Query-driven iterated neighborhood graph search for scalable visual indexing*

Jingdong Wang† Xian-Sheng Hua‡ Shipeng Li†
†Microsoft Research Asia ‡Microsoft Corporation

August 10, 2012

Abstract

In this paper, we address the approximate nearest neighbor (ANN) search problem over large scale visual descriptors. We investigate a simple but very effective approach, neighborhood graph (NG) search, which conducts the local search by expanding neighborhoods with a best-first manner. Expanding neighborhood makes it efficient to locate the descriptors with high probability being true NNs. However, it suffers from the *local* characteristics, and often gets sub-optimal solutions, or conducts exhaustive and continuous neighborhood expansion to find better solutions which deteriorates the query efficiency.

We propose a query-driven iterated neighborhood graph search approach to improve the performance. We follow the iterated local search (ILS) strategy widely-used for combinatorial and discrete optimization in operation research, and handle the key issue in applying ILS for neighborhood graph search, Perturbation, which generates a new pivot to restart a local search. The key novelties lie in two-fold: (1) defining the local solution of ANN search over neighborhood graph; (2) presenting a query and search history driven perturbation scheme to generate pivots to restart a new local search. The main benefit from them is avoiding unnecessary neighborhood expansion and hence more efficiently finding true NNs. Experimental results on large scale SIFT indexing and similar image search with tiny images show that our approach performs much better than previous state-of-the-art ANN search approaches.

1 Introduction

Large scale visual indexing has been attracting more and more interest in computer vision and multimedia search. It relies much on efficient and effective approximate nearest neighbor search. For instance, the state-of-the-art duplicate image search depends on the bag-of-words feature, which usually exploits ANN search techniques to group large scale local features into visual words using k-means or similar algorithms [27] and map visual features to visual words. Other examples include finding the best matches for local image features in large data sets [28] and large scale similar image search [16].

In the communities of computer vision and multimedia, there exist two main categories of ANN search schemes: partitioning trees and hashing. Partitioning trees, including kd-trees [4, 10] and its variants [2, 3, 13, 26], metric trees (e.g., ball trees [18], vantage point trees (vp-tree) [34], spill-tree [17]), and hierarchical k-means tree [21], organize data points using tree structures by recursively partitioning the space into subspaces, and each subspace, associated with a subset of data points, corresponds to a subtree. The query procedure is to traverse the tree by expanding the subtrees (subspaces) down to the leaf nodes to get NN candidates, in some order, e.g., depth-first or best-first. Such a scheme takes too much time overhead to locate the subspace that corresponds to a leaf node, and the search order typically is not effective enough to make the searching path quickly move towards the true NNs.

Hashing based approaches include locality sensitive hashing (LSH) [6], spectral hashing [32], and other variants [11, 16]. The query procedure has to check a large number of NN candidates from the buckets that correspond to the same (or similar) hash codes with the query point, to guarantee the accuracy. However, it does not discriminate the points in the buckets and has no optimized search order to access them, which

*This technical report was written at the early of 2011.

leads to high time cost on checking the points with low probability to be the true NNs. The paper [6] suggests to increase gradually the radius R in R-NN search to obtain the approximate NN, which is shown to be much slower than partitioning trees [19] .

In this paper, we adopt the neighborhood graph structure to index visual descriptors. The intuition is that the points, around a point that is close to the query, have high probability to be also close to the query. The search procedure starts from one or several data points and conducts a best-first strategy to first check the neighborhood of the best NN candidate among those whose neighborhoods are not expanded yet. The advantages include that the time overhead to access a new NN candidate is significantly reduced, only $O(1)$, and particularly the order to access the candidates determined from the neighborhood graph is better than partitioning trees and hashing based methods. Fig. 1 presents the comparison between the neighborhood graph and partitioning trees to demonstrate such advantages. However, it suffers from the *local* characteristics, and often gets sub-optimal solutions that is not acceptable in real applications (e.g., SIFT matching), or conducts exhaustive and continuous neighborhood expansion to find better solutions which deteriorates the query efficiency.

We propose a query-driven iterated neighborhood graph search approach to deal with the local issue. We follow the widely-used iterated local search (ILS) strategy for combinatorial and discrete optimization in operation research, and handle the key issue, designing Perturbation to generate a new pivot to restart a local search. The key novelties lie in two-fold: (1) defining the local solution over neighborhood graph for ANN search; (2) presenting a query and search history driven perturbation scheme to generate pivots to restart a new local search. The main benefit from them is avoiding unnecessary neighborhood expansion and more efficiently finding true NNs. The experimental results of ANN search for visual descriptors demonstrate that our approach gets the superior performance. Particularly, when searching higher-dimensional visual descriptors and requiring higher search accuracy, the superiority of our approach over existing ANN search algorithms is more significant.

