Fast kNN Graph Construction with Locality Sensitive Hashing

Yan-Ming Zhang[1], Kaizhu Huang[2], Guanggang Geng[3], and Cheng-Lin Liu[1]

[1] National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China
{ymzhang,liucl}@nlpr.ia.ac.cn,
[2] Department of EEE, Xi'an Jiaotong-Liverpool University, SuZhou, China,
kaizhu.huang@xjtlu.edu.cn
[3] China Internet Network Information Center, Chinese Academy of Sciences, Beijing, China,
gengguanggang@cnnic.cn

Abstract. The k nearest neighbors (kNN) graph, perhaps the most popular graph in machine learning, plays an essential role for graph-based learning methods. Despite its many elegant properties, the brute force kNN graph construction method has computational complexity of $O(n^2)$, which is prohibitive for large scale data sets. In this paper, based on the divide-and-conquer strategy, we propose an efficient algorithm for approximating kNN graphs, which has the time complexity of $O(l(d + \log n)n)$ only (d is the dimensionality and l is usually a small number). This is much faster than most existing fast methods. Specifically, we engage the locality sensitive hashing technique to divide items into small subsets with equal size, and then build one kNN graph on each subset using the brute force method. To enhance the approximation quality, we repeat this procedure for several times to generate multiple basic approximate graphs, and combine them to yield a high quality graph. Compared with existing methods, the proposed approach has features that are: (1) much more efficient in speed (2) applicable to generic similarity measures; (3) easy to parallelize. Finally, on three benchmark large-scale data sets, our method beats existing fast methods with obvious advantages.

Keywords: graph construction, locality sensitive hashing, graph-based machine learning

1 Introduction

Graph-based learning methods present an important category of machine learning methods, and have been widely used in areas like image processing, computer vision, and data mining. These methods first represent the data set by a similarity graph, and then perform on this graph the traditional learning tasks, such as clustering [25], dimensionality reduction [1], and classification [34]. As observed

by many researchers [28, 33], the graph construction step plays an extremely important role to this kind of methods, and has attracted much attention recently.

Although various graph construction methods have been proposed to better describing the data set [19, 9, 7], the kNN graph still presents the most popular one in practice due to its robust performance. Given a data set, the kNN graph is constructed by connecting each item to k items which are the most similar to it under a given similarity measure. Despite its simplicity in concept, a direct implementation suffers from high computational cost. Since finding the kNN for each item needs $n-1$ comparisons, it takes $O(n^2)$ time to construct the kNN graph. Obviously, it is too slow for large scale problems. In fact, after the recent development of fast graph-based learning methods [17, 32], the speed of the graph construction method is becoming the bottleneck of graph-based methods.

To alleviate this problem, substantial efforts have been made to reduce the complexity of kNN graph construction. Early works [3, 27] focused on constructing exact kNN graph. However, their complexity scales exponentially with data dimensionality. Recently, researchers switched their attention to construct approximate kNN graph and obtained encouraging results. These methods adopted techniques such as space partition tree [6, 30], and local search [12]. A brief introduction to these methods can be seen in the next section.

Following the research direction, in this work, we propose a novel approximate kNN graph construction method which leads to significantly fast speed with high accuracy. Its basic idea is to divide the whole data set into small groups, and finds each item's kNN within the group it belongs to. Since the size of a group is much smaller than the size of the whole data set, the cost for finding the approximate kNN is much lower. To make this method work well, the key is to divide the data set in such a way that: (1) The pairwise similarity between items should be preserved such that similar items remain in the same group; (2) The group size should be kept as small as possible. We propose to group similar items by adopting locality sensitive hashing (LSH) [13], which enjoys a rigorous theoretical performance guarantee even in the worst case [18]. Further, we design a simple method to take control of the group size. As groups have no overlapping, the constructed graph is a union of multiple isolated small graphs. To improve the approximation quality, we repeat the division for several times to generate multiple basic approximate graphs. Finally, we combine them to yield a graph with high accuracy.

Compared with existing methods, we emphasize that the proposed method enjoys the following appealing advantages:

1. Fast and accurate. Our method has the time complexity of $O(l(d+\log n)n)$ (d is the dimensionality and l is usually a small number). This is much faster than most existing fast methods [12, 6] (see the next section for details). Moreover, as shown by experiments, our method can generate good approximate kNN graphs by scanning a small proportion of pairwise distances. On several benchmarks, our method beats existing fast methods with obvious advantages.

