DAOCT: Dynamic Agglomerative Overlapping Clustering Tree for Fast Binary Descriptors Retrieval

Anonymous CVPR submission

Paper ID 1234

Abstract

Binary descriptor has the potential to be a very efficient form of visual feature for nearest neighbors search. However, hindered by high dimensionality and the discreteness of Hamming distance, fast and efficient approximate nearest neighbor (ANN) search on binary descriptors is still challenging. We discover that the border effect of vector quantization in Hamming space is a key factor affects the ANN searching performance, and we propose the Dynamic Agglomerative Overlapping Clustering Tree (DAOCT) algorithm to further exploit the potential of binary descriptors' ANN searching. For the first time, we try to apply a dynamic space partition method to an ever changing database, which is a common case in practical application. The main contributions lies in ***i.*** *adapting a built-online dynamic agglomerative clustering tree algorithm for binary descriptors retrieval, and avoid the high building cost;* ***ii.*** *proposing an overlapping strategy to deal with the border effect in vector quantization methods, which boosts the ANN search performance in a more efficient way than the widely used strategy of building multiple data structures;* ***iii.*** *selection of clustering centroids by eliminating unsuitable ones. Our experiments manifest DAOCT outperforms the state of the art binary descriptors' ANN searching methods on static databases.*

1. Introduction

Image descriptors are of great utility in tasks like image retrieval, image mosaic and geographic reconstruction. Various published feature extraction methods such as SIFT, SURF and GLOH find and project interest points to high dimensional vectors [9, 3, 10, 1]. Binary descriptors have recently attracted considerable research efforts for their merits on large scale storage and computation [8, 4]. Given same length, binary descriptors are several times more efficient to store than float descriptors and they require much less machine instructions to compare under modern computer architecture. Nearest neighbor (NN) search with exhaustive linear scanning is rather computational expensive for large scale databases. To tackle this issue, a considerable number of approximate nearest neighbor (ANN) search methods such as K-d partition tree have been proposed for large scale float descriptors retrieval. High retrieval accuracy can be acquired in a comparably short time.

However, most of these ANN search methods for real value descriptors can not apply to binary descriptors directly. Because Hamming distance is discrete and it is the sum of bitwise difference on each dimension, the number of all possible distance values equals to the dimension of Hamming space, it tells less distance information than Euclidean distance given same dimension. Since there is one bit for one dimension in Hamming space, it is very unlikely to apply conventional dimensionality reduction methods or to make partition on each dimension for binary descriptors. Previous works show Locality sensitive hashing (LSH) [2] and hierarchical clustering tree (HCT) [12] can achieve favourable ANN searching performance in Hamming space, but they are not efficient enough. High dimensionality is always a curse and locality sensitive hashing requires very large memory.

In the building process of divisive space partition methods such as HCT, the algorithm sometimes cannot cluster data points to its closest center. This is caused by the divisive building process which only allows data point finds its closest center in the cluster fixed by higher layer clustering. Agglomerative methods start from bottom to the top, the vectors in each layer find their closest centroids freely, without the predeterminate boundaries from upper cluster. Though agglomerative approaches acquire better clustering quality in this way, the price is its high building complexity.

In this paper, we introduce a new fast ANN searching method for binary descriptors based on agglomerative space partition. We adopt a dynamic building strategy (DBS) to waive agglomerative methods' high building costs.

In modern internet applications and practical computer vision projects, the databases are not always static. Those databases may grow from small to large as business expands

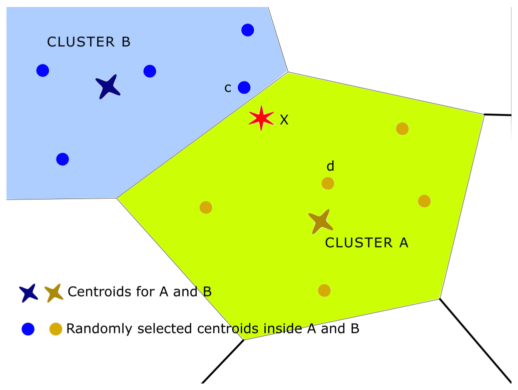

Figure 1. An illustration of border effect. If the small red star X represents a query vector, before reaching the closest centroid c in Cluster B, the priority search method exams a lot of worse candidates in Cluster A.