2 Related work

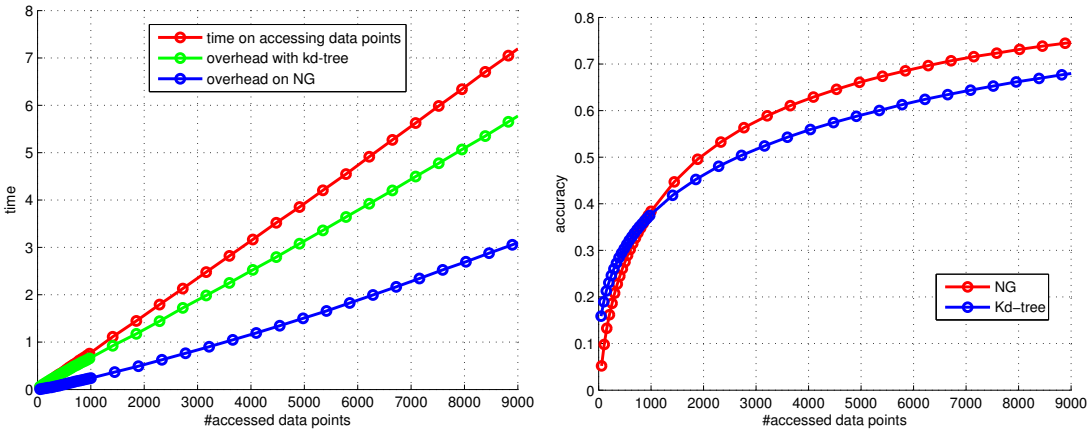
In the literature of computer vision and computational geometry, the ANN search methods include two main categories, partitioning tree and hashing, as well as other methods, e.g., low dimensional embedding. In the following, we will present a review on ANN search methods that are widely investigated and explored in computer vision. More descriptions could be found from [23].

Partitioning trees The partitioning tree based approaches recursively split the space into subspaces, and organize the subspaces in a tree (called space partitioning tree). Most space-partitioning systems use hyperplanes or hyperspheres to divide the space, and data points are accordingly partitioned into two subsets, with each lying in one side. Points exactly on the plane are usually arbitrarily assigned to either side. Recursively partitioning space in this way produces a binary partitioning tree, for example, kd-trees [4, 10] and its variants [2, 3, 13, 26], and metric trees (e.g., ball trees [18], vantage point trees (vp-tree) [34], and spill-trees [17]). Using other criteria to split the space may yield multi-way partitioning trees, such as Quadtrees [9], Octrees [33], hierarchical k-means trees [21] and so on, which are mainly for low-dimensional cases and hence not proper for ANN search over high-dimensional visual descriptors.

In the query stage, the branch-and-bound methodology [4] is usually applied to searching (approximate) nearest neighbors. This scheme needs to traverse the tree from the root to a leaf by evaluating the query at each internal node, and pruning some subtrees according to the evaluation and the current nearest neighbors. The current state-of-the-art search strategy, priority search [2] or best-first [3], builds a priority queue to access each subtree in the order so that the data points with large probability to be the true nearest neighbors are first accessed.

Recently, kd-tree based ANN search methods are widely investigated for computer vision applications. The scheme about multiple randomized kd-trees is studied in [26]. Partitioning the space, using the trinary combination of the coordinates as the partitioning plane, expects to get better space partitions with still low time cost to locate a leaf node [13].

Hashing Hashing based ANN search methods also attract a lot of attention. Locality sensitive hashing (LSH) [6], one of the typical representatives, is a method of performing ANN search in high dimensions, dependent on probabilistic dimension reduction of high-dimensional data. The key idea is to hash the



(a) (a)

Figure 1: Illustrating the superiority of ANN search with a neighborhood graph over with multiple randomized kd-trees. (a) comparison of #accessed data points vs. average search time, and (b) comparison of #accessed data points vs. search accuracy.

points using several hash functions to ensure that for each function the probability of collision is much higher for objects (points) that are close to each other than those far apart. Then, one can determine near neighbors by hashing the query and retrieving elements stored in the buckets containing it. LSH has been widely applied in computer vision, e.g., for pose matching, contour matching, and mean shift clustering, a literature review could be found in [25]. Recently, a lot of research efforts have been conducted on finding good hashing functions, by using metric learning-like techniques, including optimized kernel hashing [11], learned metrics [12], learnt binary reconstruction [15], kernelized LSH [16], and shift kernel hashing [22], semi-supervised hashing [31], spectral hashing [32]. The learning based strategy actually can also be used in our approach. However, this paper would not make investigation on it, and instead focuses mainly on the search strategy. Hashing based algorithms may be good to guarantee both the precision and recall performance, but is poor at the query efficiency [19].