2. Applicable to generic similarity measure. Thanks to the development of LSH, hash functions have been designed for different similarity measures, such as l_p [10], Mahalanobis distance [22], kernel similarity [21], and χ^2 distance [14]. Thus, we can conveniently use them to group data items according to the problem at hand; this presents one of the biggest advantages over the other methods such as [6] and [30], which can only be applied in Euclidean space.
3. Easy to parallelize. Since the construction of multiple basic approximate graphs are independent of each other, we can further speedup our method by constructing these basic graphs simultaneously.

The rest of this paper is organized as follows: the next section briefly reviews the existing approximation graph construction methods. Section 3 gives an introduction to the LSH technique, which provides the basic tools for our method. After that, section 4 presents a detailed description and complexity analysis of our method, while Section 5 gives comparison experimental results to validate the advantages of our method. Finally, we set out the conclusion in Section 7.

2 Related Work

The fast construction for kNN graph have been studied for a long time, and a comprehensive introduction to the graph construction methods can be found in [6]. In this section, we will briefly review several existing methods for approximate kNN graph construction, and a closely-related but different problem: the kNN search.

Chen, Fang, and Saad [6] proposed a divide-and-conquer style algorithm for constructing approximate kNN graph. The method recursively divides the data points into subsets with overlapping, then constructs one kNN graph on each small subset. The final graph is constructed by merging all the small graphs together using overlapping parts. Empirically, the authors reported their method had complexity of $O(dn^{1.22})$. Recently, Wang et al. [30] proposed another efficient algorithm using a similar idea. But data sets are recursively divided without overlapping. To increase the kNN recall, it constructs multiple basic graphs by repeating the division procedure for several times. To make good division, both methods use principle direction to partition data set. Thus, they need to compute $O(n)$ principle directions, one for each internal node. Although sub-sampling and Lanczos algorithm are adopted, it is still costly. Also, these methods can only be applied in Euclidean space.

Dong, Moses, and Li [12] proposed a fast kNN graph construction method based on local search. Its motivation lies in that a neighbor of a neighbor is also likely to be a neighbor. Initializing each node with a random set of neighbors, the method iteratively improves each node's neighborhood by exploring its neighbors' neighborhoods. Although the paper reported that its empirical cost was $O(n^{1.14})$, there is no formal guarantee on the algorithm complexity.

kNN search is a close-related but different problem which is extensively used in areas like information retrieval, pattern recognition. Given a query q, kNN search aims to find out the k most similar objects in the database. Thus, the

construction of a kNN graph can be viewed as a kNN search problem where each data point itself is a query. Although many excellent works, such as space partition tree [2, 26] and locality sensitive hashing (LSH) [13, 10], have been done for performing efficient kNN search, the direct use of kNN search approach for the graph construction results in unfavorable results [12, 30]. The differences between the two problems lie in two aspects:

1. Since the prime concern of kNN search is to reduce the query time, these methods typically build elaborate indexing structure in the training step. However, there is no separated training phase for the graph construction problem, and we can not afford the high cost for building complex indexing structure.
2. kNN search is an inductive problem, which means we can not acquire query points in the training phase. However, kNN graph construction is a transductive problem in which all query points are at hand. Thus, kNN graph construction is easier in general, and we could take advantages of its characteristic to design more efficient algorithm.

Recently, Goyal, Daumé and Guerra [15] proposed an approximate graph construction method for natural language processing problem by adopting LSH technique. Basically, it applies the method of [5] which returns k approximate neighbors for each query in constant time by using two hash tables.

3 Locality Sensitive Hashing

Locality sensitive hashing (LSH) is an efficient technique for approximate kNN search problem. Because it serves as a foundation to the proposed method, we briefly introduce it as follows.

The core idea of LSH is to hash items in a similarity preserving way, i.e., it tries to store similar items in the same buckets, while keeping dissimilar items in different buckets. In general, LSH method for kNN search has two steps: training and querying. In the training step, LSH first learns a hash function $h(x) = \{h_1(x), h_2(x), \ldots, h_m(x) : h_i(x) \in Z\}$ where m is the code length. For example, in the binary coding using linear projection, LSH adopts the hash function of form $h_i(x) = \mathrm{sgn}(w_i^T x + b_i) \in \{-1, 1\}$, where $\{w_i, b_i\}_{i=1,\ldots,m}$ are parameters to be learned. Then, LSH represents each item in the database as a hash code by the hash mapping $h(x)$, and constructs a hash table by hashing each item into the bucket indexed by its code. In the querying step, LSH first converts the query into a hash code, and then finds its approximate kNN in the bucket indexed by the code. One attractive feature of the LSH method is that it enjoys a rigorous theoretical performance guarantee even in the worst case [18]. In practice, it can provide constant or sub-linear search time.

