Query-Driven Iterated Neighborhood Graph Search for Large Scale Indexing

Jingdong Wang Shipeng Li
Microsoft Research Asia, Beijing, P. R. China
{jingdw, spli}@microsoft.com

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 search, which constructs a neighborhood graph to index the data points and conducts a local search, expanding neighborhoods with a best-first manner, for ANN search. Our empirical analysis shows that neighborhood expansion is very efficient, with $O(1)$ cost, for a new NN candidate location, and has high chances to locate true NNs and hence it usually performs well. However, it often gets sub-optimal solutions since local search only checks the neighborhood of the current solution, or conducts exhaustive and continuous neighborhood expansions to find better solutions, which deteriorates the query efficiency.

In this paper, we propose a query-driven iterated neighborhood graph search approach to improve the performance. We follow the iterated local search (ILS) strategy, widely-used in combinatorial optimization, to find a solution beyond a local optimum. We handle the key challenge in making neighborhood graph search adapt to ILS, Perturbation, which generates a new pivot to restart a local search. To this end, we present a criterion to check if the local search over a neighborhood graph arrives at the local solution. Moreover, we exploit the query and search history to design the perturbation scheme, resulting in a more effective search. The major benefit is avoiding unnecessary neighborhood expansions and hence more efficiently finding true NNs. Experimental results on large scale SIFT matching, similar image search, and shape retrieval with non-metric distance measures, show that our approach performs much better than previous state-of-the-art ANN search approaches.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*; H.3.1 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*

MM'12, October 29–November 2, 2012, Nara, Japan.

General Terms

Algorithms, Experimentation

Keywords

Local neighborhood graph search, iterated, query-driven, ANN search

1. INTRODUCTION

Large scale indexing has been attracting more and more interest in multimedia search. Efficient and effective approximate nearest neighbor search plays an important role in this field. For instance, the state-of-the-art duplicate image search [49, 50] depends on the bag-of-words descriptor, which usually exploits ANN search techniques to group large scale local features into visual words using k-means or similar algorithms [36] and map visual features to visual words. Other examples include finding the best matches for local image features in large data sets [37] and large scale similar image search [22].

In the communities of multimedia and computer vision, two main categories of ANN search schemes, partitioning trees and hashing, are studies. Partitioning trees, including kd-trees [6, 13] and its variants [4, 5, 19, 35], metric trees (e.g., ball trees [24], vantage point trees (vp-tree) [48], spill-trees [23]), and hierarchical k-means trees [29], 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 obtain NN candidates, in some order, e.g., depth-first or best-first. Such a scheme takes too much *time overhead*, which is defined as the time cost of discovering new NN candidates in this paper. In partitioning trees, the time overhead is the cost of locating a leaf node. Moreover, the *search order*, which means the order of accessing NN candidates, is not good enough to make the searching path quickly move towards the true NNs. Note that time overhead and search order are the key factors determining the ANN search performance.

Hashing based approaches include locality sensitive hashing (LSH) [8], spectral hashing [45], and other variants [16, 22, 46]. 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 leads to high time cost on checking the

points with low probabilities to be true NNs. The paper [8] suggests a sub-optimal search order, increasing gradually the radius R in R-NN search, to access NN candidates, which is shown to be much slower than partitioning trees [27].

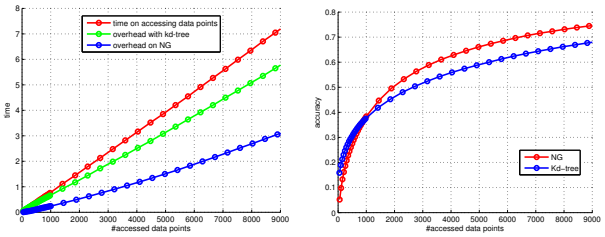
In this paper, we propose to exploit the neighborhood graph structure for large scale visual descriptor indexing. The intuition of its workability is that the points, near a point that is close to the query, have high probabilities to be also close to the query. The search procedure, early presented in [3] and reproposed in [15], named as a local neighborhood graph search, starts from one or several data points and conducts a best-first strategy to discover new NN candidates, by first checking the points in the neighborhood of the best one among the previously-discovered NN candidates 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 search order to access the candidates determined from the neighborhoods is better than partitioning trees and hashing based methods. Fig. 1 presents the comparisons over $1M$ GIST descriptors of tiny images between the neighborhood graph and partitioning trees to demonstrate such advantages. It can be observed (1) that the time overhead with the NG search is only a half of that with kd-trees and is about $\frac{2}{5}$ of the time cost of computing the distances between the query and the accessed points when accessing 6000 points and (2) that the accuracy of the NG search is larger than that of kd-trees. However, the local neighborhood graph search often gets sub-optimal solutions, or conducts exhaustive and continuous neighborhood expansions to find better solutions, which deteriorates the query efficiency.