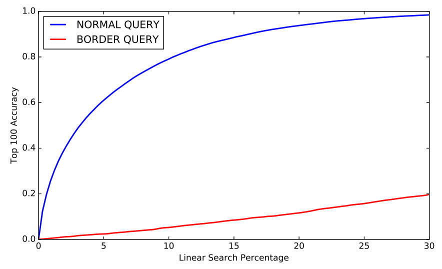

Figure 2. The border effect on nearest neighbors searching. Using one HCT as an example. We deliberately choose some binary vectors near the cluster borders as the query set based on following rules. i. this vector is in the middle point of two clustering centroids; ii. the distance to thses two centroids is shorter than the distance to any other peer centroid. This figure shows the top 100 nearest neighbors searching performance of normal query vectors and border query vectors.

and be subject to edit. The DBS also allows the implementation of ANN search on these datasets. To put it simply, we mingle the querying, building and updating process in the dynamic phase queries of the proposed DAOCT, and these ANN queries still maintain a favourable performance. The data structure will grow as the database expands and automatically update the oldest part of it to fit the changes in database. As for the implementation on static datasets, we also adapt an acceleration strategy to keep its high ANN searching performance.

For space partition approaches, when the query point is close to the border of a high level cluster, the algorithm may need to backtrack a lot to get the nearest point, as illustrated in figure 1. Because Hamming distance is discrete and only has limited number of distance values, this situation is more likely to happen in Hamming space. In order to minimize the *border effect* mentioned above, many vector quantization approaches introduce the strategy of building multiple data structures for one dataset. This is based on the notion that it is less likely of a query vector to simultaneously land on the border area in different partitions. We find that adapting a partial overlap strategy can also tackle the *border effect*, and avoids the building and storing of multiple data structures.

In terms of centroids selection, some currently available methods such as Gonzalez centroids selection is too expensive to implement on large database, and some methods like K-means are only applicable to real value space. In this paper we come up with an elimination based centroids choosing method to further refine the DAOCT's shape.

The rest of the paper is organized as follows. In Section 2, we briefly review the related works. A detailed description of the proposed DAOCT algorithm is provided in Section 3. Section 4 presents the experiment results. We make our conclusion in Section 5.

2. Problem Statement and Related Work

2.1. Problem Formulation

Finding the nearest neighbor of a query descriptor is a common step involved in many computer vision tasks. In D-dimensional space $\mathbb{R}^D$, for a given query descriptor $q \in \mathbb{R}^D$, the target is to find an element $NN(q)$, which belongs to the data set $\mathcal{X} \subset \mathbb{R}^D$, to have the shortest distance to the query vector. The nearest neighbor (NN) problem can be denoted as

$$NN(q) = \underset{x \in \mathcal{X}}{argmin}\, d(q, x) \quad (1)$$

where $d(*)$ is the distance function. When using binary descriptors the query descriptor q and data set $\mathcal{X}$ all belongs to Hamming space $H : \{0, 1\}^D$ and the distance function is to compute the total bitwise difference of two descriptors.

2.2. Related Work

For those real value descriptors, to find the nearest neighbor of a query vector usually involves laborious computation on Euclidean distance between descriptors. Numerous approaches for fast approximate nearest neighbors search (ANN) have been proposed. Usually, these ANN searching methods acquire ANN search results with high accuracy but much less computation. A common method is to use binary K-d tree to hierarchically divide space for approximate nearest neighbor search [6]. Hartley *et al.* came up with the concept of random multi K-d tree to boost its performance [16]. In K-means clustering tree, the clustering centers will be recurrently optimized. Muja *et al.* later made a concrete progress to hierarchical K-mean clustering tree by introducing a priority searching method [12]. Multiple hash tables also demonstrate its effectiveness in ANN searching, and [12] pointed out that space partition methods such as

K-d trees can acquire higher ANN searching performance than multiple hashing methods. The classic locality sensitive hashing (LSH) [2] algorithm uses a locality preserving hash function to project the high dimension real value vectors to low dimension binary ones. The main focus of hashing based methods lies in how to find an appropriate mapping matrix and binarization threshold. Methods belong to this category including minimal loss hashing, spectral hashing and semantic hashing etc [13, 7, 15, 18].