Although the classic LSH method builds hashing codes by random projection, recent works focus on learning data-dependent hash functions so as to generate more accurate and compact hash codes to accelerate the query. As many machine learning techniques, such LSH methods can be divided into three

main categories: supervised methods [24, 4], semi-supervised methods [29] and unsupervised methods [31, 23, 20].

Most of LSH methods described before only work for l_2 similarity measure. To apply LSH for different similarity measure, researchers have designed different hash functions, such as l_p [10], Mahalanobis distance [22], kernel similarity [21], and χ^2 distance [14]. This provides another motivation of our method: according to the problem at hand, we can conveniently choose from these methods to group data items.

4 *k*NN Graph Construction with LSH

4.1 Problem Definition

Given a set of n items $S = \{x_1, x_2, \ldots, x_n\}$ and a similarity measurement $\rho(x_i, x_j)$, the kNN graph for S is a directed graph that there is an edge from node i to j if and only if x_j is among x_i's k most similar items in S under ρ. Here, ρ could be any similarity measurement defined on domain S. For example, it can be cosine similarity, kernel similarity, Mahalanobis distance etc.

4.2 Algorithm

The key idea of our method is to divide the whole data set into small groups, then find each item's kNN within the group it belongs to. Since it is very hard to ensure that each item and its real kNN are located in the same group, the method only provides an approximate result. From the perspective of graph construction, it is equivalent to say the method constructs one kNN graph on each group, and then takes the union of all these small kNN graphs as the approximation kNN graph.

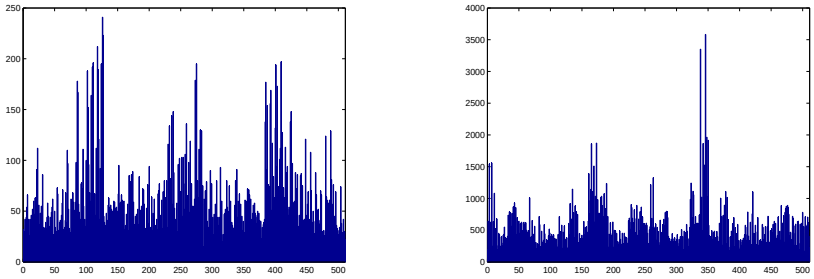
Since the kNN search is performed within a small subset, finding one item's kNN needs only block-sz comparisons, where block-sz is the size of the group it locates in. Assuming all groups are of equal size, the method's complexity is $O(\text{block-sz} \times n)$ instead of $O(n^2)$ for the brute-force manner.

To make the strategy work well, two conditions should be satisfied:

1. Similar items should be grouped together, which implies most of the real kNN of an item can be found in its group. Therefore, one can expect the resulting graph is a good approximation to the true kNN graph.
2. Since the complexity of the method is $O(\text{block-sz} \times n)$, to make the method efficient, each group should be as small as possible (block-sz $\ll n$).

Obviously, there is a contradiction between the two conditions: to make Condition **1** valid, we tend to use groups of big size which will violate Condition **2**. Thus, a balance should be made by choosing a feasible block-sz.

In this work, we explore LSH to divide the data set with the hope that similar items will be grouped together. A straightforward way is to use LSH to hash all items into a hash table, then construct a kNN graph for each bucket. However,



(a) Caltech256 (b) NUS-WIDE

Fig. 1. Distribution of the number of items in buckets. Experiments are performed on Caltech256 and NUS-WIDE data sets. We use the function $h(x) = \{\text{sgn}(w_i^T x + b_i)\}_{i=1,\ldots,8}$ to hash all items into a hash table, where $\{w_i\}$ are sampled from a Gaussian distribution $N(0, I)$ and $\{b_i\}$ are the median values of the projections. Since $m = 8$, the hash table has 512 buckets.

typical LSH methods yield highly un-even hash table which means some buckets contain a large number of items and some contain few items as shown in Fig 1. The problems brought by this phenomenon are: (a) the kNN graph constructed on a small bucket will fail the Condition **1**, and suffer a poor precision; (b) Constructing a kNN graph on a large bucket will fail the Condition **2**, and suffer a high cost because of the complexity $O(\text{block-sz}^2)$. Actually, the direct use of LSH has been observed to result in unsatisfying performance[12, 30], which is also verified by our experiments.

