result quality, search time and search scalability.
- Except *DPG* and *HNSW*, we also recommend *Annoy*, due to its excellent search performance, construction cost, and robustness to the datasets. Moreover, it can provide a good trade-off between search performance and index size/construction time.
- If the construction time is a big concern, *FALCONN* would be a good choice because of its small construction time and good search performance.
- To deal with large scale datasets (e.g., 1 billion of data points) with moderate computing resources, *OPQ* and *SRS* are good candidates due to their small memory cost and construction time. It is worthwhile to mention that, SRS can easily handle the data points updates and have theoretical guarantee, which distinguish itself from other seven algorithms.

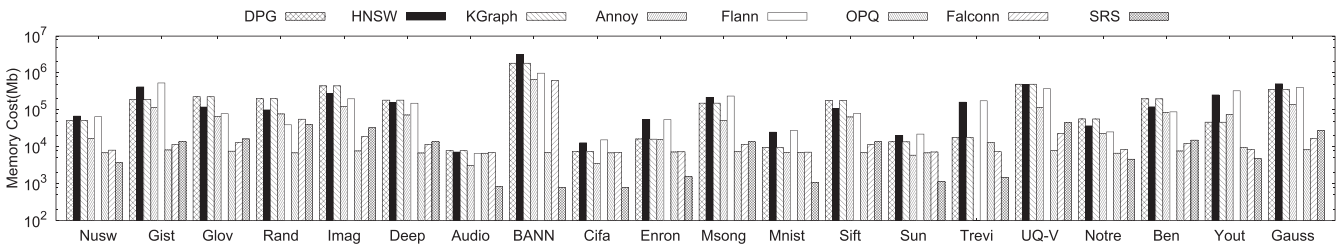


Fig. 10. The ratio of index size and data size (%).

6 Further Analyses

In this section, we analyze the most competitive algorithms in our evaluations, grouped by category, in order to understand their strength and weakness.

6.1 Space Partition-based Approach

Our comprehensive experiments show that *Annoy* and *FLANN* have the best performance among the space partition-based methods. Note that *FLANN* chooses *FLANN-HKM* in most of the datasets. In addition, *OPQ* divides the space by utilizing the production of $\mathsf{k-means}$ on M disjoint subspaces. Therefore, all three algorithms are based on *k-means* space partitioning. We define the algorithms who borrow the idea of *k-means* as *k-means-like* algorithms.

We identify that a key factor for the effectiveness of *k-means-like* algorithms is that the large number of clusters, typically $\Theta(n)$. Note that we cannot *directly* apply $\mathsf{k-means}$ with $k = \Theta(n)$ because (i) the index construction time complexity of $\mathsf{k-means}$ is linear to k, and (ii) the time complexity to identify the partition where the query is located takes $\Theta(n)$ time. Both OPQ and $\mathsf{FLANN-HKM}$ /Annoy achieve this goal indirectly by using the ideas of subspace partitioning and recursion, respectively.

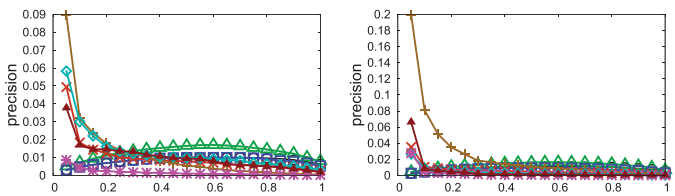
We perform experiments to understand which idea is more effective. We consider the goal of achieving *k-means-like* space partitioning with approximately the same number of non-empty partitions. Specifically, for Audio dataset, we consider the following choices: (i) Use *OPQ* with 2 subspaces and each has 256 clusters. The number of effective partitions k (i.e., non-empty partitions) is counted. For Audio dataset, $k = 18,611$. (ii) Use original *FLANN-HKM* to build a tree with roughly k leaf nodes. After carefully tuning the value of branching factor L, we found the number of total leaf nodes is 18000 when $L = 42$. (iii) Use *FLANN-HKM* with $L = 2$ and modify the stopping condition as the points in each leaf node is no larger than m. using this approach, the number of all leaf nodes k could be decided. After tuning the value of m, we get $k = 17898$ for $L = 2$. (iv) Use *k-means* directly with $k = 18,611$ to generate the partition.

