A Unified Approximate Nearest Neighbor Search Scheme by Combining Data Structure and Hashing

Debing Zhang Genmao Yang Yao Hu Zhongming Jin Deng Cai Xiaofei He
State Key Lab of CAD&CG, College of Computer Science,
Zhejiang University, Hangzhou 310058, China.
{debingzhangchina, oliver.ygm, huyao001, zhongmingjin888, dengcai, xiaofeihe}@gmail.com

Abstract

Nowadays, Nearest Neighbor Search becomes more and more important when facing the challenge of big data. Traditionally, to solve this problem, researchers mainly focus on building effective data structures such as hierarchical k-means tree or using hashing methods to accelerate the query process. In this paper, we propose a novel unified approximate nearest neighbor search scheme to combine the advantages of both the effective data structure and the fast Hamming distance computation in hashing methods. In this way, the searching procedure can be further accelerated. Computational complexity analysis and extensive experiments have demonstrated the effectiveness of our proposed scheme.

1 Introduction

The increasing interest in social network and multimedia has led an explosive growth of data. The problem of k Nearest Neighbor (kNN) search becomes one of the most fundamental issues when facing the challenge of *Big Data* for many applications such as large scale image and video retrieval [Gong and Lazebnik, 2011; Wang *et al.*, 2012; Song *et al.*, 2011], object classification [Sengupta, 2012] and large scale image clustering [Shindler *et al.*, 2011]. Formally, given a data set $X = [x_1, x_2, \ldots, x_N] \subset \mathbb{R}^d$ containing N points and a query point $q \in \mathbb{R}^d$, k Nearest Neighbor search aims to find the k nearest points of q under some distance measure such as Euclidean distance.

The exact nearest neighbors can be found by linear search method with a computational cost of $O(Nd)$, which is prohibitively expensive for large scale data set. Some Approximate Nearest Neighbor (ANN) methods have been explored by sacrificing some precision to gain more efficiency. Currently, most ANN methods can be classified into two categories: data structure based methods and hashing based methods. Data structure based methods usually build space-partitioning index structures (such as hierarchical k-means tree [Nister and Stewenius, 2006], kd-trees [Friedman *et al.*, 1977], R-tree [Cheung and Fu, 1998] and kNN graph [Hajebi *et al.*, 2011]) for similarity search, while hashing based methods project the data into a low dimensional Hamming space and the efficient Hamming distance is utilized to approximate the similarity.

Data structure based methods can reduce the number of data points that need to be searched by pruning the search space, thus improves the query efficiency. But when computing the similarity between the query and the candidate data points, traditional data structure based methods simply calculate the exact Euclidean distance which is still very time consuming. Euclidean distance computations can become the bottleneck when dealing with large scale and high dimensional data.

Hashing based methods try to solve the ANN problem in high dimensional data space by encoding the data into binary codes. Given a binary code of query point q, hashing based methods firstly find all the points falling into a ball centered at the query q with Hamming radius r, then a linear scan search is performed over these points to return the required nearest neighbors. Supposing the bit length is n, $O(\sum_{i=0}^{r} C_n^i)$ operations are needed to find all the candidates in the Hamming ball [Norouzi *et al.*, 2012]. Thus, it is impossible for hashing based methods with long binary codes to be directly used in real applications. Based on this fact, most of current hashing methods aim to derive compact hash codes to avoid this difficulty. However, the accuracy is hard to be further improved since the limited representative ability of compact codes.

In this paper, we are primarily interested in how to take full use of the hamming distance computation to accelerate the query process. We emphasize that the hamming distance computation is extremely fast on modern CPUs. As is shown in Table 1, 50$\sim$100 speedup can be achieved if we calculate the similarity by hamming distance instead of Euclidean distance. We also notice that, when calculating hamming distance with different bits, the computational time varies little when the bit length is smaller or equal to 128. And long binary codes have better represent the original data. Motivated by these facts, we propose a novel unified approximate nearest neighbor search scheme by integrating the fast hamming distance computation into some data structures such as hierarchical k-means tree. In this way, we can solve the time consuming Euclidean distance computation problem in data structure based methods, and data structure based pruning can also be used to tackle the difficulty of exhaustive search in hamming space. Further more, long binary codes are used in our scheme to better approximate the original

Dim	Euclidean Distance	Hamming Distance	Speedup
64	1.29	0.02	64.5
128	2.43	0.02	121.5
256	4.9	0.07	70
512	9.62	0.22	43.7

Table 1: The speedup of Hamming distance compared with Euclidean distance under different dimensions. This table shows the total time cost (s) of 25,000,000 tests.

data while the limitation of hashing methods with long binary codes is avoided. Extensive experimental results on SIFT and GIST data sets demonstrate the effectiveness of our proposed scheme in comparison with the state-of-the-art ANN search approaches.