We propose an efficient way to overcome the problem and obtain equal size groups. Given the data set's hash code matrix $Y \in \{0,1\}^{n\times m}$ where the ith row $y_i \in \{0,1\}^m$ is the hash code of x_i, we first project items' hash codes onto a random direction $w \in \Re^m$, and get $p = Yw$. Then we sort items by their projection values to get the sequence $\{x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_n}\}$ with $\{p_{\pi_1} \leq p_{\pi_2} \ldots \leq p_{\pi_n}\}$. Finally, we obtain $n/\text{block-sz}$ groups $\{S_i\}$ with equal size of block-sz by defining $S_i = \{x_{\pi_{(i-1)\times block-sz+1}}, \ldots, x_{\pi_{i\times block-sz}}\}$. Because items in the same bucket will have same projection values, they will remain in the same group with high probability. We summarize the procedure described above in Algorithm 1.

Algorithm 1 generates a basic approximation to the kNN graph. However, it is just an union of $n/\text{block-sz}$ isolated small graphs, and may suffer from a low accuracy. To improve it, we repeat Algorithm 1 using different hash functions for multiple times and then combine the resulting graphs. Denoting the approximate kNN of x found in the i-th iteration by $N_i(x)$, we can obtain at most $k \times l$ NN different candidates as $\{N_i(x)\}_{i=1,\ldots,l}$ after l iterations. Obviously, by increasing l, it will cover more and more true kNN of x. The final graph is achieved by connecting each x to k items in $\{N_i(x)\}_{i=1,\ldots,l}$ which are the nearest to it.

Function: basic_ann_by_lsh(X, k, m, block-sz)
begin
  $Y = LSH(X, m)$
  Project Y onto a random direction w, $p = Yw$
  Sort items by p values, and get $\{x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_n}\}$
  for $i = 1, \ldots, n/$*block-sz* **do**
    $S_i = \{x_{\pi_{(i-1)\times block-sz+1}}, \ldots, x_{\pi_{i\times block-sz}}\}$ $g_i =$ brute_force_kNN(S_i, k)
  return: $G = \bigcup\{g_i\}$

Algorithm 1: Basic kNN Graph Construction with LSH

Algorithm: Approximate kNN Graph Construction with LSH
Input: X, k, l, m, block-sz
begin
  for $i = 1, \ldots, l$ **do**
    $G_i =$ basic_ann_by_lsh(X, k, m, block-sz)
  Combine $\{G_1, G_2, \ldots, G_l\}$ to get G
  Refine G by one step neighbor propagation
  return: G

Algorithm 2: Main Algorithm

To further boost our method, we use a one-step neighbor propagation procedure which is widely used in efficient kNN graph construction methods [6, 12, 30]. It is based on the following observation: if x is similar to y and y is similar to z, then it is likely that x is similar to z. It implies that we could improve an item's approximate kNN by selecting from its neighbors and the neighbors of its neighbors. Formally, denoting the approximate kNN of x found by now as $N(x)$, we update it by reselecting x's kNN from the set $N(x) \cup \{\cup_{v\in N(x)} N(v)\}$. As we will see in the next section, this simple local search procedure is a good complementary strategy with LSH, and gives significant improvement to the final results. The whole method is summarized in Algorithm 2.

In the proceeding of multiple basic graph constructions and neighbor propagation, some pairwise distances could be computed for many times. Thus, another technique to speedup the method is to store all the pairwise distances that have been computed so far in a hash table. When requiring the distance between two items, we first check the hash table if it has been evaluated before. However, we do not actually adopt this technique as the hash table may be too large to store. For example, assuming $n = 1,000,000$ and 10% of all the n^2 pairwise distances have been computed, the hash table needs about 8 GB memory to store these values.

4.3 Complexity Analysis

We briefly analyze the complexity of the proposed method as follows.

1. Complexity for computing LSH(S,k), projecting and sorting are $O(nmd)$, $O(nm)$ and $O(n \log n)$ respectively. Complexity for the construction of all g_i ($1 \leq i \leq n/$block-sz) is $O(nd$block-sz). Thus, the complexity of Algorithm 1 is $O(n(md + m + d\text{block-sz}) + n \log n)$.
2. Combining l graphs needs $O(lnk)$ operations.
3. Each item has at most $k^2 + k$ candidate neighbors (k neighbors and k^2 neighbors of neighbor), and needs $O(dk^2)$ operations to find out the k nearest neighbors. Thus, the complexity of neighborhood propagation step is $O(ndk^2)$

Summarizing the above analysis, the complexity of our algorithm is $O(ln(md + d\text{block-sz} + \log n + k) + ndk^2)$. Since k, m and block-sz are small in practice, the actual complexity is $O(ln(d + \log n))$.

5 Experiments

To empirically validate the effectiveness of the proposed approximate kNN graph construction method, we provide results on several data sets. The primary goal is to verify: (1) The proposed method is much faster than existing ones; (2) Our method can generate good approximate graph by scanning only a small part of the total pairwise distances; (3) For many classification tasks, an approximate graph is enough to achieve good classification accuracy.