There are some other methods for ANN search. LSH essentially is also an embedding method, and other classes of embedding methods include Lipschitz embedding [14] and FastMap [7]. Neighborhood graph methods are another class of index structures. Unlike a tree structure hierarchically organizing subspaces or subsets of data points, it organizes the data with a graph structure to connect data points, for example, Delaunay graph in Sa-tree [20], relative neighborhood graph [30], and k-NN (ϵ-NN) graph [24]. The focus of this paper does not lie in the neighborhood graph construction, and for convenience a (connected) k-NN graph is adopted. To conduct ANN searches, additional points (called pivots) are required as the starting points to guide the neighborhood graph search, for example, selecting the pivots, by clustering [24] (query independent) or from the first NN candidate of kd-trees [1] (query dependent). These approaches, however, suffer from the local characteristics. This paper proposes a query-driven iterated neighborhood graph search approach to address such a problem.

3 Query-driven iterated neighborhood graph search

Given a set of n data points $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathcal{R}^k$ being a k-dimensional point, the goal is to build a data structure to index these points so that the nearest neighbors of a query $\mathbf{x}_q$ can be fast found. A neighborhood graph is a directed graph to organize data points by connecting each data point with its neighboring points. The neighborhood graph is denoted as $G = \{(v_i, Adj[v_i])\}_{i=1}^n$, where v_i corresponds to a point $\mathbf{x}_i$ and $Adj[v_i]$ is a list of nodes that correspond to its neighbors.

Local neighborhood graph search Local neighborhood graph search for ANNs is propagating the search by continuously accessing their neighbors from seeds to traverse the graph. The *best-first* strategy is usually adopted for local search. To this end, a *priority queue* is used to maintain the accessed points, and initially

Algorithm 1 Iterated local search

1. $s_0 \leftarrow$ GenerateInitialSolution
2. $s^* \leftarrow$ LocalSearch(p_0)
3. **repeat**
4. $\quad s' \leftarrow$ Perturbation$(s^*,\ history)$
5. $\quad s^{*\prime} \leftarrow$ LocalSearch(s')
6. $\quad s* \leftarrow$ AcceptanceCriterion$(s*,\ s^{*\prime},\ history)$
7. **until** termination condition met

Algorithm 2 Query-driven iterated neighborhood graph search

1. $P_0 \leftarrow$ GenerateInitialSolution(q, T)
2. $R^* \leftarrow$ LocalNGSearch$(P_0,\ G)$
3. **repeat**
4. $\quad P' \leftarrow$ Perturbation$(q,\ T,\ history)$
5. $\quad R^{*\prime} \leftarrow$ LocalNGSearch$(P',\ G,\ history)$
6. $\quad R^* \leftarrow$ AcceptanceCriterion$(R^*,\ R^{*\prime})$
7. **until** termination condition met

contains only seeds. The current best candidate in the priority queue is extracted out, and the points in its neighborhood is first expanded and pushed into the priority queue. The resulting search path may not be monotone, but always attempts to move closer to the query point without repeating points. Due to the local property, the search will be stuck at a locally optimal point and has to conduct exhaustive neighborhood expansion to find better solutions. We present a query-driven iterated neighborhood graph search to deal with this issue.

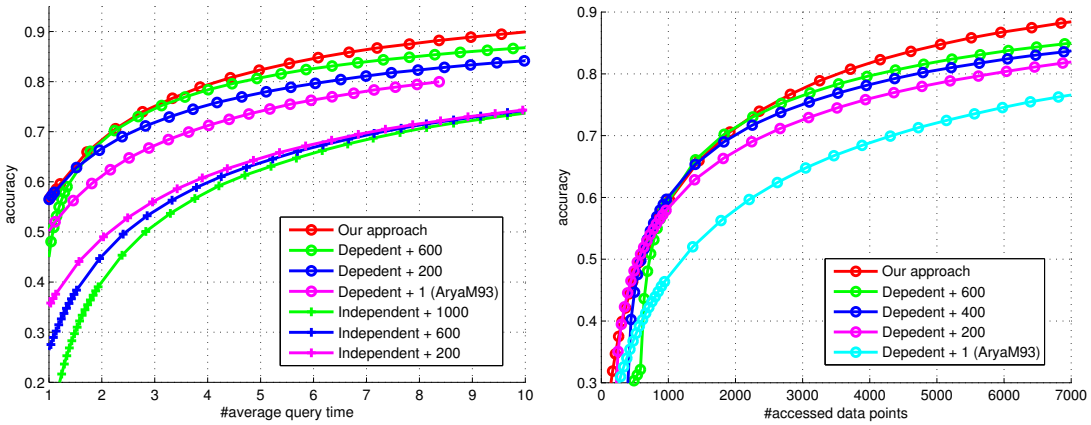
Iterated local search A simple modification consists of iterating calls to the local search routine, each time starting from a different initial configuration. This is called repeated local search, and implies that the knowledge obtained during the previous local search phases is not used. In contrast, iterated local search is based on building a sequence of locally optimal solutions by perturbing the current local minimum and applying local search after starting from the modified solution. The standard procedure is as Algorithm 1.