We propose a query-driven iterated neighborhood graph search approach for searching ANNs beyond local optima. In order to make the local neighborhood graph search adapt to the widely-used iterated local search strategy in operation research, we resolve the key challenge, i.e., designing Perturbation to generate a new pivot to restart a local search, by introducing a criterion to check if the ANN search over a neighborhood graph reaches a local optimum. Moreover, we propose to utilize the query and search history to drive the perturbation scheme to make the search more effective. The major benefit from them is avoiding unnecessary neighborhood expansions and more efficiently finding true NNs. Experimental results of ANN search for visual descriptors (e.g., 128-dimensional SIFT descriptors) demonstrate that our approach gets the superior performance. Particularly, when searching higher-dimensional visual descriptors (e..g, 384-dimensional GIST 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 multimedia search, 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 short review on ANN search methods that are widely investigated in multimedia. More could be found from [31].

Partitioning trees. The partitioning tree based approaches recursively split the space into subspaces, and organize the subspaces using a tree (called space partitioning tree). Most space-partitioning systems use hyperplanes or



(a) (b)

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

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 [6, 13] and its variants [4, 5, 19, 35], and metric trees (e.g., ball trees [24], vantage point trees (vp-tree) [48], and spill-trees [23]). Using other criteria to split the space may yield multi-way partitioning trees, such as Quadtrees [11], Octrees [47], hierarchical k-means trees [29] and so on, which are mainly for low-dimensional cases and hence improper for ANN search over high-dimensional visual descriptors.

In the query stage, the branch-and-bound methodology [6] 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-identified nearest neighbors. The current state-of-the-art search strategy, priority search [4] or best-first [5], builds a priority queue to produce an order of accessing subtrees so that the data points with large probabilities to be true nearest neighbors are first accessed.

Recently, kd-tree based ANN search methods are widely investigated for multimedia search applications. The scheme about multiple randomized kd-trees is studied in [35]. 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 [19].

Hashing. Hashing based ANN search methods also attract a lot of attention. Locality sensitive hashing (LSH) [8], 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 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 nearest neighbors by hashing the query and retrieving elements stored in the buckets containing it. LSH has many applications, e.g., for pose matching, contour matching, and mean shift clustering, a literature review could be found in [33]. Recently, a lot of research efforts have been conducted on finding good hashing functions, by using metric-learning-like techniques, such as optimized kernel hashing [16], learned metrics [18], learnt binary reconstruction [21], kernelized LSH [22], and shift

Algorithm 1 Iterated local search

1. $s_0 \leftarrow$ GenerateInitialSolution
2. $s^* \leftarrow$ LocalSearch(s_0)
3. **repeat**
4. $s' \leftarrow$ Perturbation(s^*, *history*) /*Produce a new configuration from the current optimum s^* and the history.*/
5. $s^{*\prime} \leftarrow$ LocalSearch(s')
6. $s^* \leftarrow$ AcceptanceCriterion(s^*, $s^{*\prime}$, *history*) /*Determine if the new solution is accepted.*/
7. **until** termination condition met

kernel hashing [30], semi-supervised hashing [42], weakly-supervised hashing [25], non-metric hashing [26], multi-feature hashing [38], spectral hashing [45], complementary hashing [46]. The learning based strategy actually can also be used in our approach. However, this paper would not make investigation on index construction, and instead focuses mainly on the search strategy. Hashing based algorithms may be good to guarantee good precision and recall performances, but is poor at the query efficiency [27].