A major advantage of binary descriptors is that computing Hamming distance on binary vectors is hundreds times faster than computing Euclidean distance on real value vectors, given the same length. The proposed binary descriptors include BRIEF, FREAK and BRISK, etc. Tricinski *et al*. pointed out that due to the thick boundaries in Hamming space, the fast approximate nearest neighbor search methods designed for Euclidean space are no longer applicable to the ANN search in Hamming space [17], and the speed up rate of fast ANN searching methods in Hamming space is much worse than their counterparts in Euclidean space. A large sum of research efforts have been devoted to find some fast ANN search methods for large scale databases.

One group of ANN searching approaches in Hamming space are based on locality sensitive hashing. The key notion of these methods is that when two vectors have identical bits on some dimensions, they are more likely to be close in Hamming space. The ANN searching performance of this kind of methods depends on the chosen hashing functions. The locality sensitive hash turns the querying vector into a shorter hash code, and then go to each column in the hash table to find the hash bins which have indentical hash code. Finally it returns the best candidate in these bins by linear search. The error weighed hash [5] was proposed by Esmaeili *et al*. This method assigns a score to all vectors in the hash bins within an error range. If the hash code of a bin has higher similarity with the query vector's hash code, all data vectors in that hash bin will get a higher score. Then for each data vector, this method adds all the score it gets in different columns. Only those data vectors with high scores are returned for linear scanning. A multiple hash tables method for searching exact nearest neighbor within a given radii [14] was introduced by Norouzi *et al*. As a form of multi-index hash, this method explores the idea that when conducting exact nearest neighbors search with a given radii, in each hash table, only the examination of these buckets with error less than the radii divided by number of hashing tables is necessary.

Another family of methods are based on space partition [11, 12]. The clustering centroids are randomly selected, and the binary descriptors are assigned to its closest centroid. In this scenario, each data vector can be represented by its clustering centroid in approximate nearest neighbor searching. A representative of this kind of binary descriptors ANN search approach is the multi hierarchical clustering trees (multi-HCT) proposed by Muja *et al*. This method starts from the top and divides the dataset into m chunks. Then it recursively divides each cluster into m smaller clusters based on their distances to the randomly selected centroids, until the cluster's size is small enough. In the searching process, the clusters close to the query descriptor are returned by priority searching. The approximate nearest neighbor to the query descriptor is found by linear scanning on the returned clusters. Using multiple trees is a rudimentary but effective strategy to boost HCT's ANN searching performance.

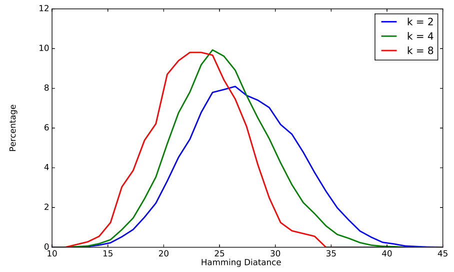

Figure 3. The distribution of the Hamming distance between data vectors in a LSH bin and query vector. Where k is the key length factor. Measured on 64 bits binary vectors.

3. Proposed Method

In this section, we begin by describing a new dynamic building strategy for agglomerative trees. In the second part of this section, we give description to the idea of border overlapping, which we adapted in DAOCT and solves $bordereffect$ in a efficient way. In the final part of this section, we represent that the dynamic building strategy not only waives the building cost, but also makes the DAOCT algorithm applicable to ever changing databases.

3.1. Dynamic Agglomerative Clustering Strategy

3.1.1 Agglomerative Clustering

For those clustering methods based on random centroids, the agglomerative method usually generates better clustering quality. Building from bottom to top, the agglomerative clustering method minimizes the distance cost in global scenario, $i.e.$ it chooses the closest clustering centroid from all centroids for each data point. The divisive clustering strategy chooses the closest centroid only from several candidates in one higher level cluster. In this way, the agglomerative clustering methods acquires better clustering quality at the cost of higher building complexity, and this is the main hindrance of its application on large databases.