5.1 Experimental Setting

Data Set. We use 3 popular image data sets to evaluate the proposed method: Caltech256, Imagenet and NUS-WIDE. Caltech256 [16] is a benchmark for object classification, while Imagenet [11] and NUS-WIDE [8] are two real-world web image databases. Features are extracted by the bag-of-word (BOW) model based on SIFT descriptions. For Caltech256, we perform k-means clustering of SIFT descriptors to form a visual vocabulary of 1024 visual words. Then, SIFT descriptors are quantized into visual words using the nearest cluster center. BOW features for Imagenet and NUS-WIDE are directly downloaded from websites[4][5]. For the Imagenet database, we only adopt the first 100 categories of the training set. We summarize the size and dimensionality of the data sets in Table 1.

Table 1. Data sets description.

	Caltech256	Imagenet	NUS-WIDE
Size	30607	120955	269643
Dimensionality	1024	1000	500

[4] http://www.image-net.org/challenges/LSVRC/2010/download-public
[5] http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm

Comparison Methods. We compare our method, denoted by LSH, with 3 popular approximate kNN graph construction methods: OverTree [6], MultiTree [30] and NN-Descent [12]. We employ the direct use of LSH to construct approximate kNN graph as the baseline method [6], denoted by DirectLSH. To examine the effect of the one-step neighbor propagation, we will also report the results of our algorithm without the neighbor propagation procedure, denoted by LSH–. The setting for different methods is given as follows.

1. For LSH, LSH– and DirectLSH, we adopt a LSH coding method proposed in [23] to divide data points. It is a nonlinear method based on spectral decomposition of a low-rank graph matrix. To efficiently generate different hash tables, we randomly sample 20 items as the anchor points for each iteration. The length of the hash code is set to $m = \lceil \log_2(n/\text{block-sz}) \rceil + 1$. For LSH and LSH–, block-sz (the maximum set size for performing brutal force kNN) is set to 100.
2. For OverTree, the specific algorithm we use is kNN-glue, and the codes are provided by authors [7]. block-sz is set to 100.
3. For MultiTree, we randomly sample 20 points on each internal node, compute the principal direction on the sampled set by Lanczos algorithm, then use it to perform the projection. block-sz is set to 100.
4. For NN-Descent, there is no hyperparameter to set, and the codes of NN-Descent are provided by authors [8].

Evaluation Criterion. To evaluate the quality of an approximate kNN graph G', we define the following accuracy measurement $acc(G') = \frac{|E(G') \cap E(G)|}{|E(G)|}$, where G is the exact kNN graph, $E(\cdot)$ denotes the set of the directed edges in the graph and $|\cdot|$ denotes the cardinality of the set. The exact kNN graph is computed by the brute-force method, and the time for constructing 10NN graph is shown in Table 2.

Table 2. 10NN graph construction time for the brute-force method.

	Caltech256	Imagenet	NUS-WIDE
Time (sec.)	1108	17991	43696

Experiment Environment. All experiments are run on a Linux server with two 8-core 2.66GHz CPUs and 128G RAM. Parallelization is disabled for all method, and each method is restricted on one thread.

5.2 Experimental Results

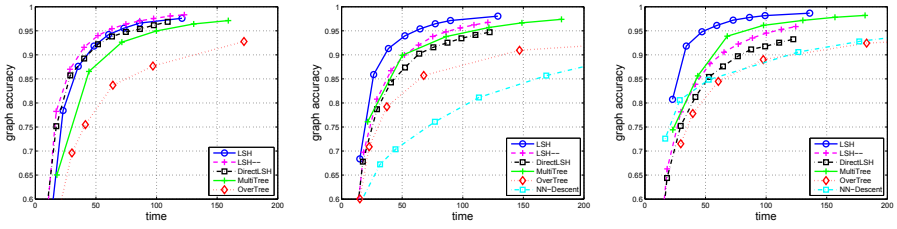
Time versus Approximate Accuracy. We test various methods by comparing their time for constructing graphs with different accuracies. To generate

[6] In each iteration, we use LSH to group data, then construct one kNN graph for each bucket.