Fig. 13a reports the recalls of the above choices on Audio against the percentage of data points accessed. Partitions are accessed ordered by the distances of their centers to the query. We can see that OPQ -based partition has the worst performance, followed by (modified) $\mathsf{FLANN-HKM}$ with $L = 42$, and then $L = 2$. $\mathsf{k-means}$ has the best performance, although the performance differences between the latter three are not significant. Therefore, our analysis suggests that hierarchical $\mathsf{k-means}$ -based partitioning is the most promising direction so far.

Our second analysis is to investigate whether we can further boost the search performance by using multiple hierarchical $\mathsf{k-means}$ trees. Note that Annoy already uses multiple trees and it significantly outperforms a single hierarchical $\mathsf{k-means}$ tree in $\mathsf{FLANN-HKM}$ on most of the datasets. It is natural to try to enhance the performance of $\mathsf{FLANN-HKM}$ in a similar way. We set up an experiment to construct multiple $\mathsf{FLANN-HKM}$ trees. In order to build different trees, we perform $\mathsf{k-means}$ clustering on a set of random samples of the input data points. Fig. 13b shows the resulting speedup versus recall where we use up to 50 trees. We can see that it is not cost-effective to apply multiple trees for $\mathsf{FLANN-HKM}$ on Audio, mainly because the trees obtained are still similar to each other, and hence the advantage of multiple trees cannot offset the extra indexing and searching overheads.

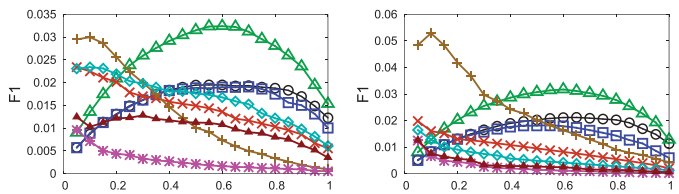
6.2 Graph-based Approach

Our first analysis is to understand why KGraph, DPG and HNSW work very well (esp. attaining a very high recall) in most of the datasets. Our preliminary analysis indicates that this is because (i) the k-NN points of a query are typically *closely connected* in the proximity graph, and (ii) most points are *well connected* to at least one of the k-NN points of a query. (ii) means there is a high empirical probability that one of the p entry points selected randomly by the search algorithm can reach one of the k-NN points, and (i) ensures that most of the k-NN points can be returned. By well connected, we mean there are many paths from an entry point to one of the k-NN point, hence there is a large probability



(a) Nusw (b) Gist

Fig. 11. Index construction time (seconds).



(a) Nusw (b) Gist

Fig. 12. Index memory cost (MB).

TABLE 3
Ranking of the Algorithms Under Different Criteria

Category	Search Performance	Index			Index Scalabiliity		Search Scalabiliity		Theoretical Guarantee
		Size	Memory	Time	Datasize	Dim	Datasize	Dim	
DPG	**1st**	4th	7th	7th	=5th	**=1st**	**=1st**	6th	No
HNSW	**1st**	7th	6th	8th	=7th	5th	**=1st**	5th	No
KGraph	3rd	5th	7th	6th	=5th	**=1st**	**=1st**	8th	No
Annoy	4th	8th	4th	3rd	=7th	6th	6th	=3rd	No
FLANN	5th	6th	5th	5th	4th	8th	**=1st**	7th	No
OPQ	6th	2nd	**1st**	4th	**1st**	=3th	5th	=3rd	No
FALCONN	7th	3rd	2nd	**1st**	2nd	=3th	8th	2nd	No
SRS	8th	**1st**	3rd	2nd	3rd	7th	7th	**1st**	**Yes**

that the "hills" on one of the path is low enough so that the search algorithm won't stuck in the local minima.

We also investigate why KGraph does not work well on some datasets and why DPG and HNSW works much better. KGraph does not work on Yout and Gauss mainly because both datasets have many well-separated clusters. Hence, the index of KGraph has many disconnected components. Thus, unless one of the entrance points used by its search algorithm is located in the same cluster as the query results, there is no or little chance for KGraph to find any near point. On the other hand, mainly due to the diversification step and the use of the reverse edges in DPG, there are edges linking points from different clusters, hence resulting in much improved recalls. Similarly, the neighbors selection strategy used in HNSW could be seen as another form of neighborhood diversification. The neighbors selection strategy and bidirectional connections also makes the edges in HNSW well linked.