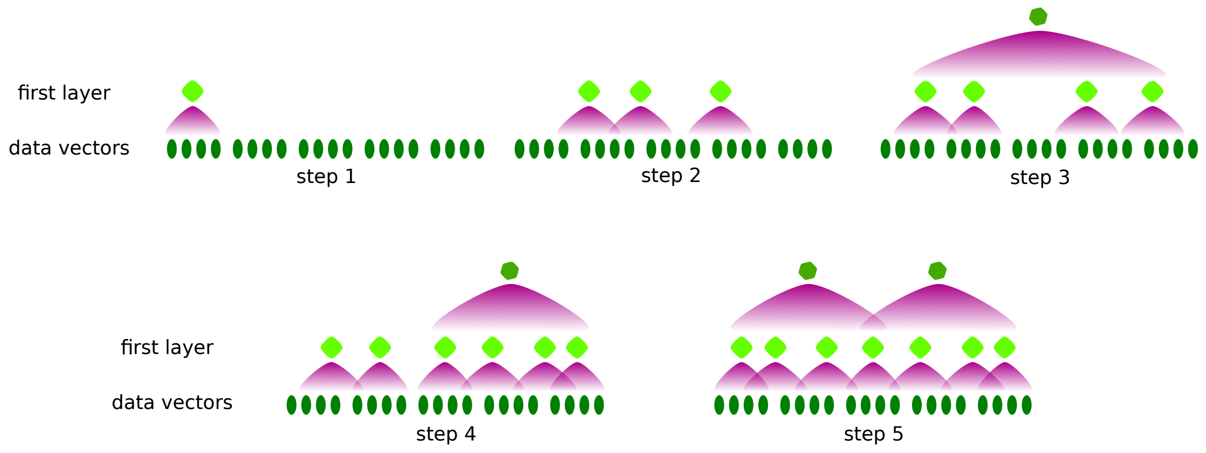

Figure 4. Illustration of DAOCT's dynamic building strategy.

3.1.2 Dynamic Building: Using Query Results

In order to avoid the long building time of agglomerative clustering tree and make it applicable to ever changing databases, we adopt a dynamic building strategy in the DAOCT algorithm.

Unlike other ANN searching methods that start with a data structure building process, the proposed DAOCT approach starts with the querying process. Because in the dynamic building strategy, we mingle the ANN querying, structure building and updating processes into the query process, and the cost of tree building is waived in this way. The key notion of the dynamic building strategy is to form a cluster in each layer of the DAOCT, by utilizing the distance information created by the querying process, and adopt the query vector as the centroids of these clusters.

There are two kinds of querying phases in it. The first is the dynamic query phase, which combines the building procedure. The second is the static query phase, which shut down the dynamic part in dynamic query phase and renders better ANN search performance. Our experiments in next section shows the ANN search performance in dynamic query phase is about the same as the state of the art methods, and the query performance in static query phase is much better.

The first node of the DAOCT comes from the first query. It is a leaf node whose centroid is the first query vector. The nearest W data vectors to the first query vector are assigned to the first node as its children. The first node in DAOCT is also a root node, which the priority search algorithm starts from. In the following queries, the algorithm performs priority search on the DAOCT and scans the data vectors which are not in the tree. The nearest W data vectors to query vector in each query process are utilized to form new leaf nodes in the DAOCT. The DAOCT adds all nodes on the same layer as the root node (which is the first node and a leaf node in this case) to the nodes queue for priority search. If the number of nodes on the same layer as the root node reaches the branching parameter m, a new node will be added to higher layer in the DAOCT. This new node will be assigned as the new root node, and all the m nodes in the previous root layer will be its children. We randomly select a child node of it and use the centroid of that child node as the centroid of the newly built root node. In next query, the algorithm adds a leaf node to DAOCT using same method. This leaf node is not in the root layer and it is not assigned to any node in higher layer, we denote this kind of nodes as *orphan nodes*. For each query, all the *orphan nodes* are added to the nodes queue for priority search. With priority search, the distance information that tells which are the closest nodes to the query vector in each layer of DAOCT is obtained. In each layer, if the recorded distance information tells the top m nearest nodes contain more than p orphan nodes, a new orphan node will be built on higher layer. The top m nearest nodes will be assigned to that node, and the centroid of that node is the current query vector. The algorithm performs these steps similarly on higher layers to build the DAOCT.

The threshold policy adapted in DAOCT sets the maximum number of children a node can share with others. The dropout strategy is when a node is added to the DAOCT, we delete the earliest added *orphan nodes* in that layer at k rate to ensure the clutering quality.

For a dynamic database, the DAOCT is created at the same the database is created. At first, the database and the DAOCT are both empty. Then the data vectors will be continuously added to the database by the vision application and ANN searches will be performed on the database simultaneously. When apply DAOCT to some databases already have large volumes, it is better to adopt an assistive ANN searching function to do joint ANN search with DAOCT in its dynamic query phase.