There are some other methods for ANN search. LSH essentially is also an embedding method, and other classes of embedding methods include Lipschitz embedding [20] and FastMap [9]. 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 [28], relative neighborhood graph [40], k-NN (ϵ-NN) graph [32], and degree-reduced neighborhood graph [1]. The focus of this paper does not lie in the neighborhood graph construction, and but focuses on the search procedure. 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 [32] (query independent) or from the first NN candidate of kd-trees [3] (query dependent). These approaches, however, are local search and tend to be stuck at local optima. This paper proposes a query-driven iterated neighborhood graph search approach to address such a problem. As an ongoing work, we are investigating the ANN search over billions of points using a two-stage inverted indexing approach, by combining the proposed approach and the approach presented in [41] and later we will show a demo based on the technology [43] in ACMMM12.

3. OUR APPROACH

Given a set of n reference data points $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathbb{R}^k$ 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 that organizes 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.

3.1 Preliminaries

The following introduces the preliminary techniques local neighborhood graph search [3] that performs ANN search over a neighborhood graph and iterated local search [17] that finds solutions beyond local optima.

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. $P' \leftarrow$ Perturbation(R^*, q, T, *history*)
5. $R^{*\prime} \leftarrow$ LocalNGSearch(P', G, *history*)
6. $R^* \leftarrow$ AcceptanceCriterion(R^*, $R^{*\prime}$)
7. **until** termination condition met

Local neighborhood graph search. Local neighborhood graph search for ANNs is a procedure that starts from a set of seeding points as initial NN candidates and propagates the search by continuously accessing their neighbors from previously-discovered NN candidates to discover more NN candidates. The *best-first* strategy [3] is usually adopted for local neighborhood expansion. To this end, a *priority queue* is used to maintain the previously-discovered NN candidates whose neighborhoods are not expanded yet, and initially contains only seeds. The best candidate in the priority queue is extracted out, and the points in its neighborhood are discovered as new NN candidates and then pushed into the priority queue. The resulting search path, discovering NN candidates, may not be monotone, but always attempts to move closer to the query point without repeating points. As a local search that finds better solutions only from the neighborhood of the current solution, the local neighborhood graph search will be stuck at a locally optimal point and has to conduct exhaustive neighborhood expansions to find better solutions. Instead, we present a query-driven iterated neighborhood graph search to efficiently find solutions beyond local optima.

Iterated local search. Various modifications have been done over local search to kick a solution out from a local optimum. 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 [17] is based on building a sequence of locally optimal solutions by perturbing the current local optimum and applying local search after starting from the modified solution. The standard procedure is outlined in Algorithm 1.

3.2 Query-driven iterated search

Our approach improves the local neighborhood graph search from two aspects: iterated search and query-driven. The basic procedure is outlined in Algorithm 2.

- GenerateInitialSolution(q, T) searches over trees T, which are constructed to index the reference data points. The initial solution contains a small amount of initial NN candidates that might not be good enough but have high probabilities to be near true NNs. The trees are kd-trees in our implementation, and the search over trees is the same to the best-first search strategy [19, 27, 35].

- LocalNGSearch(P_0, G) is similar to the local neighborhood graph (NG) search mentioned before, starting from a set of seeds P_0 and searching over G by conducting neighborhood expansions in a best-first manner. The difference lies in that once reaching a locale

solution, the NG search will pause and return discovered NNs. We will describe the criterion to inspect the local optimum for our local NG search.

- Perturbation(R^*, q, T, *history*) generates new seeds from trees T according to the search history and previously-identified NNs (R^*), to avoid accessing too many points that are accessed in both searches over trees and the neighborhood graph. The detail is given later.

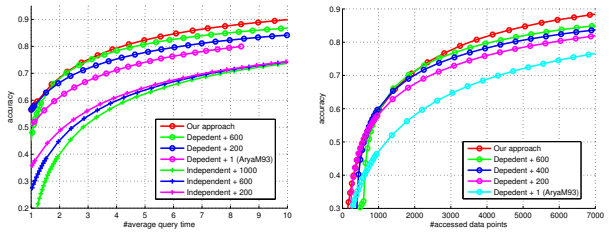
- LocalNGSearch(P', G, *history*) is slightly different from LocalNGSearch(P_0, G) as the search history, i.e., the NNs discovered up to the current iteration, is considered in neighborhood expansion. During the searches over both trees and the neighborhood graph, a single priority queue is maintained to store NN candidates.