The rest of the paper is organized as follows. In section 2, we briefly review the related work of Approximate Nearest Neighbor search. And in Section 3, we take hierarchical k-means as an example to show how data structures and hash methods can be unified to improve ANN search, and give some computational analysis. In Section 4, we demonstrate the effectiveness and efficiency of our proposed unified ANN search scheme.

2 Related Work

In the following two subsections, we briefly introduce two popular Approximate Nearest Neighbor search approaches.

2.1 Hashing

Locality Sensitive Hashing (LSH) is a widely used hashing technique [Gionis *et al.*, 1999; Andoni and Indyk, 2008] based on random linear projection. Considering the possible low dimensional embedding of the original data, Kulis *et al.* propose to generalize the traditional LSH to nonlinear case by incorporating arbitrary kernel functions into hash function [Kulis and Grauman, 2009]. Other hashing methods include SPH[Heo *et al.*, 2012], AGH [Me *et al.*, 2011], ITQ [Gong and Lazebnik, 2011], SIKH [Raginsky and Lazebnik, 2009] and so on.

2.2 Data Structure based Methods

Traditional kd-tree methods retrieve the nearest neighbors for a given query by simple back tracking. To accelerate the search, Best Bin First heuristic is proposed by maintaining a priority queue of nodes that have been revisited [Lowe, 2004]. Forests of randomized kd-trees [Silpa-Anan and Hartley, 2008] is proposed, and multiple kd-trees are generated and queried instead of a single tree. Further more, by incorporating tree structure and k-means clustering method, Nister *et al* describe hierarchical k-means (HKM) clustering as an ANN method [Nister and Stewenius, 2006].

K-means neighbor graph based approaches are another popular topic for approximate nearest neighbor search [Hajebi *et al.*, 2011; Paredes and Chávez, 2005]. These algorithms build a nearest neighbor graph in an offline phase and the retrieval is conducted by performing hill climbing on the obtained graph. In this way, points far away from the query can be eliminated to avoid unnecessary exhaustive search on the remaining data set.

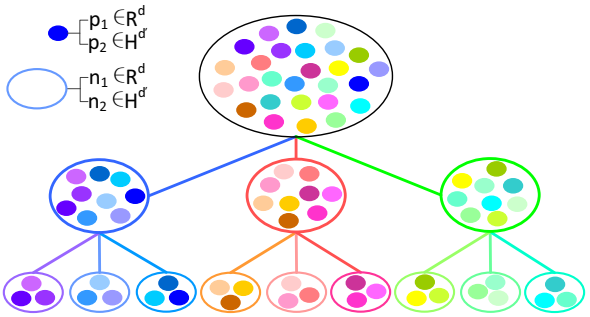

Figure 1: An example of hierarchical k-means tree with struct nodes. A solid dot represents a data point, while a hollow ellipse represents a data cluster which is a node in HKM tree. Each point has two representations: p_1 and p_2, p_1 stands for the representation in original data space $\mathbb{R}^d$ and p_2 stands for its representation in Hamming space $\mathbb{H}^{d'}$. Similarly, each node also has two representations: $n_1 \in \mathbb{R}^d$ and $n_2 \in \mathbb{H}^{d'}$.

3 A Unified ANN Search Scheme

Our idea is to build a more efficient ANN search scheme by combining data structure and hashing. On one side, we use data structure to take full advantage of hashing methods with long bit length which can get high accuracy. Well designed data structures can greatly prune the search space, thus exhaustive linear search of hash codes can be avoided. On the other side, we accelerate traditional data structure based methods by greatly reducing the proportion of high dimensional floating-point vector distance computation.

In the following subsections, we take hierarchical k-means tree as an example to show how our proposed scheme works.