For example, we perform the experiment where we use the NN of the query as the entrance point of the search on Yout . KGraph then achieves 100 percent recall. In addition, we plot the distribution of the minimum number of hops (i.e., the length of the shortest path, denoted as minHops) between a data point and any of the kNN points of a query for the indexes of KGraph and DPG on Yout and Gist in Fig. 14. We can observe that

- For KGraph, there are a large percentage of data points that cannot reach any k-NN points (i.e., those corresponding to ∞ hops) on Yout (60.38 percent), while the percentage is low on Gist (0.04 percent).
- The percentages of the ∞ hops are much lower for DPG (1.28 percent on Yout and 0.005 percent on Gist).
- There is no ∞ hops for HNSW on both datasets.
- DPG and HNSW have much more points with small minHops than KGraph, which contributes to making it easier to reach one of the k-NN points. Moreover, on Yout, HNSW has the most points with small minHops over three algorithms, which results in a better performance as shown in Fig. 5g.

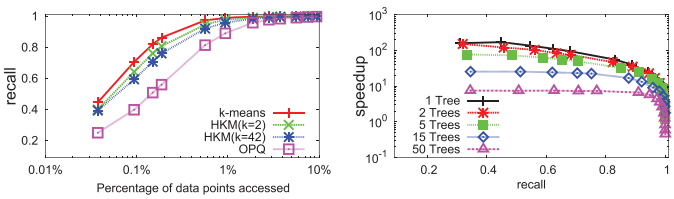
7 Conclusion and Future Work

NNS is an fundamental problem with both significant theoretical values and empowering a diverse range of applications. It is widely believed that there is no practically competitive algorithm to answer exact NN queries in sublinear time with linear sized index. A natural question is whether we can have an algorithm that *empirically* returns *most of* the k-NN points in a *robust* fashion by building an index of size $O(n)$ and by accessing at most αn data points, where α is a small constant (such as 1 percent).

In this paper, we evaluate many state-of-the-art algorithms proposed in different research areas and by practitioners in a comprehensive manner. We analyze their performance and give practical recommendations.

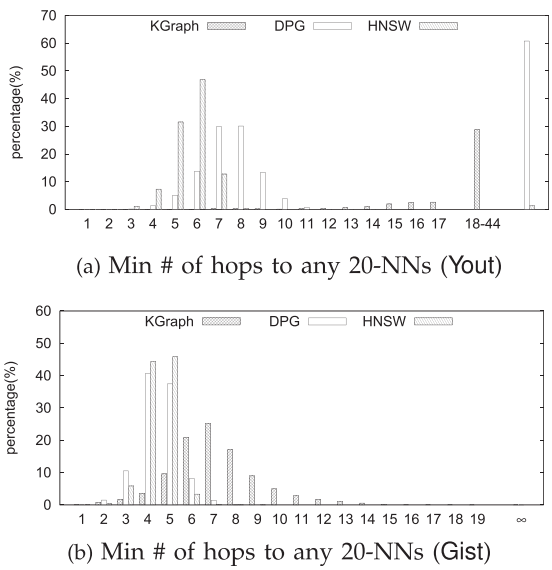
Due to various constraints, the study in this paper is inevitably limited. In our future work, we will (i) consider high dimensional sparse data; (ii) use more complete, including exhaustive method, to tune the algorithms; (iii) consider other distance metrics.

Finally, our understanding of high dimensional real data is still vastly inadequate. This is manifested in many



(a) Partition Quality (b) Multiple HKM Trees

Fig. 13. Analyses of space partitioning-based methods.



(a) Min # of hops to any 20-NNs (Yout)

(b) Min # of hops to any 20-NNs (Gist)

Fig. 14. minHops distributions of KGraph and DPG .

heuristics with no reasonable justification, yet working very well in *real* datasets. We hope that this study opens up more questions that call for innovative solutions by the entire community.

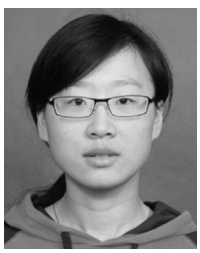
Acknowledgments

Ying Zhang is supported by FT170100128 and DP180103096. Wei Wang is supported by ARC DPs 170103710 and 180103411, and D2DCRC DC25002 and DC25003. Wenjie Zhang is supported by ARC DP180103096 and Huawei YBN2017100007. Xuemin Lin is supported by NSFC 61672235, DP170101628, and DP180103096.