Local optimum inspection. Considering 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 $|\mathbf{x} - \mathbf{x}^*| < \epsilon$. We propose an analogical definition of the local optimum (minimum) for local neighborhood graph search.

Definition 1 (Local optimum for local NG search). *The distance function $f(v_i; \mathbf{x}_q)$ over the neighborhood 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. In the implementation, we use an indication variable to record the number of unexpanded promising points among all the previously-checked points. The indication variable reaching zero means that the local neighborhood graph arrives at local solutions.

Perturbation. We generate a small amount of new seeding points according to the query and the search history, called query and history driven perturbation. We still use trees, but perform the search that is slightly different from the conventional search over trees [19, 27, 35] by taking into consideration the search history and using the previously-identified NNs from both trees and neighborhood graph to determine if a subtree should be searched. Seeds generation in perturbation will be conducted by resuming the search in trees that yields the initial solutions. But if using the same procedure with the conventional scheme [19, 27, 35], it will result in that many leaf nodes have already been accessed in the neighborhood graph search, and hence extra much time overhead, locating the leaf nodes in trees, is taken. To reduce the time overhead, but still get useful 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 that is determined by cross validation in our implementation. Here, the reason of discovering a single point from a compact tree is that finding the best point through searching the compact tree is more costly than finding the same point by searching the neighborhood graph guided by a point near the best one, e.g., the one from the compact tree here. During the period of building the trees, an extra attribute is associated with each node of the trees, indicating the average similarity of the points within the corresponding subtree and helping determine if a subtree is a compact tree.



(a) (b)

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

Termination. The iteration 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 be interpreted. This is equivalent to stopping the search when the search time reaches a predefined threshold. The latter is challenging for the local neighborhood graph search. In [32], 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 to find true NNs, and the experiments show that this stop condition is not reliable. In contrast, the proposed iterated search scheme has a stop condition to guarantee that the true nearest neighbors are found. The condition is similar to the search for exact NNs in trees, and it is that the best lower bound in the priority queue maintained for the search in trees is larger than the worst distance in the previously-identified (K-)NNs.

4. ANALYSIS AND DISCUSSIONS

4.1 Analysis

Empirical analysis. We present the empirical comparison of different search schemes over a neighborhood graph. The schemes include a representative query-independent search scheme [32] that clusters reference data points and regards the cluster centers as the seeding points, a simple query-dependent manner that uses the first NN candidate discovered from kd-trees as the seeding point [3], another query-dependent manner that modifies the scheme [3] and discovers a small amount of NN candidates as the seeding points, and the proposed query driven iterated search scheme.

Fig. 2(a) presents the comparison in terms of average query time vs. search accuracy. "Independent + n" means the query-independent manner clustering the reference data points into n (200, 600, and 1000) groups [32]. "Dependent

+ n" means the query-dependent manner finding the first n (1 [3], 200, 600) NN candidates from kd-trees as the seeding points. The comparison shows that query-dependent manners perform better and our approach performs the best. Fig. 2(b) presents the search accuracy when accessing the same number of NN candidates. It shows that our approach accesses a smaller number of NN candidates but achieves the same accuracy.

Theoretical analysis. Our approach is based on a deterministic neighborhood graph, and it is uneasy to give theoretic analysis. We follow [3] to discuss the property over a random neighborhood graph. It has been proved [3] that the query cost is upper bounded.

THEOREM 1 (FROM [3]). *Given any n point set $\mathcal{S} \in \mathbb{R}^k$ 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.*

Instead of the expected time, we are more interested in the time cost for the good and bad cases.

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 $O(\log^3 n)$.*

This property can be proved using the similar arguments in the proof of Theorem 1 [3]. Suppose the former and latter 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 to analyze more details.) 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 solutions (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.

THEOREM 2. *The expected time cost of the search procedure with P considered 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 a query-driven non-iterative one, 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 show 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.

4.2 Discussion

The summary of comparison between neighborhood graph search and partitioning trees, hashing is presented in Tab. 1.

Neighborhood graph 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 [7]. 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 the scheme presented in [44]. 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 approximate neighborhood graph if more time is available. Moreover, the proposed approach can be directly applied to the incremental neighborhood graph construction when inserting new points into the reference database.

Neighborhood graph 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 $120M$B, and for 1M GIST features they are about $78M$B and $370M$B. 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 [19]) 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.