3.1 Algorithm

Building the Data Structure

Given a data set which contains N data points in $\mathbb{R}^d$, we first build a hierarchical k-means tree with L levels to index all the N data points. Generally, we assume in the l-th level, each node is split into c_{l+1} ($l = 0, 1, 2, ..., L-1$) small clusters which are the nodes in the $(l+1)$-th level. Figure 1 shows an example with $N = 27, L = 2, c_1 = 3, c_2 = 3$. For each node which is also a cluster, we calculate the center of all the points that belong to this cluster as its first representation in the original space $\mathbb{R}^d$. Then, we can adopt an arbitrary hashing method such as LSH and KLSH to map each node's first representation to a d' dimensional Hamming space, where each node has the new d' dimensional binary code representation as its second representation. So the main difference of the new data structure from traditional HKM is that each node is a struct consisting of two representations: the first one is accurate and the second is approximate. This new data structure provides more flexibility, and Hamming distance can be used as an optional distance metric when calculating the similarity.

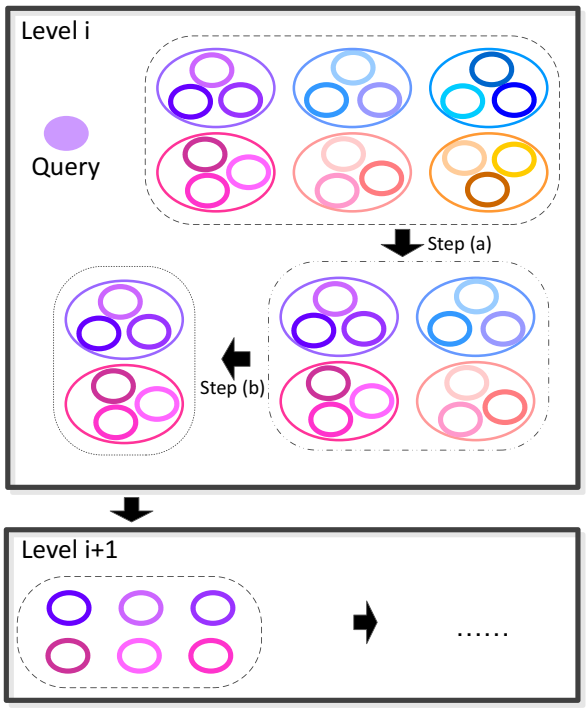

Figure 2: Example of the query procedure. The solid dot on the up-left corner is a query point. And suppose we are in the i-th level of the HKM tree, we have six nodes which are expanded by the nodes in the $(i-1)$-th level to search, and we need to keep $s_i = 2$ nearest nodes of the query in this level. We first use a fast coarse ranking algorithm based on comparing Hamming distance which is shown in step (a) to select $r_i = 4$ $(r_i \geq s_i)$ candidate nodes. Then an accurate ranking on the selected $r_i = 4$ nodes is adopted in step (b) based on the exact distance measure of the original space to select the final $s_i = 2$ nodes. The selected two nodes will be expanded in the $(i+1)$-th level. In this example we can save $6 - 4 = 2$ Euclidean distance computations by the two step ranking and the high accuracy is also kept at the same time.

Querying

When given a query which has only the first representation, we adopt the same hashing method used in building the data structure to compute its second representation. Then we search it in the HKM tree from the top level to the bottom level. In each level, only a small part of nodes that are nearest to the query will be kept to be further searched in the next level. Suppose we keep s_l nodes in the l-th level. Clearly, if we keep 10% nodes in each level, the number of data points that need to be finally searched will be exponentially reduced to $(0.1)^L N$, and the query time can be greatly saved. Traditionally, to decide which s_l nodes in the l-th level are the nearest ones to the query point, people have to exhaustively compute the Euclidean distance from the query to the candidate nodes in the l-th level, since each node has only one representation in original data space. The number of candidate nodes in the l-th level is $s_{l-1}c_l$, which are expanded from the s_{l-1} nodes of the above level. While in our new HKM data structure, we can easily accelerate this process by a two step ranking. Firstly, a coarse ranking in Hamming space is adopted to get r_l $(r_l \geq s_l)$ nearest nodes by using each node's second representation. Then in order to compensate the accuracy loss, we rerank the r_l nodes by using their first representations which are accurate. In this way, we can get high accuracy and achieve low query time simultaneously. Finally, we can get s_L nearest nodes and the data points contained in them. Then a similar coarse ranking which finds the top r_p $(r_p \geq k)$ data points will be done first, and the last step, a fine reranking to the r_p points is used to give the final k nearest neighbors of the query. We illustrate the query process in Figure 2.