3.1 Query-driven iterated search

Our approach improves the local neighborhood graph search from two aspects: iterated search and query-driven. The basic procedure is shown in Algorithm 2. LocalNGSearch$(P_0,\ G)$ is similar to the traditional local NG search, starting from seeds P_0 and searching over G, but differently once reaching a location solution, the NG search will pause and return current NNs. Perturbation$(q,\ T,\ history)$ generates new seeds from trees T according to search history. LocalNGSearch$(P',\ G,\ history)$ is different from LocalNGSearch$(P_0,\ G)$ as the search history in the previous iterations, i.e., the NNs found up to the current iterations, is considered in the search.

The initial seeds guiding the neighborhood graph search are generated by exploring both the query and the reference data points. We build a data structure, e.g., random kd-trees (T) in our implementation, and search for a few points from the random kd-trees with priority search as the initial seeds. Generating seeds in such a query-dependent way can get better performances compared with query-independent based methods. By comparison, we show the results of a query-independent manner, clustering reference data points to find seeds [24]. The comparison is shown in Fig. 2(a). As expected, query-dependent methods perform better than query-independent methods. Merely query-dependent seeds still suffer from be stuck at local solutions. Fig. 2 also shows the comparison between our approach and query-driven non-iterated approaches. It demonstrates that the query-driven iterated approach indeed requires to access fewer data points (Fig. 2(b)) and accordingly take less time (Fig. 2(a)) to get the same search accuracy. The following describes the details on the iteration: local solution inspection, perturbation and termination condition.

Given the neighborhood graph $G = \{(v_i, Adj[v_i])\}_{i=1}^n$ and the query point $\mathbf{x}_q$, the distances of this query to vertices can be written as a function over the graph vertices, $f(v_i; \mathbf{x}_q)$. In the continuous case, a function $g(\mathbf{x})$ is said to reach a local minimum at the point $\mathbf{x}^*$, if there exists some $\epsilon > 0$ such that $g(\mathbf{x}^*) \leqslant g(\mathbf{x})$ when $|\mathbf{x} - \mathbf{x}^*| < \epsilon$. As an analogy, the distance function $f(v_i; \mathbf{x}_q)$ over the neighborhood



(a) (b)

Figure 2: (a) shows the comparison of average search time vs. search accuracy over query independent, query dependent and query driven iterated (our approach) schemes, and (b) shows the comparison of #accessed data points vs. search accuracy from the query dependent and query driven iterated (our approach) schemes.

graph is said to reach a local minimum at v_i^* if $f(v_i^*;\mathbf{x}_q) \leqslant f(u_i;\mathbf{x}_q)$ for all $u_i \in Adj[v_i^*]$. The local minimum can be inspected by checking the graph gradients, $\{f(u_i;\mathbf{x}_q) - f(v_i;\mathbf{x}_q)\}$, at v_i along the edges (u_i, v_i). In this paper, we call a point $\mathbf{x}_i$ corresponding to v_i^* as a promising point if there exists at least one graph gradient $f(u_i;\mathbf{x}_q) - f(v_i^*;\mathbf{x}_q) < 0$ at v_i^*, among u_is that belong to $Adj[v_i^*]$ and have not been accessed. We use an indication variable to record the number of unexpanded promising points among all the previously-checked points and then arriving at local solutions is equivalent to that it reaches zero.

Below is about perturbation, which depends on both the query and the search history. We still use trees, but perform the search that is slightly different from the conventional search such that it also considers the search history. Seeds generation in perturbation will be conducted by resuming the search in kd-trees that yields the initial solutions. But if using the same procedure with the conventional scheme, it will result in that many leaf nodes have already been accessed in the neighborhood graph search, and hence extra much time overhead is taken. To reduce the time overhead, but still get seeds to help jump out of the local solution, we propose to find only one leaf node as the seed from a subtree, called a compact subtree, which contains a set of points with the average similarity larger than a threshold. Our approach determines this threshold by cross validation. To this end, during the construction of the partitioning trees, an extra attribute is associated with each node of the trees, indicating the average similarity of the points within the corresponding subtree. After perturbation, the local neighborhood graph search is resumed.

The ANN search process will terminate if the number of accessed points reaches the threshold or the true nearest neighbors are found. The former condition is easy to interpreted. This is equivalent to stopping the search when the search time reaches a predefined threshold. The latter is challenging for conventional neighborhood graph search [1, 24]. In [24], the number of better points, similar to the definition of our promising point, is used to determine if the true nearest neighbors have been found. But it could not be theoretically guaranteed, and the experiments show that this stop condition is not reliable. In contrast, the iterated search scheme has a stop condition to guarantee that the true nearest neighbors are found. The condition is similar to the conventional search for exact NNs in trees, that's the best lower bound in the priority queue maintained for the search in trees is larger than the worst distance $\bar{d}$ in the previously-identified NNs.