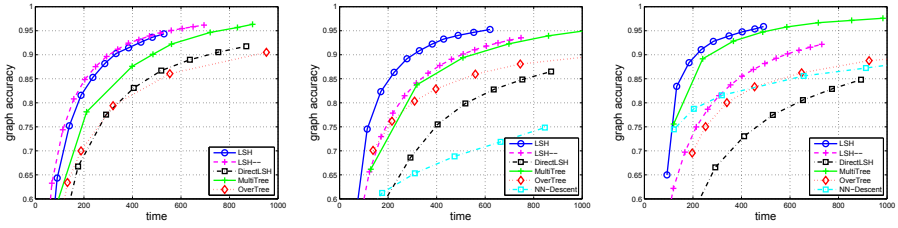
[7] http://www.mcs.anl.gov/ jiechen/software.html#knn

[8] http://code.google.com/p/nndes/

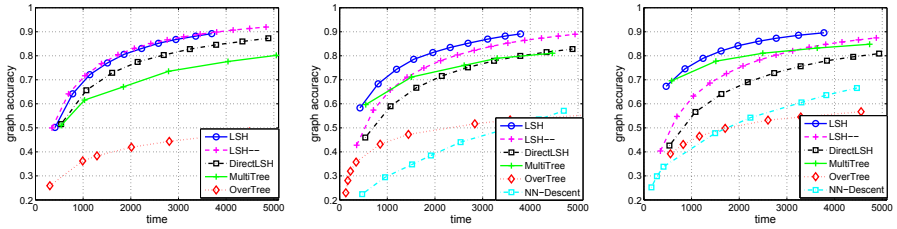
different accuracy graphs, (1) For LSH, LSH–, DirectLSH and MultiTree, we use different times of divisions (parameter l). (2) For OverTree, we vary the overlapping factor from [0.1 0.5]. (3) For NN-Descent, we use different sample rates. The performance comparison is shown in Fig 2. The horizontal axis represents



(a) Caltech256



(b) Imagenet100



(c) NUS-WIDE

Fig. 2. Graph accuracy versus construction time for different methods on 3 data sets. The 3 columns correspond to the results of building approximate 1NN, 10NN and 20NN graphs respectively.

the time (in seconds) consumed for graph construction, and the vertical axis represents the graph accuracy defined above. Each row of the figure corresponds to one data set, and each column corresponds to a choice of k value. As the NN-Descend method fails for small k, its performance curve is not shown for $k = 1$.

From Fig 2, we can clearly see the superiority of our method. For $k = 10$ and $k = 20$, LSH is consistently better than LSH–, and has similar performance for $k = 1$, which shows the effectiveness of the neighbor propagation procedure. On

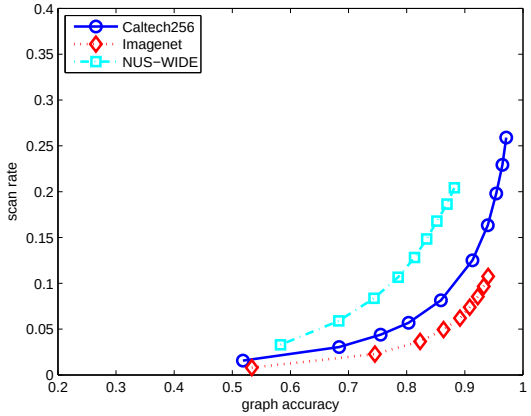


Fig. 3. Scan rate of the proposed method with respect to different graph accuracy on 3 data sets($k = 10$).

the other hand, the performance of LSH– is always better than DirectLSH, which shows the effectiveness of the equal size splitting. In fact, the accuracy of one basic isolated graph produced by DirectLSH is higher than that of LSH–, but it takes DirectLSH about 2 3 time than LSH–. Thus, in the same time our method builds more basic graphs than DirectLSH, and obtains higher accuracies.

Our method is at least 4 times faster than OverTree and NN-Descent methods on all data sets. MultiTree exhibits similar performance on Caltech256 and Imagenet, but is dramatically slower than the proposed method on NUS-WIDE. Furthermore, MultiTree and OverTree are only applicable for Euclidean space with l_2 norm. Compared with the brute force method, our approach generates graphs with accuracies of 90% using at most 5% time of the brute force method on Caltech256 and Imagenet, and using less than 10% time of the brute force method on NUS-WIDE.

Approximate Accuracy vs. Scan Rate. Because the dominant cost of kNN graph construction is the pairwise distance calculation, we evaluate our method by reporting the ratio of the number of actual distances calculated to the total number of pairwise distances $n(n-1)/2$. The values for graphs of different accuracies under different data sets are shown in Fig 3. We can observe that: (1) Our method generates good approximate kNN graphs by scanning only a small proportion of the distances, but to generate graphs with high accuracy, the method still has to scan a large number of distances. This is because: neighbors of some items are far away, and locality-sensitive hashing in nature is not good at searching for such points. (2) Different data sets have very different properties. For example, constructing approximate kNN graphs for Imagenet is much easier than for NUS-WIDE.