3.2 Complexity Analysis

In this subsection, We give some complexity analysis of both the offline HKM building and the online querying.

Building HKM Complexity

As is known, if we cluster n data points of $\mathbb{R}^d$ into c clusters by k-means clustering, the complexity is $O(ndcm)$, where m is the number of iterations needed for k-means to converge. Since m could be set to be a constant number such as 20 in real applications, we will omit it below. For the complexity of HKM, it's clear that the building of HKM contains many small k-means clusterings on different sizes of data set. In the $(l-1)$-th level of HKM, there are $C_{l-1} = \prod_{i=1}^{l-1} c_i$ nodes, each containing approximately $N_{l-1} = \frac{N}{C_{l-1}}$ data points. To build the l-th level, we need to execute C_{l-1} times k-means clusterings, each of which clusters N_{l-1} data points into c_l clusters, and the complexity is $O(C_{l-1}N_{l-1}c_l d) = O(Nc_l d)$. So by adding each level's building complexity together, we get the complexity of building the whole HKM, which is $O(N(\sum_{i=1}^{L} c_i)d)$. In a common HKM structure where $c_i = c, i = 1, 2, ..., L$, the complexity is $O(NdLc)$.

Similarly, we give the complexity of calculating the first representation of each node as $O(NdL)$, and the complexity of calculating the second representation of each node as $O(Ndd')$.

Thus, the total complexity of building the proposed HKM data structure is $O(Nd(Lc + L + d'))$. This complexity is similar with that of traditional HKM tree.

Query Complexity

Based on the query algorithm, in the l-th level, we analyze the complexity of the coarse ranking and the fine reranking separately. For the coarse ranking, we first need to select the nearest r_l nodes of the query from $s_{l-1}c_l$ nodes expanded by the s_{l-1} nodes in the $(l-1)$-th level by Hamming distance measure. We calculate the $s_{l-1}c_l$ Hamming distances, and the complexity is $O(s_{l-1}c_l)$ since Hamming distance calculation is near $O(1)$. Then we partially sort all the $s_{l-1}c_l$ distances to get the top r_l nodes, and the complexity is $O(s_{l-1}c_l \log r_l)$. For the fine reranking, similarly, we have a complexity of $O(r_l d)$ in Euclidean distances computation and a complexity of $O(r_l \log s_l)$ in partially sorting. Clearly, the complexity in the l-th level is $O(s_{l-1}c_l + s_{l-1}c_l \log r_l + r_l d + r_l \log s_l)$. For the final step, each of the s_L nodes has approximately $\frac{N}{C_L}$ data points, thus the complexity of the final step is

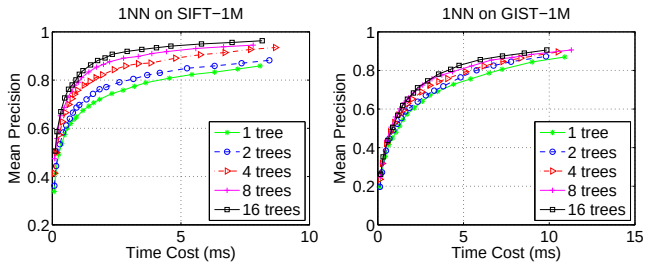

Figure 3: kd-trees with 1, 2, 4, 8 and 16 trees are compared, the mean precision and time cost per query are obtained by varying the number of check nodes.

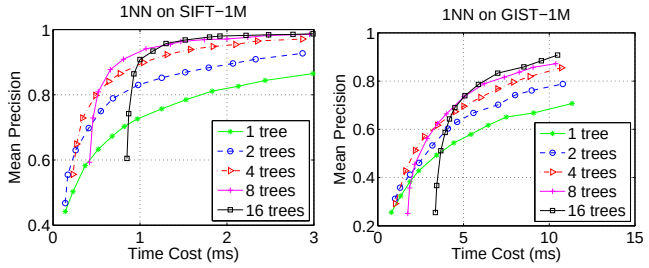

Figure 4: HKM with 1, 2, 4, 8 and 16 trees are compared, the mean precision and time cost per query are obtained by varying the number of check nodes.