3.2 Analysis

Since our approach is greedy, it is hard to give theoretic analysis. So we follow [1] to discuss the property over a random neighborhood graph. It has been proved [1] that the query cost is upper bounded:

Theorem 1 (From [1]). *Given any n point set $\mathcal{S} \in \mathbb{R}^d$ and any constant $\epsilon > 0$, one can search over*

a randomized neighborhood graph from a random seed so that $(1+\epsilon)$*-nearest neighbor queries can be answered in* $O(\log^3 n)$ *expected time.*

From the proof of Theorem. 1 presented in [1], we can have a lemma as follows.

Property 1. *In the good case, starting from an ideal seed, the expected query time of the search procedure is* $O(\log^2 n)$*. In the bad case, the expected query time is* $O(\log^3 n)$*.*

Suppose the former and later cases in Property. 1 appear with the probabilities P and $1-P$, then the expected query time is $O(P\log^2 n + (1-P)\log^3 n)$. (From the perspective of complexity analysis, $O(P\log^2 n + (1-P)\log^3 n) = O(\log^3 n)$, but our analysis differentiates them for the detailed analysis.) This suggests that the time cost could be reduced with carefully-selected seeds. Motivated by this, our approach aims to find a practical implementation toward a high chance to quickly locate the neighborhood of the optimal solution (the good case), i.e., increasing P. The time cost getting P can be roughly estimated as $O(r(P)\log n)$ with $r(P)$ representing the number of points accessed in trees to reach P.

Property 2. *The expected time cost of the search procedure with improving* P *by selecting the best seed from a number of seeds is* $O(r(P)\log n + P\log^2 n + (1-P)\log^3 n)$*.*

A possible way to improve P is query-driven non-iterative way, e.g., generating sufficient seeds at the beginning. This makes it uneasy to get a good balance between the cost improving P and the benefit from greater P as experiments shows that $r(P)\log n$ also plays an important role in practice although it can be ignored in theory. Therefore, we have proposed a query-driven iterated way to iteratively generate more seeds to improve P, which yields a better balance.

3.3 Implementation details

We present a speedup scheme to eliminate distance computation for the points that definitely cannot be the true NNs. To this end, we record the distances between neighboring points in the NG, which are used to estimate the lower bound of the distance to the query using the triangle inequality. Suppose the distance from the accessed point u to the query q is denoted by $d(u,q)$ and the precomputed distance from the point u to its neighbor point v is denoted by $d(u,v)$, the triangle inequality shows that $d(v,q) \geqslant |d(u,q) - d(u,v)|$. The distance computation $d(v,q)$ could be eliminated, if $d(p^*,q)(=\bar{d}) < |d(u,q) - d(u,v)|$ holds with $d(p^*,q)$ being the worst distance in the previously identified NNs. This elimination could save much time, especially for the high-dimensional case.

Besides, we introduce two practical schemes to balance the local NG search and perturbation for seed generation such that the advantages of local NG search and seed generation are well exploited. We observed that at the early NG search stage it is not easy to reach local solutions. This is because the seeds are still far away from the true NNs and hence the search path has a high chance to get closer to better NNs. As a consequence, there may be a low chance for the NG search to quickly jump toward the true NNs. To deal with this issue, we trigger the perturbation step, if the search has conducted a fixed number of successive neighborhood expansions failing to yield promising points.

At the later search stage, instead, it is found that the local search in the neighborhood graph arrives at local solutions frequently and the switch to search in trees is too frequent. The search at this stage mostly aims to conduct a finer search to get the true NNs. But the overhead of search in trees is larger than in the NG. Therefore, it is desired at this stage not to search trees too frequently. To this end, we introduce a scheme to balance the numbers of points accessed in trees and the NG so that the points from trees do not exceed a fraction of the total accessed points.

It should be noted that we do not discriminate whether the search is at the early stage or at the later stage and that the whole search process always checks if the above two schemes are conducted. The above analysis and our experiments show that the first scheme only take effects at the early stage and the second only at the later stage. The two parameters, the thresholds of the number of successive failing neighborhood expansions and the ratio between the number of points accessed in trees and total accessed points, in the above two schemes, are obtained by cross validation through selecting a small number of points as the validation queries to tune the two parameters. This cross validation scheme is feasible and the cost is very low as it only affects the search procedure without affecting the neighborhood graph construction.