When the size of DAOCT becomes suitable for the size of database, its building and updating process will slow down. We shut down the dynamic part in dynamic query phase, then assign each *orphan node* to its closest centroid in higher layer and add all data not in the DAOCT to their closest leaf nodes. After that the ANN queries will be implemented in the static query phase and their performance will be better.

3.1.3 Complexity analysis

The major risk of using ANN querying prosess to handle the tree building task lies in its potential effect on querying performance. Though nearly all ANN search methods for binary descriptors involve linear scanning on returned candidates and thus possess a $O(N)$ complexity, it is still necessary to examine the effect of linear search on *orphan nodes*. It seems our method unpleasently change the searching time complexity from $O(klog(N) + LN)$ (if we use k HCTs as the assistive ANN search function, for example.) to $O(log(N)+\lambda N+LN)$, where L is a assigned parameter denotes the percentage of data returned for linear scanning. If a query is done with a large L, generally it will reap high accuracy. We only utilize those queries with high accuracy for building in dynamic query phase, and those queries with low target accuracy will be handled solely by the assistive ANN search function. This means the number of *orphan nodes*, denoted as λ, is much smaller than L in DAOCT's implementation.

Conventionaly, a agglomerative clustering tree is built by selecting some data points as the centroids, and assign data to their closest centroids, and do the same thing for upper layers. The time complexity of this building process is $O(\sum_{i=0}^{log_m(N)} N \times m^{-i} \times N \times m^{-i-1}) = O(N^2 log(N))$, where m is the branching factor. This complexity is relatively high for some large databases, and the building process is waived by dynamic building strategy. Even implemented with assistive ANN search functions like HCT or LSH, the building complexity is $O(Nlog(N))$ or $O(N)$, and they are less horrible than $O(N^2 log(N))$.

3.2. Soft border clustering

In order to tackle the *border effect* stated above, we use overlapping border strategy in DAOCT. To the best of our knowledge, all mainstream space partition methods for ANN search adopt crisp border partition strategy. In general, these crisp border clustering methods slove the border effect by creating multiple cluster codebook for one dataset. Using multiple cluster for one dataset is not efficient to boost the searching performance. For example, adding n cluster codebooks requires n times storage and building cost, but the marginal utility becomes very narrow if n is larger than 2.

The proposed overlapping strategy focuses on where the problem comes from: the cluster borders. In fuzzy clustering, data points can belong to different clusters with different memberships. However, membership system will cost much memory and computational resources. In our DAOCT implementation, if a node shares similar distances to more than one centroids, the algorithm directly assigns the node to all of these centroids. This is a cheap imitation of membership system and it functions well in a similar way.

3.3. Algorithm

The pseudo codes of the dynamic query phase of the proposed DAOCT is shown in Alogorithm 1. The DAOCT utilizes these queries which are supposed to have high ANN accuracy for building in the dynamic query phase, and the centroid of each node created by query is the query vector. Q is the set of qualified queries, the branching parameter is m, the max number of data points to examine is L_{max} and the overlap threshold is tr. It has a dropout rate of k, and PQ, R, B are set to be empty queue for each iteration. N_{root} is the number of nodes at root layer, L denotes the number of data points checked. The priority search procedure of building process is demostrated in Algorithm 2.

Algorithm 1 DAOCT (dynamic query phase)

```
1:  Input: query set $Q$, dataset $D$
2:  Output: nearest neighbor for each query $q_i$
3:  for each query $q_i$ do
4:      if $N_{root} = m$ then
5:          create new root node in higher layer
6:      $PQ \leftarrow$ nodes at root layer other than the root
7:      Travese(root, PQ, R, B)
8:      while $L < L_{max}$ and PQ is not empty do
9:          $N \leftarrow$ closest node to $q_i$ in PQ
10:         Travese(N, PQ, R, B)
11:     for layer $S_i$ under the root layer do
12:         $D \leftarrow m$ nearest nodes to $q_i$ in layer $S_i$ in $B$
13:         if number of orphan nodes in $D > tr$ then
14:             create a new node at layer $S_i + 1$, whose children are $D$
15:             delete first orphan nodes at layer $S_i + 1$ at $k$ rate
16:     search nearest $W$ data points not in tree and add them to $R$
          return $G \leftarrow$ nearest $W$ data points to $q_i$ in $R$
17:     if number of data points not in tree but in $G > tr$ then
18:         create new bottom node using $G$
```

We add all the *orphan nodes* and data points not in the tree to their closest centroids when the DAOCT is about to finish. After that the ANN queries will be implemented in the static query phase and their performance will be better.