In experiments, we found the scan rate of MultiTree is even lower than ours. However, because of computing principle directions, in constructing a basic approximate graph, it takes about 3 times of the running time of Algorithm 1.

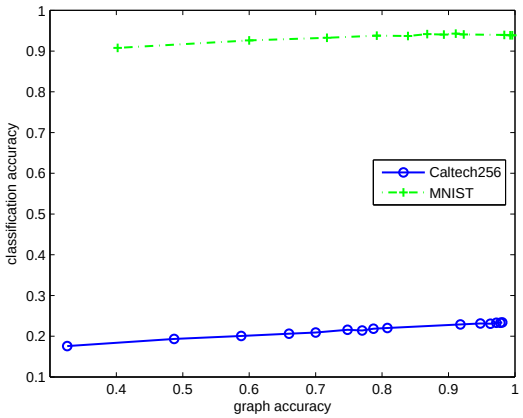


Fig. 4. Variance of classification accuracy with respect to graph accuracy.

Thus, its overall speed is slow. Actually, by increasing the number of anchor points and using kmeans to choose anchor points (see experimental setting for LSH), our method can also achieve better scan rate. But this will make the cost of learning the hash coding much higher, and decrease the overall performance. Therefore, instead of pursuing the accurate but complex indexing structure as in kNN search, a trade-off must be made between the speed and accuracy in building the indexing structure for the graph construction problem.

Approximate Accuracy vs. Classification Accuracy. As we shown in the previous experiments, it is relatively easy to yield a good approximate graph, but hard to obtain graph with very high accuracy. The progress towards the real kNN graph becomes slow when l is large. Actually, this is a common problem shared by all approximate kNN graph construction methods. But fortunately for machine learning tasks, like classification, what we really care is not graph accuracy, but the classification accuracy. As long as edges found by our method are locally enough, we can expect the approximate graph leads to good description to the data set, and therefore yields similar results as the true kNN graph.

In this experiment, we examine the variance of classification accuracy with respect to the graph accuracy. First, approximate 20NN graphs with different accuracies are generated by our method on Caltech256 and MNIST(a popular handwritten digit data set with $n = 70000$, $d = 784$). We randomly sample 15420 nodes for Caltech256 and 700 nodes for MNIST as labeled data, and use Gaussian random fields [33] to classify the rest nodes. The classification accuracies are reported in Fig 4.

As observed from the results, the classification accuracy is very robust with the graph accuracy. Actually, on Caltech256, the classification accuracy is 22.0% on graph of accuracy 80%, while is 23.4% on exact 20NN graph; on MNIST, the classification accuracy is 93.8% on graph of accuracy 80%, while is 93.9% on exact 20NN graph. This implies that for many applications, it is sufficient to construct a graph with a reasonable accuracy.

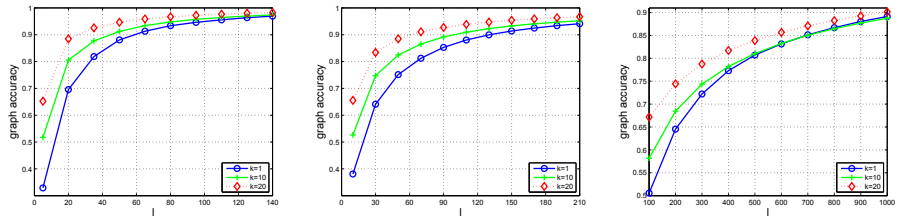


Fig. 5. Variance of classification accuracy with respect to the parameter l for building approximate 10NN graphs on Caltech256, Imagenet and NUS-WIDE respectively.

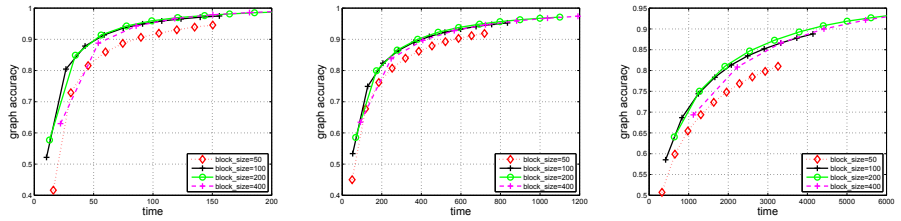


Fig. 6. Effect of the size of block for building approximate 10NN graphs on Caltech256, Imagenet and NUS-WIDE respectively.

Approximate Accuracy vs. the Number of Divisions. In the proposed method, we use the number of divisions l to control the speed and accuracy of graphs. In this experiment, we examine how the graph accuracy varies with the number of divisions. For each data set, we construct $k = \{1, 10, 20\}$ NN graphs with different l, and report the accuracy of resulting graphs. The results are shown in Fig 5.