$O(\frac{N}{C_L}+\frac{N}{C_L}\log r_p+r_p d+r_p \log k)$. So, totally the complexity of the proposed scheme is $O(\sum_{l=1}^{L}(s_{l-1}c_l + s_{l-1}c_l \log r_l + r_l d + r_l \log s_l) + \frac{N}{C_L} + \frac{N}{C_L}\log r_p + r_p d + r_p \log k)$.

Note that for high dimensional data set where d is very large, the main part of query complexity is $O(r_p d + \sum_{l=1}^{L} r_l d)$, corresponding to the complexity of Euclidean distance computation. And from the experimental results in the following section, we can see that $r_p + \sum_{l=1}^{L} r_l$ in our proposed scheme is much smaller than that in traditional data structured based ANN search methods such as HKM trees and kd-trees for a given accuracy.

4 Experiments

kd-trees and HKM trees

In this section, we compare our unified ANN search scheme with several state-of-the-art ANN search techniques including both data structure based methods and hashing based methods on several data sets.

4.1 Data sets

The experiments are carried out on real-world publicly available image data sets

- SIFT-1M: It contains one million SIFT descriptors and each descriptor is represented by a 128-dim vector. This data set has been used in [Wang *et al.*, 2012; Jun Wang, 2010; Joly and Buisson, 2011].

Data Set	Bit Length	Method	MP	Time Cost
SIFT-1M	256	LSH	0.864	4.83+32.2
		KLSH	0.870	4.86+30.6
	512	LSH	0.989	13.0+32.3
		KLSH	0.974	13.0+31.7
GIST-1M	256	LSH	0.604	4.69+32.0
		KLSH	0.836	4.77+31.8
	512	LSH	0.745	13.2+33.4
		KLSH	0.879	12.9+32.0

Table 2: Mean precision and time cost of LSH and KLSH on 1NN search. An exhaustively linear scan is adopted to obtain the results. The time cost consists of computing and sorting the Hamming distances, which are the bottlenecks of traditional hashing methods especially when dealing with large scale data set.

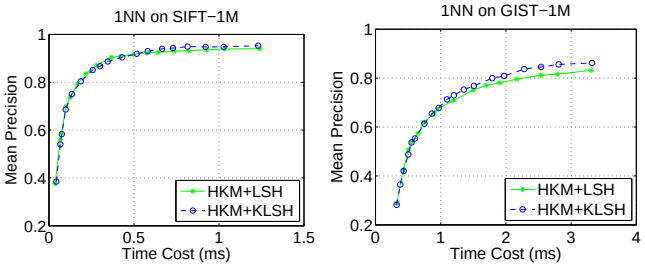

Figure 5: Our scheme's performance with LSH and KLSH. On SIFT data set, the code length is set to 512. On GIST data set, the code length is set to 256.

- GIST-1M: It contains one million GIST descriptors and each descriptor is represented by a 960-dim vector. This data set is publicly available[1] and has been used in [Wang *et al.*, 2012; Heo *et al.*, 2012; Liu *et al.*, 2012].

4.2 Compared Methods

- Data structure based methods include
  - kd-trees: Randomized kd-trees with best-bin-first search heuristic has been compared. We also notice that a PCA rotation on the original data will improve the precision of randomized kd-trees method significantly. So all experimental results of kd-trees are carried out on the rotated data.
  - Hierarchical K-Means trees (HKM): HKM builds the data structure by clustering the data using hierarchical k-means algorithm which can significantly reduce the search space when given a query.