Advantages over partitioning trees. 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).

Advantages over hashing. 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 probabilities to be true NNs. The paper [8] suggests to increase gradually the radius R in R-NN search to obtain the approximate NN, which is shown in [27] to be much slower than partitioning trees.

4.3 Implementation details

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	low	poor	poor	low	low

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 $d(p^*,q) < |d(u,q)-d(u,v)|$ 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 (condition 1).

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 time 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 (condition 2). It should be noted that our algorithm does not discriminate whether the search is at the early stage or at the later stage and that the whole search process always checks the two conditions described above

The two parameters, the thresholds of the number of successive failure 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.

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 [10] and recognition benchmark images [29]. 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 reference 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 [39] 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 [22], 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 reference 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, with partitioning trees, hashing and existing neighborhood graph search algorithms. Partitioning trees for comparison include tp-trees [19], bd-trees [4], vp-tree [48], spill-trees [23], and flann [27]. We report the result from flann [27] and do not report the results from kd-trees [35] and hierarchical k-means tree (clustering-tree) [29] because flann aims to find the optimal configuration between multiple randomized kd-trees and hierarchical k-means tree and hence expects to be better as described in [27]. In addition, we also report the results from local neighborhood graph search with a single seed (AryaM93) [3], and query-independent ("NG + independent") [32]. In our experiments, kd-trees are adopted as the partitioning trees for our approach. We run the implementations of spectral hashing [45], locality sensitive hashing (LSH) [8], 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 algorithms and tuning the parameters, to get the best results. Consistent with the report in [27], 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 other methods. All the experiments are run on a 2.66GHz desktop PC.

5.1 Comparisons

Quantitative results. The ANN search performance comparison is shown in Fig. 3 on two data sets, similar patch search over SIFT features from the Caltech 101 data set [10] and recognition benchmark images [29]. 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. 4.2.

In addition, we present the comparison for similar image search over the tiny images [39]. 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 1-NN search, over two reference databases sampled from tiny images. It can be observed that our approach is much better than partitioning trees and existing local neighborhood graph search methods. Moreover, we present the comparison about searching for top 20 similar images (i.e., approximate 20-NNs), 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) 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.

Qualitative results. We present the qualitative comparisons with representative methods over similar image search, shown in Fig. 5 and Fig. 6, where (a), (b), (c), and (d) show top 10 similar images of our approach, AryaM93 [3], tp-tree [19], and "NG + independent" [32]. In each row, the first image with a blue bounding box is the query, and the ones bounded by red boxes are truly similar images. We can see that the results from our approach are much better than those from other approaches.

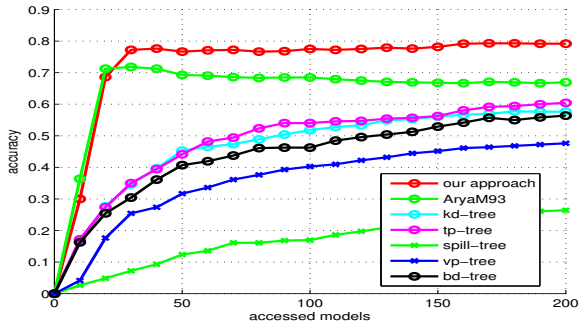


Figure 7: Shape search performance. The horizontal axis corresponds to the number of accessed models, and the vertical axis corresponds to the accuracy.

5.2 Application with non-metric distances

We present a 3D model retrieval experiment to show the unique advantage of neighborhood search that it works well for the non-metric distance. The input of the 3D model retrieval task is a 2D sketch or a 2D projection, and the goal is to search for 3D models that belong to the same category of the input. To conduct such a task, we organize 3D shapes using their 2D contours. The 2D contours are obtained by rendering perspective 2D contours of each model from 9 views. For each model, we randomly sample one as a query and the remaining eight for the reference database. The 2D contours are further vectorized by fitting line segments to the points with robust moving least-squares [12] and represented by a set of points with 2D coordinates. To match two 2D contours, we estimate an affine transformation between them using RANSAC and use the registration error as the distance, which is described in [34]. In our experiment, we collect all the models in the Princeton Shape Benchmark[1], which contains 1815 models, and then expand the database with models of the same categories from the INRIA GAMMA 3D mesh research database[2], yielding 5000 models, and the storage cost of contour images is about 1.5GB.