Table 1: The comparison of neighborhood graph with partitioning trees and hashing for ANN search.

	search stage			construction stage	
	time overhead	search order	performance	storage cost	time cost
neighborhood graph	low	good	good	medium	high
partitioning trees	high	medium	medium	medium	medium
hashing	medium	poor	poor	low	low

4 Discussion

Comparison with partitioning trees and hashing Compared with partitioning trees, the neighborhood graph search has at least two advantages. On the one hand, the neighborhood structure provides a more efficient way to locate good NN candidates because the candidates can be quickly accessed from the neighborhood, while candidates with partitioning trees are accessed with the necessity of tree branching and tracing. On the other hand, our experiments show that the candidates from the neighborhood are usually better than those from partitioning trees and hence the neighborhood graph yields a better order to access the points. The two advantages have been illustrated in Figs. 1(a) and 1(b).

The comparison with hashing is summarized as follows. Hashing based approaches find NN candidates from the bucket that has the same hash code with the query. On the one hand, the bucket may be wrongly located. This drawback could be partially resolved by accessing multiple buckets with the similar hash codes (whose Hamming distance to the hash code of the query is smaller than a constant) or multiple hash coding, but it would increase the number of the data points that need to be accessed and hence leads to much more cost. On the other hand, the search procedure does not discriminate data points in the buckets and has to check each data point without ordering the accessing. This results in a waste of time on data points with low probability to be the true NNs. The paper [6] suggests to increase gradually the radius R in R-NN search to obtain the approximate NN, which is shown to be much slower than partitioning trees [19] .

Construction cost The construction itself is a little more costly compared with trees and hashing. There are some approximate methods, e.g., a divide-and-conquer algorithm [5]. However, for most applications, e.g., image search, the neighborhood graph could be built offline, and the construction can also benefit from parallel computing. In our implementation, we build the approximate neighborhood graph using kd-trees, followed by a postprocess to make sure that the resulted graph is connected. The cost is $O(n \log n)$ and acceptable as an offline process. Particularly, the proposed search approach can be applied to boosting the accuracy of the neighborhood graph by viewing each point as a query and accordingly updating its neighbors. Moreover, the proposed approach can be directly applied to the incremental neighborhood graph construction when inserting new points into the search database.

Storage cost The neighborhood graph is organized by attaching an adjacent list to each point, which requires additional storage to save adjacent lists including both the indices and distances. The total storage will be $O(n(k + 2l))$, with l the length of the adjacent list (i.e., the size of the neighborhood) and k the dimension of the data. In practice, our approach only requires a small neighborhood, and a neighborhood with 20 NNs (in all the experiments) leads to sufficiently good results. Therefore, the increased storage is much smaller than the storage of the data points, in high-dimensional cases (for example, GIST's dimension is 384.). In our experiment, the graph for indexing 1M SIFT features takes about 78MB storage while the features takes about 120MB, and for 1M GIST features they are about 78MB and 370MB. In contrast, vp-tree and spill-tree would cost $O(n(k+k/b))$ with b being the bucket size associated with the leaf nodes, and $2l$ in our case is usually smaller than k/b. A kd-tree and tp-tree (trinary projection tree [13]) may require relatively smaller storage because only the indexes of one or more splitting coordinates and the partitioning value are attached into each internal node, but the cost of randomized kd-trees for better search performance will increase. With the same storage cost, the ANN search performance of our approach is much better than those of randomized kd-trees and tp-trees. The comparison is summarized in Tab. 1.

5 Experiments

Data sets We demonstrate the proposed approach to ANN search over visual descriptors, SIFT features for patch matching, and GIST features for similar image search. The SIFT features are collected from the Caltech 101 data set [8] and recognition benchmark images [21], to construct the database. We extract maximally stable extremal regions (MSERs) for each image, and compute a 128-dimensional SIFT feature for each MSER. For each image set, we randomly sample 1000k SIFT features and 100k SIFT features, respectively as the search and query database, and additional 1K SIFT features as the validation data to determine the parameters in the search algorithm. We guarantee that the three data sets do not contain the same points.

Besides, we conduct the ANN search experiments on the tiny image set [29] to justify our approach for scalable similar image search. The tiny image data set consists of 80 million images and the sizes of all the images in this database are 32×32. Similar to [16], we use a global GIST descriptor to represent each image, which is a 384-dimensional vector describing the texture within localized grid cells. Its dimension is higher than that of the SIFT feature, and hence the ANN search is more challenging. We generate two data sets for similar image search, each including 1000k images as the search database and 100k images as the queries, additional 1k images for the validation database.

Evaluation scheme To evaluate the performance, we adopt the accuracy measurement to check whether the approximate nearest neighbors for each query is exactly the ground truth, the true nearest neighbors. In our experiments, we build the ground truth through the exhaustive linear scan. To evaluate the search performance for approximate k-nearest neighbors, the accuracy is computed by the ratio of the number of data points appearing both in ANN search results and the true nearest neighbors to the number of desired NNs.