References

[1] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 30th Annu. ACM Symp. Theory Comput.*, 1998, pp. 604–613.
[2] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proc. 24rd Int. Conf. Very Large Data Bases*, 1998, pp. 194–205.
[3] E. Bernhardsson, "Annoy at github," 2005. [Online]. Available: https://github.com/spotify/annoy
[4] M. E. Houle and M. Nett, "Rank-based similarity search: Reducing the dimensional dependence," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 1, pp. 136–150, Jan. 2015.
[5] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.
[6] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 4, pp. 744–755, Apr. 2014.
[7] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin, "SRS: Solving C-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index," *Proc. VLDB Endowment*, vol. 8, no. 1, pp. 1–12, 2014.
[8] W. Dong, "Kgraph," 2014. [Online]. Available: http://www.kgraph.org
[9] W. Liu, J. Wang, S. Kumar, and S. Chang, "Hashing with graphs," in *Proc. 28th Int. Conf. Int. Conf. Mach. Learn.*, 2011, pp. 1–8.
[10] Y. Park, M. J. Cafarella, and B. Mozafari, "Neighbor-sensitive hashing," *Proc. VLDB Endowment*, vol. 9, no. 3, pp. 144–155, 2015.
[11] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi, "DSH: Data sensitive hashing for high-dimensional k-nnsearch," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1127–1138.
[12] R. Caruana, N. Karampatziakis, and A. Yessenalina, "An empirical evaluation of supervised learning in high dimensions," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 96–103.
[13] B. Naidan, L. Boytsov, and E. Nyberg, "Permutation search methods are efficient, yet faster search is possible," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1618–1629, 2015.
[14] E. Bernhardsson, "Benchmarking nearest neighbors," 2016. [Online]. Available: https://github.com/erikbern/ann-benchmarks, 2016.
[15] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. 20th Annu. Symp. Comput. Geometry*, 2004, pp. 253–262.
[16] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *Proc. 47th Annu. IEEE Symp. Foundations Comput. Sci.*, 2006, pp. 459–468.
[17] A. Andoni, P. Indyk, H. L. Nguyen, and I. Razenshteyn, "Beyond locality-sensitive hashing," in *Proc. 25th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2014, pp. 1018–1028.
[18] K. Terasawa and Y. Tanaka, "Spherical LSH for approximate nearest neighbor search on unit hypersphere," in *Proc. Workshop Algorithms Data Structures*, 2007, pp. 27–38.
[19] A. Andoni and I. Razenshteyn, "Optimal data-dependent hashing for approximate near neighbors," in *Proc. 47th Annu. ACM Symp. Theory Comput.*, 2015, pp. 793–801.
[20] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. 25th Int. Conf. Very Large Data Bases*, 1999, pp. 518–529.
[21] R. Panigrahy, "Entropy based nearest neighbor search in high dimensions," in *Proc. 17th Annu. ACM-SIAM Symp. Discrete Algorithm*, 2006, pp. 1186–1195.
[22] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Quality and efficiency in high dimensional nearest neighbor search," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 563–576.
[23] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 950–961.
[24] A. Joly and O. Buisson, "A posteriori multi-probe locality sensitive hashing," in *Proc. 16th Int. Conf. Multimedia*, 2008, pp. 209–218.
[25] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Intelligent probing for locality sensitive hashing: Multi-probe LSH and beyond," *Proc. VLDB Endowment*, vol. 10, no. 12, pp. 2021–2024, 2017.
[26] J. Gan, J. Feng, Q. Fang, and W. Ng, "Locality-sensitive hashing scheme based on dynamic collision counting," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 541–552.
[27] Y. Zheng, Q. Guo, A. K. H. Tung, and S. Wu, "Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 2023–2037.
[28] H. Wang, J. Cao, L. Shu, and D. Rafiei, "Locality sensitive hashing revisited: Filling the gap between theory and algorithm analysis," in *Proc. 22nd ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 1969–1978.
[29] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. 21st Int. Conf. Neural Inf. Process. Syst.*, 2008, pp. 1753–1760.
[30] W. Liu, C. Mu, S. Kumar, and S. Chang, "Discrete graph hashing," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3419–3427.
[31] J. He, W. Liu, and S. Chang, "Scalable similarity search with optimized kernel hashing," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 1129–1138.
[32] Y. Lin, R. Jin, D. Cai, S. Yan, and X. Li, "Compressed hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 446–451.
[33] J. Wang, J. Wang, N. Yu, and S. Li, "Order preserving hashing for approximate nearest neighbor search," in *Proc. 21st ACM Int. Conf. Multimedia*, 2013, pp. 133–142.
[34] J. Wang, W. Liu, A. X. Sun, and Y. Jiang, "Learning hash codes with listwise supervision," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 3032–3039.
[35] Z. Jin, Y. Hu, Y. Lin, D. Zhang, S. Lin, D. Cai, and X. Li, "Complementary projection hashing," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 257–264.
[36] A. Joly and O. Buisson, "Random maximum margin hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2011, pp. 873–880.
[37] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 12, pp. 2916–2929, Dec. 2013.
[38] W. Kong and W. Li, "Isotropic hashing," in *Proc. 26th Annu. Conf. Neural Inform. Process. Syst.*, 2012, pp. 1655–1663.
[39] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," *CoRR*, vol. abs/1408.2927, 2014. [Online]. Available: http://arxiv.org/abs/1408.2927
[40] J. Wang, W. Liu, S. Kumar, and S. Chang, "Learning to hash for indexing big data–A survey," *Proc. IEEE*, vol. 104, no. 1, pp. 34–57, Jan. 2016.
[41] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *CoRR*, vol. abs/1606.00185, 2016. [Online]. Available: http://arxiv.org/abs/1606.00185
[42] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2325–2383, Oct. 1998.
[43] M. Norouzi and D. J. Fleet, "Cartesian K-means," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 3017–3024.
[44] A. Babenko and V. S. Lempitsky, "Tree quantization for large-scale similarity search and classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 4240–4248.
[45] Y. Xia, K. He, F. Wen, and J. Sun, "Joint inverted indexing," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 3416–3423.
[46] Y. Kalantidis and Y. S. Avrithis, "Locally optimized product quantization for approximate nearest neighbor search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 2329–2336.
[47] A. Babenko and V. S. Lempitsky, "The inverted multi-index," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3069–3076.
[48] H. Jegou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: Re-rank with source coding," in *Proc. Int. Conf. Acoustics Speech Signal Process.*, 2011, pp. 861–864.