Algorithm 2 TRAVERSE

```
 1: procedure TRAVESE(N, PQ, R, B)
 2:     if $N$ is a leaf node then
 3:         add all dats points in $N$ to $R$
 4:         $L = L + |N|$
 5:     else
 6:         $C \leftarrow$ children nodes of N and all orphan nodes at same layer of N's children
 7:         $C_m \leftarrow m$ closest nodes which haven't been chosen in this query process in C
 8:         save each node in $C$ and its distance to $q_i$ to $B$
 9:         $C_q \leftarrow$ closest node to $q_i$ in $C_m$
10:         add $C_m$ other than $C_q$ to PQ
11:         Travese($C_q$, PQ, R, B)
```

3.4. Application on dynamic data

The dynamic building strategy is designed to remove the tree building cost for static datasets. Here we extend its utility to perform ANN search on dynamic databases. We refer dynamic database to the database which:

i. serves a running operation of adding and deleting.

ii. has changes inside its data. For example, a stored data vector changed from '00001111' to '10001111'. A possible situation: in a database of features of some visual materials, there is some constant ramdon minor changes to these visual materials.

For a dynamic database, the DAOCT is created at the same the database is created. At first, the database and the DAOCT are both empty. Then the data vectors will be continuously added to the database by the vision application. If ANN searches are performed on the database simultaneously, the DAOCT will grow in pace with the database and automatically update itself. No assistive ANN function is needed for implementation on a dynamic database.

Conducting ANN searching on dynamic databases requires constant update of the data structure applied. Adding data to the space partition based methods requires reconstruction of the data structure, or it will cause inbalance. Adding data to hash table is much more convenient than space partition methods and it has only $O(1)$ complexity. However, the optimal parameters of LSH vary on different data. When the data is dynamic, constant search for best parameters and reconstruction are required to ensure LSH's performance. So without a priority searching scheme the hash table based methods are not flexible to different database size. To deal with the changes inside data, these two categories of method need to do periodic reconstruction or recurrently check, delete and add the data points.

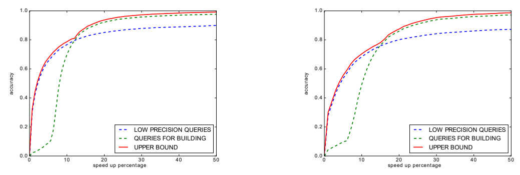

Figure 5. ANN query performance on growing data. Judged by the given L_{max}, the algorithm only adopts these queries have high accuracy for updating. If the threshold is near the intersection point of the green and blue dash curve, the ANN performance will be their upper bound.

Dynamic building scheme offers a solution to these problems with some additional requirements. Adding new data to the algorithm is quiet simple. Because the dynamic building strategy will internalize these data when doing ANN searches. As for the changes inside data, instead of doing periodic reconstruction, the dynamic building scheme updates the clustering codebook piece by piece. The earliest built *orphan node* in a layer is likely to be affected more by the changes than the nodes built later, if the changes happen evenly to all the data vectors. That *orphan node* will be deleted at k rate if a new node is added to that layer. The update procedure is processed in a stable and efficient way, and most importantly, for conventional methods, update is an addition task, but for dynamic building, the update process is incorporated in the ANN searching prosess. The limitation of this method for dynamic data is that it requires the number of queries keeps in pace with the number of new added data.

4. Experiment Results

4.1. Implementation Details

4.1.1 Datasets

The databases we chose for binary descriptors' ANN search was IRISA 1 million SIFT dataset and image patches of Statue of Liberty, Notre Dame and Half Dome. The 1 million SIFT dataset composes of a million 128 dimensional SIFT descriptors, which is of moderate size for modern internet applications. These descriptors were extracted with Hessian-affine detector. The image patch datasets include more than a million 64×64 image patches of the Statue of Liberty, Notre Dame and Half Dome. These patches were extracted from photos using Difference of Gaussian (DoG) method and they are presented in canonical scale and orientation. For each image patch, we convert it to a 128 dimensional real value descriptor using Lowes method. For all of these real value image descriptors, we use LDAHash to make 64 bits binary descriptors from them, and we randomly choose 1000 data points from each of the binary databases as the query set.