We compare the search performances of widely-used ANN search algorithms, over partitioning trees, hashing and existing neighborhood graph search algorithms. Partitioning trees for comparison include tp-trees [13], bd-trees [2], vp-tree [34], spill-trees [17], and flann [19]. We report the result from flann [19] and do not report the results from kd-trees [26] and hierarchical k-means tree (clustering-tree) [21] because flann is to find the optimal configuration between multiple randomized kd-trees and hierarchical k-means tree and hence expects to be better as described in [19]. In addition, we also report the results from local neighborhood graph search with a single seed (AryaM93) [1], and query-independent (NG + independent) [24]. In our experiments, kd-trees are adopted as the partitioning trees for our approach. We run the implementations of spectral hashing [32], locality sensitive hashing (LSH) [6]. flann, and bd-tree, downloaded from the Web sites, with the parameters determined by their algorithms, to get the search results. We follow the algorithm descriptions in vp-trees, spill-trees and tp-trees, and report results by well implementing the algorithm and tuning the parameters, to get the best results. Consistent as the report in [19], the performance from hashing based approaches is much poorer than tree based methods with even an order of magnitude, and hence we do not plot the curves in the result for clear comparison of our approach with tree based methods. All the experiments are run on 2.66GHz desktop PC.

Results The ANN search performance comparison is shown in Fig. 3 over two data sets, similar patch search over SIFT features from the Caltech 101 data set [8] and recognition benchmark images [21]. The horizontal axis corresponds to the average query time (milliseconds), and the vertical axis corresponds to the search accuracy. We can have the following observations from the comparison. The proposed approach consistently gets the best performance. As shown in Figs. 3(a) and 3(b), at the accuracy of 0.9, it gets at least twice speedup compared with all other methods, and even costs only about one-fourth of the time costed by NG + independent over the visual descriptors of recognition benchmark images, as shown in Fig. 3(b). The superiority of our approach over other tree-based algorithms is consistent with the analysis in Sec. 4.

In addition, we present the comparison for similar image search over the tiny images [29]. The image is represented by a GIST feature, a higher-dimensional feature, which is more challenging. In this case, the superiority of our approach over other methods is even more significant. Figs. 4(a) and 4(c) show the performance comparison for approximate 1NN search, over two search databases sampled from tiny images and it can be observed that our approach is much better than partitioning trees and existing neighborhood graph search. Moreover, we also present the comparison about searching for top 20 similar images (i.e., approximate 20NNs), shown in Figs. 4(b) and 4(d), and the improvement is consistently significant.

Through a deep inspection of the comparisons, shown in Fig. 3 (on 128-dimensional descriptors)

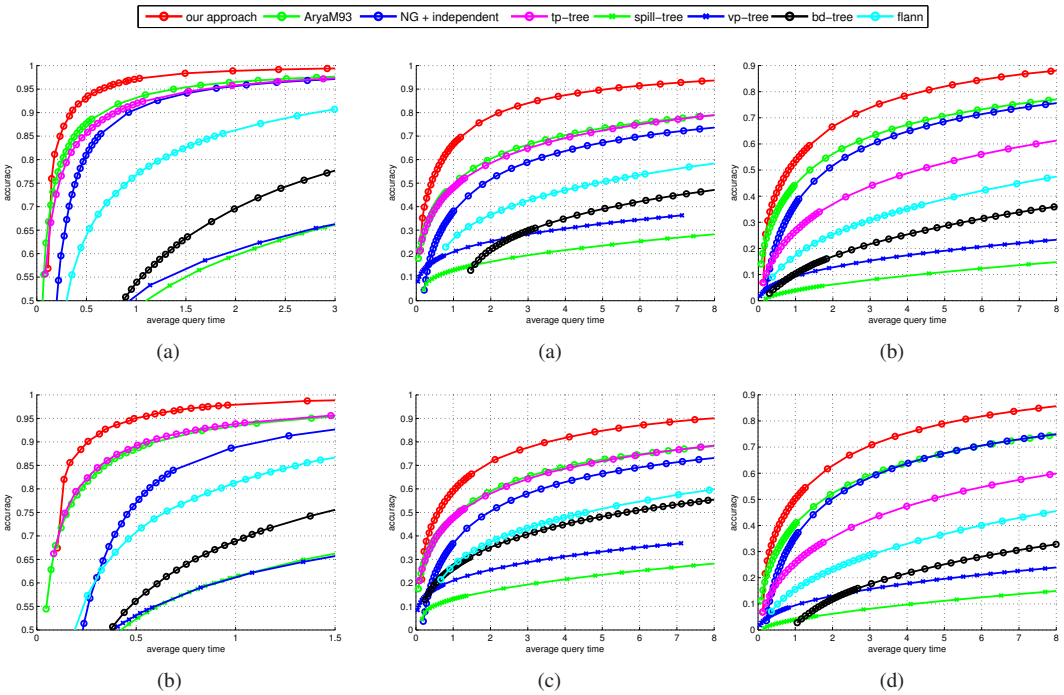


Figure 3: Performance comparison over (a) the Caltech 101 data set [8], (b) recognition benchmark images [21].