Effect of the size of block. From the analysis in Section 4.2, we should choose a feasible block size to make a balance between the accuracy and the cost of basic graph. In this experiment, we examine how the block size affects the performance of our method. We construct approximate 10NN graphs on Caltech256, Imagenet and NUS-WIDE data sets with block-sz $\in \{50, 100, 200, 400\}$. For each data set, we build graphs with different accuracies by using different number of divisions, and report the building time and accuracies in Fig 6. As we can see, block-sz $\in \{100, 200\}$ are consistently better than $\{50, 400\}$ for all data sets. This is because block-sz$=$ 50 is too small to generate a good basic graph while the cost for building basic graph with block-sz$=$ 400 is too high. Actually, for all the previous experiments, we have fixed block-sz to 100.

6 Conclusion

In this paper, we have proposed a fast approximate kNN graph construction method for generic similarity measures. It engages the LSH technique to efficiently partition items into small groups, and then constructs kNN graph on each group. As shown by the experiments, our method can efficiently generate

graphs with good accuracies by just computing a small proportion of distances, which is sufficient for many learning tasks. In addition to its high efficiency, the proposed algorithm is ready to be applied under many similarity measures, and is also natural to be parallelized.

Acknowledgements
This works have been supported in part by the National Basic Research Program of China (973 Program) Grant 2012CB316301, the National Natural Science Foundation of China (NSFC) Grants 61203296, 61075052 and 61005029, Tsinghua National Laboratory for Information Science and Technology (TNList) Cross-discipline Foundation.

References

1. M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.
2. J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
3. J.L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.
4. M.M. Bronstein and P. Fua. LDAHash: Improved matching with smaller descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):66–78, 2012.
5. Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 2002.
6. J. Chen, H. Fang, and Y. Saad. Fast approximate kNN graph construction for high dimensional data via recursive lanczos bisection. *The Journal of Machine Learning Research*, 10:1989–2012, 2009.
7. B. Cheng, J.C. Yang, S.C. Yan, Y. Fu, and T. Huang. Learning with l1-graph for image analysis. *IEEE Transaction on Image Processing*, 19:858–866, 2010.
8. Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao. Zheng. NUS-WIDE: A real-world web image database from national university of singapore. In *Proceedings of ACM Conference on Image and Video Retrieval*, 2009.
9. S.I. Daitch, J.A. Kelner, and D.A. Spielman. Fitting a graph to vector data. In *Proceedings of the International Conference on Machine Learning*, 2009.
10. M. Datar, N. Immorlica, P. Indyk, and V.S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Annual Symposium on Computational Geometry*, 2004.
11. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.
12. W. Dong, M. Charikar, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the International Conference on World Wide Web*, 2011.
13. A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Data Bases*, 1999.

14. D. Gorisse, M. Cord, and F. Precioso. Locality-sensitive hashing for chi2 distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(2):402–409, 2012.
15. Amit Goyal, Hal Daumé III, and Raul Guerra. Fast large-scale approximate graph construction for nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012.
16. G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical report, California Institute of Technology, 2007.
17. M. Herbster, M. Pontil, and S. R. Galeano. Fast predciton on a tree. In *Advances in Neural Information Processing Systems*, 2008.
18. P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 1998.
19. T. Jebara, J. Wang, and S.F. Chang. Graph construction and b-matching for semi-supervised learning. In *Proceedings of the International Conference on Machine Learning*, 2009.
20. W. Kong and W.J. Li. Isotropic hashing. In *Advances in Neural Information Processing Systems*, 2012.
21. B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision*, 2009.
22. B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2143–2157, 2009.
23. W. Liu, J. Wang, S. Kumar, and S.F. Chang. Hashing with graphs. In *Proceedings of the International Conference on Machine Learning*, 2011.
24. R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.
25. J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2002.
26. J.K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.
27. P.M. Vaidya. An O(nlogn) algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry*, 4(1):101–115, 1989.
28. U. Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.
29. J. Wang, S. Kumar, and S.F. Chang. Semi-supervised hashing for scalable image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.
30. J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan, and S. Li. Scalable k-NN graph construction for visual descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.
31. Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, 2008.
32. Y.M. Zhang, K. Huang, and C.L. Liu. Fast and robust graph-based transductive learning via minimum tree cut. In *IEEE International Conference on Data Mining*, 2011.
33. X. Zhu. Semi-supervised learning literature survey. Technical report, Computer Science, University of Wisconsin-Madison, 2008.
34. X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the International Conference on Machine Learning*, 2003.