  Mujas' FLANN (Fast Library for Approximate Nearest Neighbors) library [Muja and Lowe, 2009] has been used for both the kd-trees and HKM implementations. The FLANN library is a mature, reasonably optimized free software package of ANN search algorithms.
- Hashing based methods include

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

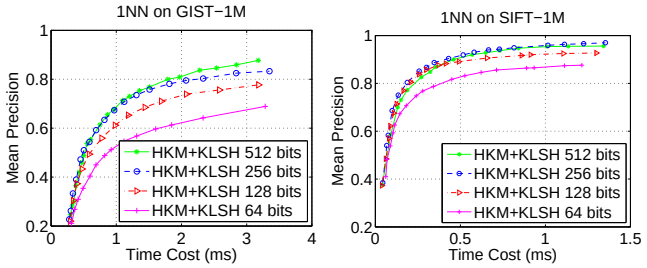

Figure 6: Results of HKM+KLSH with different binary code lengths (64, 128, 256 and 512).

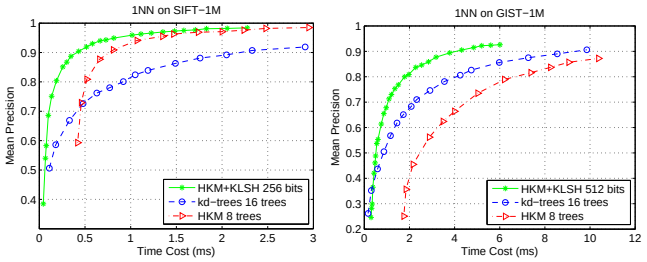

Figure 7: 1NN performances on SIFT and GIST.

  - Locality Sensitive Hashing (LSH): LSH projects the original data to Hamming space by random projections. Then an exhaustively search is used in the Hamming space.
  - Kernelized Locality Sensitive Hashing (KLSH): KLSH [Kulis and Grauman, 2009] formulates the random projection used by LSH in arbitrary kernel space. KLSH with a Gaussian RBF kernel has been used in our experiment. KLSH may outperform LSH especially when the data dimensionality is high, as is shown in Table 2.

- For the proposed unified ANN search scheme, we show two examples including
  - HKM+LSH: A unified search technique by combining HKM and LSH.
  - HKM+KLSH: A unified search technique by combining HKM and KLSH.

In our scheme, we build hierarchical k-means trees for SIFT and GIST data sets similarly. Both trees have two levels where $c_1 = c_2 = 100$, which means the data points are firstly grouped into 100 clusters, then each cluster is divided again into 100 smaller clusters each containing approximately 100 data points. For the query process, to simplify the problem of choosing search parameters, we set $1 \leqslant s_1 = s_2 \leqslant 16$ and $r_1 = 100 = c_1$ while r_2 is adjusted from 100 to 800 and r_p is adjusted from 100 to 2500 corresponding to the different choices of s_1.

4.3 Evaluation

For each data set, we randomly select 1000 data points as the queries and use the remaining 999,000 data points to form

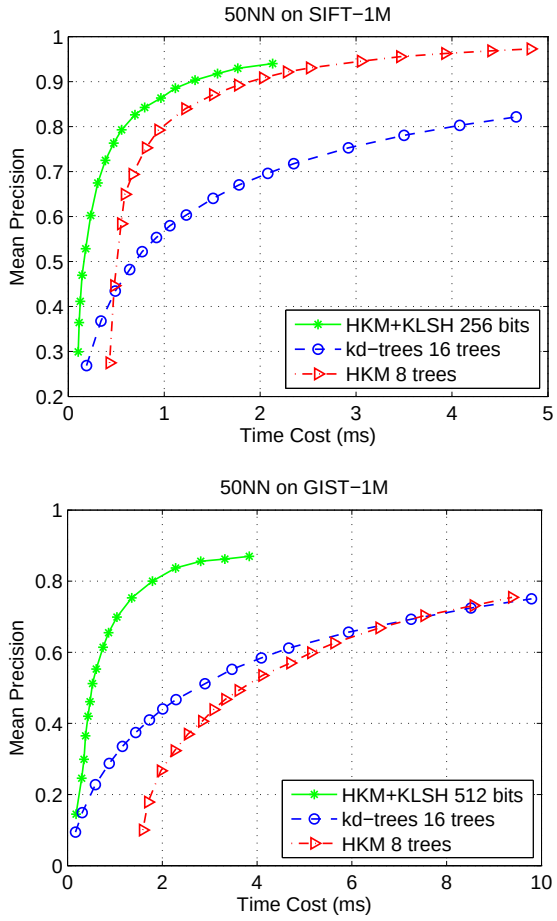

Figure 8: 50NN performances on SIFT and GIST.