Figure 4: Performance comparison of searching for the most similar image in (a) and (c) and top 20 similar images in (b) and (d) over tiny images [29].

and Fig. 4 (on 384-dimensional descriptors), it can be observed that the advantage of our approach to ANN search is more significant for high dimensional cases and the superiority at the requirement of higher accuracy in high-dimensional cases is also more significant.

6 Conclusion

In this paper, we introduce neighborhood graph search for scalable visual descriptor indexing, and propose a query-driven iterated neighborhood graph search to deal with the drawback, the local characteristics in the original neighborhood graph search. We follow the iterated local search strategy and handle the key problem, how to perform perturbation to trigger a new local search. The key novelties lie in defining the local solution over neighborhood graph for ANN search and presenting a query and search history driven perturbation scheme to generate pivots to restart a new local search. Experimental results on large scale SIFT indexing and similar image search show that our approach gets significant improvement over representative ANN search algorithms.

References

[1] Sunil Arya and David M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, pages 271–280, 1993. 3, 5, 6, 8

[2] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998. 1, 2, 8

[3] Jeffrey S. Beis and David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *CVPR*, pages 1000–1006, 1997. 1, 2

[4] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. 1, 2

[5] Jie Chen, Haw ren Fang, and Yousef Saad. Fast approximate nn graph construction for high dimensional data via recursive lanczos bisection. *Journal of Machine Learning Research*, 10:1989–2012, 2009. 7

[6] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004. 1, 2, 7, 8

[7] Christos Faloutsos and King-Ip Lin. FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In *SIGMOD Conference*, pages 163–174, 1995. 3

[8] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *CVPR 2004 Workshop on Generative-Model Based Vision*, 2004. 8, 9

[9] Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974. 2

[10] Jerome H. Friedman, Jon Louis Bentley, and Raphael A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977. 1, 2

[11] Junfeng He, Wei Liu, and Shih-Fu Chang. Scalable similarity search with optimized kernel hashing. In *KDD*, pages 1129–1138, 2010. 1, 3

[12] Prateek Jain, Brian Kulis, and Kristen Grauman. Fast image search for learned metrics. In *CVPR*, 2008. 3

[13] You Jia, Jingdong Wang, Gang Zeng, Hongbin Zha, and Xian-Sheng Hua. Optimizing kd-trees for scalable visual descriptor indexing. In *CVPR*, pages 3392–3399, 2010. 1, 2, 7, 8

[14] William Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984. 3

[15] Brian Kulis and Trevor Darrells. Learning to hash with binary reconstructive embeddings. In *NIPS*, pages 577–584, 2009. 3

[16] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137, 2009. 1, 3, 8

[17] Ting Liu, Andrew W. Moore, Alexander G. Gray, and Ke Yang. An investigation of practical approximate nearest neighbor algorithms. In *NIPS*, 2004. 1, 2, 8

[18] Andrew W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *UAI*, pages 397–405, 2000. 1, 2

[19] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISSAPP (1)*, pages 331–340, 2009. 2, 3, 7, 8

[20] Gonzalo Navarro. Searching in metric spaces by spatial approximation. *VLDB J.*, 11(1):28–46, 2002. 3

[21] David Nistér and Henrik Stewénius. Scalable recognition with a vocabulary tree. In *CVPR (2)*, pages 2161–2168, 2006. 1, 2, 8, 9

[22] Maxim Raginsky and Svetlana Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*, pages 1509–1517, 2009. 3

[23] Hanan Samet. *Foundations of multidimensional and metric data structures*. Elsevier, Amsterdam, 2006. 2

[24] Thomas B. Sebastian and Benjamin B. Kimia. Metric-based shape retrieval in large databases. In *ICPR (3)*, pages 291–296, 2002. 3, 4, 5, 8

[25] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT press, 2006. 3

[26] Chanop Silpa-Anan and Richard Hartley. Optimised kd-trees for fast image descriptor matching. In *CVPR*, 2008. 1, 2, 8

[27] Josef Sivic and Andrew Zisserman. Efficient visual search of videos cast as text retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(4):591–606, 2009. 1

[28] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3D. *ACM Trans. Graph.*, 25(3):835–846, 2006. 1

[29] Antonio B. Torralba, Robert Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):1958–1970, 2008. 8, 9

[30] Godfried T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980. 3

[31] Jun Wang, Ondrej Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, pages 3424–3431, 2010. 3

[32] Yair Weiss, Antonio B. Torralba, and Robert Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008. 1, 3, 8

[33] K. Yamaguchi, T. L. Kunii, and K. Fujimura. Octree-related data structures and algorithms. *IEEE Computer Graphics and Applications*, 4(1):53–59, 1984. 2

[34] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, pages 311–321, 1993. 1, 2, 8