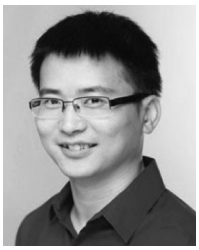
[49] J. Heo, Z. Lin, X. Shen, J. Brandt, and S. Yoon, "Shortlist selection with residual-aware distance estimator for K-nearest neighbor search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 2016, pp. 2009–2017.
[50] A. Babenko and V. S. Lempitsky, "Additive quantization for extreme vector compression," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 931–938.
[51] T. Zhang, C. Du, and J. Wang, "Composite quantization for approximate nearest neighbor search," in *Proc. 31st Int. Conf. Int. Conf. Mach. Learn.*, 2014, pp. 838–846.
[52] D. Cai, X. Gu, and C. Wang, "A revisit on deep hashings for large-scale content based image retrieval," *CoRR*, vol. abs/1711.06016, 2017. [Online]. Available: http://arxiv.org/abs/1711.06016
[53] R. Salakhutdinov and G. Hinton, "Semantic hashing," *Int. J. Approximate Reasoning*, vol. 50, no. 7, pp. 969–978, 2009.
[54] K. Lin, J. Lu, C. Chen, and J. Zhou, "Learning compact binary descriptors with unsupervised deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 1183–1192.
[55] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, "Deep hashing for compact binary codes learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 2475–2483.
[56] G. Wu, L. Liu, Y. Guo, G. Ding, J. Han, J. Shen, and L. Shao, "Unsupervised deep video hashing with balanced rotation," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 3076–3082.
[57] J. Lu, V. E. Liong, and J. Zhou, "Deep hashing for scalable image search," *IEEE Trans. Image Process.*, vol. 26, no. 5, pp. 2352–2367, May 2017.
[58] M. A. Carreira-Perpinan and R. Raziperchikolaei, "Hashing with binary autoencoders," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 557–566.
[59] Z. Xia, X. Feng, J. Peng, and A. Hadid, "Unsupervised deep hashing for large-scale visual search," in *Proc. Int. Conf. Image Process. Theory Tools Appl.*, 2017, pp. 1–5.
[60] T. Do, A. Doan, and N. Cheung, "Learning to hash with binary deep neural network," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 219–234.
[61] T. Do, A. Doan, and N. Cheung, "Discrete hashing with deep neural network," *CoRR*, vol. abs/1508.07148, 2015. [Online]. Available: http://arxiv.org/abs/1508.07148
[62] F. Shen, Y. Xu, L. Liu, Y. Yang, Z. Huang, and H. T. Shen, "Unsupervised deep hashing with similarity-adaptive and discrete optimization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 3034–3044, Dec. 2018.
[63] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proc. 4th Annu. ACM-SIAM Symp. Discrete Algorithms*, 1993, pp. 311–321.
[64] L. Cayton, "Fast nearest neighbor retrieval for bregman divergences," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 112–119.
[65] S. Dasgupta and Y. Freund, "Random projection trees and low dimensional manifolds," in *Proc. 40th Annu. ACM Symp. Theory Comput.*, 2008, pp. 537–546.
[66] C. Silpa-Anan and R. I. Hartley, "Optimised kd-trees for fast image descriptor matching," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2008, pp. 1–8.
[67] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2227–2240, Nov. 2014.
[68] K. Fukunaga and P. M. Narendra, "A branch and bound algorithms for computing k-nearest neighbors," *IEEE Trans. Comput.*, vol. 24, no. 7, pp. 750–753, Jul. 1975.
[69] G. Navarro, "Searching in metric spaces by spatial approximation," *VLDB J.*, vol. 11, no. 1, pp. 28–46, 2002.
[70] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 97–104.