We index these 2D contours by connecting each 2D contour with its 20 nearest neighbors using the distance described above to form a matching graph. Particularly we also record the affine transform for each edge so that the matching over the graph from a node v_1 to another one v_2 can be speeded up by combining the recorded transform and the transform from the query to v_1 to get an initial transform estimation to v_2. It takes about four hours to compute the matching graph on a Dell desktop with dual quad-core Intel Xeon E5540 processors, and about 30MB to store. We also represent the 2D contour by a vector, a boundary-descriptor [14]. The vector representations are used to organize the 2D contours by kd-trees and other related index structures.

To compare the search performance, for the proposed approach and local NG search, once a 2D contour is accessed the distance between the associated model and the query is evaluated. For other approaches, we search over the index structures to get a set of candidates and then perform a reranking step by matching the query and the models corresponding to the searched 2D contours. Due to the stored transforms over our matching graph, our approach and local NG search are much faster (over 5 times) than other approaches. So we only report the comparison in terms of the accuracy vs. the number of accessed models, shown in Fig. 7. The accuracy is computed by checking if the searched models and the query belong to the same category. As we can see, our approach performs much better than other approaches. This is because the search with a neighborhood graph is more effectively towards true nearest neighbors and our it-

[1]http://shape. cs.princeton.edu/benchmark/
[2]http://www-roc.inria.fr/gamma/ download/

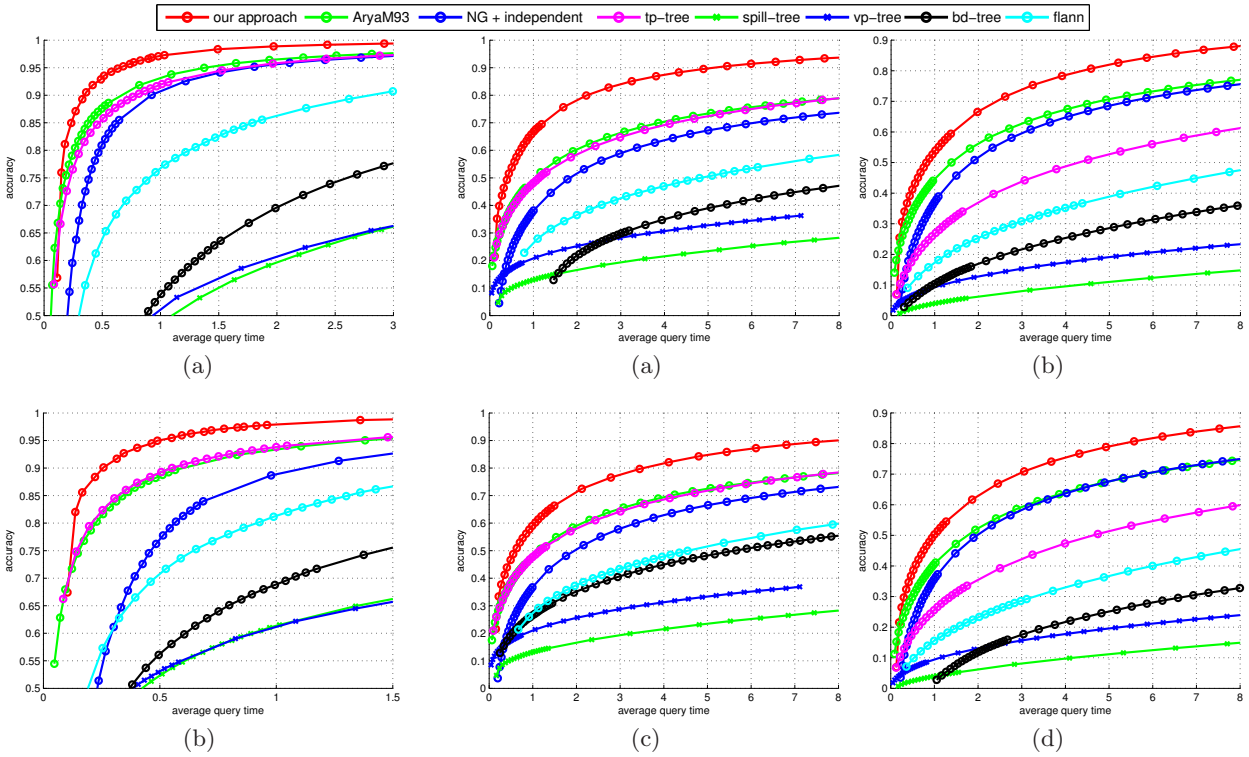


Figure 3: Performance comparison over (a) the Caltech 101 data set [10], (b) recognition benchmark images [29].