the database. Following the similar work in [Heo *et al.*, 2012; Joly and Buisson, 2011], a retrieved point is considered to be a true neighbor if it is among the k nearest neighbors (measured by the Euclidian distance in the original space) of the query. Mean Precision (MP) is used throughout the paper as a main criteria and is defined as follows

$$Mean\ Precision = \sum_{i=1}^{q} \sum_{j=1}^{k} \frac{p_{i,j}}{kq} \qquad (1)$$

Here q is the sum of query, $p_{i,j}$ is set to be 1 if the j-th retrieved point of the i-th query lies in the k nearest neighbors, otherwise $p_{i,j}$ is set to be 0.

The experiments are carried out on a computer with an Intel(R) Core(TM) i7-3770 3.40GHz CPU and 8GB memory. And all methods are tested in a single thread mode.

4.4 Parameter Selection

kd-trees and HKM

Although the building time of both kd-trees and HKM increases with the number of trees, both methods are likely to obtain better precision with more trees. We exhaustively tried kd-trees and HKM trees with different number of trees (1, 2,

Data Set	Mean Precision	Method	Time (ms)
SIFT-1M	80%	HKM+KLSH	0.18
		HKM	0.51
		kd-trees	0.92
	90%	HKM+KLSH	0.42
		HKM	0.78
		kd-trees	2.2
GIST-1M	80%	HKM+KLSH	1.8
		HKM	6.7
		kd-trees	4.2
	90%	HKM+KLSH	4.2
		HKM	12.2
		kd-trees	9.3

Table 3: 1NN results of all ANN methods.

Data Set	Mean Precision	Method	Time (ms)
SIFT-1M	80%	HKM+KLSH	0.59
		HKM	4.0
		kd-trees	1.0
	90%	HKM+KLSH	1.3
		HKM	9.5
		kd-trees	1.9
GIST-1M	70%	HKM+KLSH	1.0
		HKM	7.5
		kd-trees	7.5
	80%	HKM+KLSH	1.8
		HKM	12
		kd-trees	13

Table 4: 50NN results of all ANN methods.

4, 8 and 16), and the results are shown in Figure 3 and Figure 4. We find that kd-trees with 16 trees and HKM with 8 trees are the best.

LSH and KLSH

We then briefly compare the performances of two hashing methods, and the results are shown in Table 2. Obviously, KLSH performs better than LSH under different bit lengths on GIST, while on SIFT data set the performances of both methods are similar. And we note that when the bit length is short, hashing based methods can not achieve good results. But when the bit length is long, traditionally, people have to search exhaustively over all hash codes which is very time consuming. Hashing methods based on linear search are about 10 times slower than methods that based on tree structures, and this gap will be more obvious with the increasing scale of data set. Thus, hashing methods won't be further compared in the following experiments.

Our Scheme

We first test the performances of different combinations: HKM with LSH and HKM with KLSH. Figure 5 shows the comparisons on SIFT and GIST data sets. For SIFT data set, the performances of the two combinations are similar, and for GIST data set, HKM with KLSH achieves higher precision compared with HKM with LSH. This is reasonable since KLSH performs better than LSH on high dimensional data set which is shown in Table 2. So we adopt HKM with KLSH as a better scheme.

Next, we test the performance of HKM with KLSH under different bit lengths. Long bits are accurate but may take more time. In Figure 6, we can see that on SIFT data set whose dimensionality is not very high, 256 bits is good enough for HKM with KLSH, and on GIST data set whose dimensionality is much higher, HKM with KLSH needs more bits, 512 bits is the best choice.

4.5 Results

In this subsection, we show the comparisons of all these ANN search methods with their own best parameters. Figure 7 shows the detailed results of nearest neighbor search (1NN) on SIFT and GIST data sets. HKM with KLSH as an combination of data structure and hashing method shows its great advantage. The time costs to reach 80% and 90% mean precision on both data set are shown in Table 3. On SIFT data set, to reach the 90% precision, for each query, HKM with KLSH only needs 0.42ms, while HKM trees needs 0.78ms and kd-trees needs 2.2ms. On GIST data set, to reach the 90% precision, for each query, HKM with KLSH only needs 4.2ms, while HKM trees needs 12.2ms and kd-trees needs 9.3ms. So the results show the great power of HKM with KLSH when dealing with high dimensional problems.