[71] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph," in *Proc. 22nd Int. Joint Conf. Artif. Intell.*, 2011, pp. 1312–1317.
[72] R. Gan, "Scalable k-NN graph construction for visual descriptors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 1106–1113.
[73] W. L. Zhao, J. Yang, and C. H. Deng, "Scalable nearest neighbor search based on KNN graph," *CoRR*, vol. abs/1701.08475, 2017. [Online]. Available: http://arxiv.org/abs/1701.08475
[74] W. Dong, M. Charikar, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *Proc. 20th Int. Conf. World Wide Web*, 2011, pp. 577–586.
[75] Y. M. Zhang, K. Huang, G. Geng, and C. L. Liu, "Fast KNN graph construction with locality sensitive hashing," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2013, pp. 660–674.
[76] J. Yang, W. L. Zhao, C. H. Deng, H. Wang, and S. Moon, "Fast nearest neighbor search based on approximate k-NN graph," in *Proc. Int. Conf. Internet Multimedia Comput. Service*, 2018, pp. 327–338.
[77] J. L. Bentley, "Multidimensional divide-and-conquer," *Commun. ACM*, vol. 23, no. 4, pp. 214–229, 1980.
[78] W. Dong, *High-Dimensional Similarity Search for Large Datasets*. Princeton, NJ, USA: Princeton Univ., 2011.
[79] Z. Jin, D. Zhang, Y. Hu, S. Lin, D. Cai, and X. He, "Fast and accurate hashing via iterative nearest neighbors expansion," *IEEE Trans. Cybern.*, vol. 44, no. 11, pp. 2167–2177, Nov. 2014.
[80] C. Fu and D. Cai, "EFANNA: An extremely fast approximate nearest neighbor search algorithm based on KNN graph," *CoRR*, vol. abs/1609.07228, 2016. [Online]. Available: http://arxiv.org/abs/1609.07228
[81] J. M. Kleinberg, "Navigation in a small world," *Nature*, vol. 406, no. 6798, 2000, Art. no. 845.
[82] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Inf. Syst.*, vol. 45, pp. 61–68, 2014.
[83] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *CoRR*, vol. abs/1603.09320, 2016. [Online]. Available: http://arxiv.org/abs/1603.09320
[84] M. Radovanovic, A. Nanopoulos, and M. Ivanovic, "Hubs in space: Popular nearest neighbors in high-dimensional data," *J. Mach. Learn. Res.*, vol. 11, pp. 2487–2531, 2010.
[85] C.-C. Kuo, F. Glover, and K. S. Dhir, "Analyzing and modeling the maximum diversity problem by zero-one programming*," *Decision Sci.*, vol. 24, no. 6, pp. 1171–1185, 1993.
[86] Y. Sun, W. Wang, Y. Zhang, and W. Li, "Nearest neighbor search benchmark," 2016. [Online]. Available: https://github.com/DBWangGroupUNSW/nns_benchmark
[87] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng, "Query-aware locality-sensitive hashing for approximate nearest neighbor search," *Proc. VLDB Endowment*, vol. 9, no. 1, pp. 1–12, 2015.
[88] Q. Jiang and W. Li, "Scalable graph hashing with feature transformation," in *Proc. 24th Int. Conf. Artif. Intell.*, 2015, pp. 2248–2254.
[89] J. Gao, H. V. Jagadish, B. C. Ooi, and S. Wang, "Selective hashing: Closing the gap between radius search and k-NN search," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 349–358.
[90] L. Boytsov and B. Naidan, "Learning to prune in metric and non-metric spaces," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 1574–1582.
[91] J. He, S. Kumar, and S. Chang, "On the difficulty of nearest neighbor search," in *Proc. 29th Int. Conf. on Machine Learning, {ICML} 2012, *, Edinburgh, Scotland, UK, June 26 - July 1, 2012, http://icml.cc/2012/papers/580.pdf.
[92] L. Amsaleg, O. Chelly, T. Furon, S. Girard, M. E. Houle, K. Kawarabayashi, and M. Nett, "Estimating local intrinsic dimensionality," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 29–38.