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 the tiny images [39].

eration scheme avoids the local issue that is a problem for the local NG search (AryaM93) [2].

6. CONCLUSIONS

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 that the local neighborhood graph search tends to be stuck at a locally optimal point. We handle the key problem, how to perform perturbation to trigger a new local search. The major technologies lie in defining the local solution over a 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 with metric and non-metric distance show that our approach gets significant improvement over other ANN search algorithms.

7. REFERENCES

[1] K. Aoyama, K. Saito, H. Sawada, and N. Ueda. Fast approximate similarity search based on degree-reduced neighborhood graphs. In *KDD*, pages 1055–1063, 2011.

[2] S. Arya and D. M. Mount. Algorithms for fast vector quantizaton. In *Data Compression Conference*, pages 381–390, 1993.

[3] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, pages 271–280, 1993.

[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[5] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *CVPR*, pages 1000–1006, 1997.

[6] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

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

[8] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004.

[9] C. Faloutsos and K.-I. Lin. FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In *SIGMOD Conference*, pages 163–174, 1995.

[10] L. Fei-Fei, R. Fergus, and P. 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.

[11] R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.

[12] S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph.*, 24(3):544–552, 2005.

[13] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[14] T. A. Funkhouser, P. Min, M. M. Kazhdan, J. Chen, J. A. Halderman, D. P. Dobkin, and D. P. Jacobs. A search engine for 3d models. *ACM Trans. Graph.*, 22(1):83–105, 2003.

[15] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *IJCAI*, pages 1312–1317, 2011.

[16] J. He, W. Liu, and S.-F. Chang. Scalable similarity search with optimized kernel hashing. In *KDD*, pages 1129–1138, 2010.

[17] H. H. Hoos and T. Stützle. *Stochastic Local Search Foundations and Applications*. Morgan Kaufmann/Elsevier, 2004.

[18] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *CVPR*, 2008.

[19] Y. Jia, J. Wang, G. Zeng, H. Zha, and X.-S. Hua. Optimizing kd-trees for scalable visual descriptor indexing. In *CVPR*, pages 3392–3399, 2010.

[20] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[21] B. Kulis and T. Darrells. Learning to hash with binary reconstructive embeddings. In *NIPS*, pages 577–584, 2009.

[22] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137, 2009.

[23] T. Liu, A. W. Moore, A. G. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *NIPS*, 2004.

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

[25] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *CVPR*, pages 3344–3351, 2010.

[26] Y. Mu and S. Yan. Non-metric locality-sensitive hashing. In *AAAI*, 2010.

[27] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISSAPP (1)*, pages 331–340, 2009.

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

[29] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *CVPR (2)*, pages 2161–2168, 2006.

[30] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*, pages 1509–1517, 2009.

[31] H. Samet. *Foundations of multidimensional and metric data structures*. Elsevier, Amsterdam, 2006.

[32] T. B. Sebastian and B. B. Kimia. Metric-based shape retrieval in large databases. In *ICPR (3)*, pages 291–296, 2002.

[33] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT press, 2006.

[34] T. Shao, W. Xu, K. Yin, J. Wang, K. Zhou, and B. Guo. Discriminative sketch-based 3d model retrieval via robust shape matching. *Comput. Graph. Forum*, 30(7):2011–2020, 2011.

[35] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *CVPR*, 2008.

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

[37] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. *ACM Trans. Graph.*, 25(3):835–846, 2006.

[38] J. Song, Y. Yang, Z. Huang, H. T. Shen, and R. Hong. Multiple feature hashing for real-time large scale near-duplicate video retrieval. In *ACM Multimedia*, pages 423–432, 2011.

[39] A. B. Torralba, R. Fergus, and W. 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.

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

[41] W. Tu, R. Pan, and J. Wang. Similar image search with a tiny bag-of-delegates representation. In *ACM Multimedia*, 2012.