50NN search experiments are also conducted, and results are shown in Figure 8. Generally, it takes longer time to achieve a high precision in 50NN search problem. But HKM with KLSH still performs the best, and the results are most encouraging when dealing with 50NN search task on GIST data set. To achieve the precision of 80%, HKM with KLSH can be more than 6 times faster compared with kd-trees and HKM trees, as is shown in Table 4.

5 Conclusion and Future Work

In this paper, we propose a novel unified approximate nearest neighbor search scheme by combining data structures with hashing methods. We introduce how the proposed scheme works by taking the hierarchical k-means tree as an example. Each node of traditional HKM has been extended to a struct which consists of two representations: one is the accurate representation in original space and the other is the approximate representation in Hamming space. In this way, traditional search strategy which is purely based on the time consuming Euclidean distance computation now can be approximated and accelerated by Hamming distance computation which only takes $O(1)$ time in modern CPUs' architecture.

For future work, similar changes to HKM can also be adopted in the k nearest neighbor graph. When choosing the direction of hill climbing, the two step ranking procedure may also be helpful. And extending the unified scheme to multi data structures such as multi trees and multi graphs may improve the ANN search performance further.

6 Acknowledgements

This work is supported by National Basic Research Program of China (973 Program) under Grant 2012CB316400 and

National Natural Science Foundation of China (Grant No: 61125203, 61222207, 61233011, 90920303).

References

[Andoni and Indyk, 2008] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[Cheung and Fu, 1998] King Lum Cheung and Ada Wai-Chee Fu. Enhanced nearest neighbour search on the r-tree. *ACM SIGMOD Record*, 27(3):16–21, 1998.

[Friedman *et al.*, 1977] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[Gionis *et al.*, 1999] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, 1999.

[Gong and Lazebnik, 2011] Yunchao Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[Hajebi *et al.*, 2011] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, 2011.

[Heo *et al.*, 2012] Jae-Pil Heo, YoungWoon Lee, Junfeng He, Shih-Fu Chang, and Sung eui Yoon. Spherical hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[Joly and Buisson, 2011] Alexis Joly and Olivier Buisson. Random maximum margin hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[Jun Wang, 2010] Shih-Fu Chang Jun Wang, Sanjiv Kumar. Sequential projection learning for hashing with compact codes. In *Proceedings of the International Conference on Machine Learning*, 2010.

[Kulis and Grauman, 2009] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision*, 2009.

[Liu *et al.*, 2012] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[Lowe, 2004] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[Me *et al.*, 2011] Wei Me, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *Proceedings of the International Conference on Machine Learning*, 2011.

[Muja and Lowe, 2009] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *In VISAPP International Conference on Computer Vision Theory and Applications*, 2009.

[Nister and Stewenius, 2006] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006.

[Norouzi *et al.*, 2012] Mohammad Norouzi, Ali Punjani, and David J. Fleet. Fast search in hamming space with multi-index hashing. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2012.

[Paredes and Chávez, 2005] Rodrigo Paredes and Edgar Chávez. Using the k-nearest neighbor graph for proximity searching in metric spaces. In *Proceedings of the international conference on String Processing and Information Retrieval*, 2005.

[Raginsky and Lazebnik, 2009] Maxim Raginsky and Svetlana Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems*. 2009.

[Sengupta, 2012] Sunando Sengupta. Efficient discriminative learning of parametric nearest neighbor classifiers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[Shindler *et al.*, 2011] Michael Shindler, Alex Wong, and Adam W. Meyerson. Fast and accurate k-means for large datasets. In *Advances in Neural Information Processing Systems 24*. 2011.

[Silpa-Anan and Hartley, 2008] Chanop Silpa-Anan and Richard Hartley. Optimised kd-trees for fast image descriptor matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2008.

[Song *et al.*, 2011] Jingkuan Song, Yi Yang, Zi Huang, Heng Tao Shen, and Richang Hong. Multiple feature hashing for real-time large scale near-duplicate video retrieval. In *Proceedings of the ACM international conference on Multimedia*, 2011.

[Wang *et al.*, 2012] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2393–2406, 2012.