Wen Li received the BSc degree in computer science from HeBei University, and the PhD degree from the China University of Mining and Technology. She is a lecturer of Nanjing Audit University and also a visitor in the University of Technology, Sydney. Her research interests include similarity search and trajectory data mining.

Ying Zhang received the BSc and MSc degrees in computer science from Peking University, and the PhD degree in computer science from the University of New South Wales. He is a associate professor and ARC Future fellow (2017-2021) at CAI, the University of Technology, Sydney (UTS). His research interests include query processing on data stream, uncertain data, and graphs. He was an Australian Research Council Australian postdoctoral fellowship (ARC APD) holder (2010-2013).

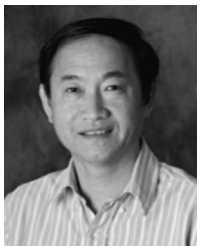
Yifang Sun received the BSc degree in computational mathematics from Nanjing Normal University, and the MSc and PhD degrees in computer science from the University of New South Wales. He is a research associate with the School of Computer Science and Engineering, the University of New South Wales, Australia. His research interests include query processing and optimization on different data types.



Wei Wang received the PhD degree in computer science from the Hong Kong University of Science and Technology, in 2004. He is currently a professor with the School of Computer Science and Engineering, University of New South Wales, Australia. His research interests include integration of database and information retrieval techniques, similarity search, and query processing, and optimization.

Mingjie Li received the bachelor's degree in computer science from Guangxi Normal University, in 2013, and the master's degrees in computer science from the Northeast Normal University, P.R. China, in 2016. He is currently working toward the PhD degree in the Centre for Artificial Intelligence, University of Technology, Sydney. His major research interests include the hashing and embedding methods for similarity search in high dimensional space.

Wenjie Zhang received the PhD degree in computer science and engineering from the University of New South Wales, in 2010. She is currently an associate professor with School of Computer Science and Engineering, the University of New South Wales, Australia. Since 2008, she has published more than 20 papers in SIGMOD, VLDB, ICDE, *TODS, TKDE*, and *VLDBJ*. She is the recipient of the Best (Student) Paper Award of the National DataBase Conference of China 2006, APWebWAIM 2009, Australasian Database Conference 2010, and DASFAA 2012, and also co-authored one of the best papers in ICDE2010, ICDE 2012, and DASFAA 2012. In 2011, she received the Australian Research Council Discovery Early Career Researcher Award.



Xuemin Lin received the BSc degree in applied math from Fudan University, in 1984, and the PhD degree in computer science from the University of Queensland, in 1992. He is a scientia professor with the School of Computer Science and Engineering, the University of New South Wales (UNSW). He has been the head of the Database Research Group at UNSW since 2002. He is a concurrent professor with the School of Software, East China Normal University. Before joining UNSW, he held various academic positions at the University of Queensland and the University of Western Australia. During 1984-1988, he studied for PhD in Applied Math at Fudan University. He currently is an associate editor of the *ACM Transactions on Database Systems*. He is a fellow of the IEEE. His current research interests include the data streams, approximate query processing, spatial data analysis, and graph visualization.