[42] J. Wang, O. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, pages 3424–3431, 2010.

[43] J. Wang, J. Wang, X.-S. Hua, and S. Li. Scalable similar image search by joint indices. In *ACM Multimedia*, 2012.

[44] J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan, and S. Li. Scalable k-nn graph construction for visual descriptors. In *CVPR*, pages 1106–1113, 2012.

[45] Y. Weiss, A. B. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[46] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu. Complementary hashing for approximate nearest neighbor search. In *ICCV*, pages 1631–1638, 2011.

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

[48] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, pages 311–321, 1993.

[49] S. Zhang, Q. Tian, G. Hua, Q. Huang, and S. Li. Descriptive visual words and visual phrases for image applications. In *ACM Multimedia*, pages 75–84, 2009.

[50] W. Zhou, Y. Lu, H. Li, Y. Song, and Q. Tian. Spatial coding for large scale partial-duplicate web image search. In *ACM Multimedia*, pages 511–520, 2010.

Appendix

The pseudo-code of our approach is outlined as follows.

Algorithm 3 IteratedLocalNGSearch(q, G, T)

// R: a max-heap with a fixed size; Q: a priority queue;
// $V[G]$: the node set of graph G; n: #accessed points;
// m: #promosing points; r: #accessed points from trees T;

for each $u \in V[G]$ **do**
  $color[u] \leftarrow$ WHITE;
end for
$n \leftarrow 0$;
$m \leftarrow 0$; $r \leftarrow 0$;
$P \leftarrow$ GenerateInitialSolution(q, T)
for each $u \in P$ **do**
  $key[u] \leftarrow dist(u, q)$; $color[u] \leftarrow$ BLACK;
  $Q \leftarrow u$;
  if MAX(R) $> key[u]$ **then**
    $R \leftarrow u$;
  end if
  $n \leftarrow n + 1$;
  $m \leftarrow m + 1$; $r \leftarrow r + 1$;
end for
$t \leftarrow 0$;
while ($Q \neq \emptyset$ && $n < N$) **do**
  $u \leftarrow$ EXTRACT-MIN(Q);
  /*Start the local neighborhood graph search.*/
  $m \leftarrow m - 1$; $b \leftarrow$ FALSE;
  for each $v \in Adj[u]$ **do**
    if $color[v] \neq$ BLACK **then**
      $color[v] \leftarrow$ BLACK; $key[v] \leftarrow dist(v, q)$;
      $Q \leftarrow v$;
      if MAX(R) $> key[v]$ **then**
        $R \leftarrow v$;
      end if
      $n \leftarrow n + 1$;
      if IsPromising(v) **then**
        $m \leftarrow m + 1$; $b \leftarrow$ TRUE;
      end if
    end if
  end for
  if $b ==$ FALSE **then**
    $t \leftarrow t + 1$;
  else
    $t \leftarrow 0$;
  end if
  /*$m <= 0$ means arriving at a local optimum. $t >= F$ and $r/n < L$ mean the conditions in the two schemes described in Sec. 4.3*/
  if $(m <= 0 \| t >= F) \&\& (r/n < L)$ **then**
    $P \leftarrow$ Perturbation(q, T, R);
    for each $u \in P$ **do**
      $key[u] \leftarrow dist(u, q)$; $color[u] \leftarrow$ BLACK;
      $Q \leftarrow u$;
      if MAX(R) $> key[u]$ **then**
        $R \leftarrow u$;
      end if
      $n \leftarrow n + 1$; $m \leftarrow m + 1$; $r \leftarrow r + 1$;
    end for
    if $t >= F$ **then**
      $t \leftarrow 0$;
    end if
  end if
end while
return R;

Figure 5: Qualitative comparisons. (a), (b), (c), and (d) show top 10 similar images of our approach, AryaM93 [3], tp-tree [19], and "NG + independent" [32]. In each row, the first image with a blue bounding box is the query, and the ones bounded by red boxes are truly similar images.

Figure 6: Qualitative comparisons. (a), (b), (c), and (d) show top 10 similar images of our approach, AryaM93 [3], tp-tree [19], and "NG + independent" [32]. In each row, the first image with a blue bounding box is the query, and the ones bounded by red boxes are truly similar images.