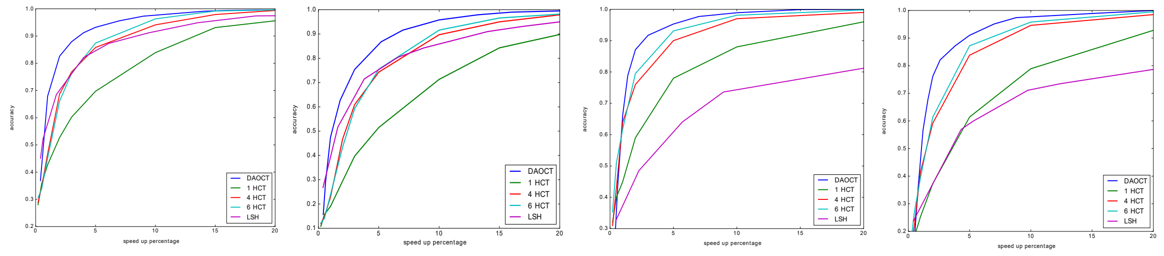

Figure 6. From left to right is the experiment result of ANN search result on 1 million SIFT database, top 2 most adjacent neighbors search result on 1 million SIFT database, top 1 and top 2 ANN search results on the image patches database.

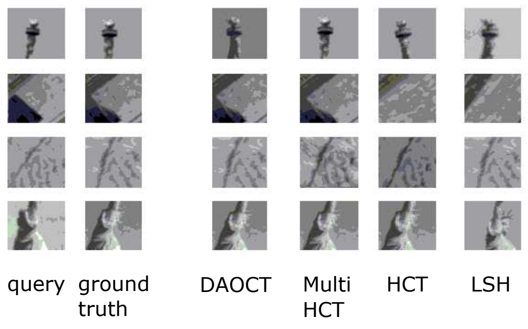

Figure 7. Illustrative examples of image patches matching results

4.1.2 Results on Approximate Nearest Neighbors Search

For comparison, we test the accuracy to speed up rate (query time compare to linear search) performance of multiple hierarchical clustering tree (multi-HCT), locality sensitive hashing (LSH) and DAOCT on same static dataset. Figure 5 shows the top 1 and top 2 approximate nearest neighbors search results on these datasets. We build 1, 4 and 6 hierarchical clustering trees for ANN search. In our implementation, the hyper-parameters of the multi-HCT method and the proposed DAOCT are quite similar. Their branch factors are set to 16, and their leaf nodes' sizes average to 130. In 1 million SIFT descriptors dataset, the average distance between query vector and it's neatest neighbor is around 7 bits. For the image patch datasets, we mingle these extracted binary descriptors into one dataset. For all the approaches, if any vector has same Hamming distance to the query vector as the nearest neighbor is found, the result will be deemed as a successful nearest neighbor retrieval.

4.1.3 Parameters Analysis

Our experiments also demonstrate the solid effectiveness of partial overlap strategy. In our implementation, there is a parameter controls the overlap threshold for nodes. Only these nodes which share less child nodes than the threshold will be added to the DAOCT. In the first figure in Figure 6, the assigned acceptable overlap rate ranges from 40% to 70%, and the accuracy to speed up rate performance increase steadily as the overlap threshold rise. It takes more constructive queries to form a DAOCT with lower overlap threshold assigned.

The second figure in Figure 6 is the result of ANN search using different branch factors on same descriptors database. The assigned overlap threshold of DAOCT is fixed to 50%. The result shows little correlation between branch factors and final performance.

For each query, the DAOCT building phase algorithm tries to build a node on each layer of the tree. Due the overlap threshold, the increment of tree nodes gradually slow down. The third figure in Figure 6 shows the tree nodes increment curve as the building phase proceeds. One can refer to the node increment curve to decide when to transform DAOCT into static query phase. The node increment curve varies with different parameters.

4.1.4 Building Time Cost

Empirically, the more high accuracy queries are used in the building phase, the better the DAOCT will be. How much queries to use depends on the parameters. We suggest that the number of queries to use times the leaf node size should be larger than the number of data points. The aggregate number of left out points depends on the overlapping thresholds and the number of queries to build the DAOCT. There will be less point left out if the leaf node size is small and queries for building are sufficient. Generally, the left out points number accounts less than 10% of the database size when the DAOCT is about to finish.

We test the query speed in DAOCT building process. Figure 7 shows the experiment result. The ANN searching performance in DAOCT building phase is at some extent inferior to the finished ones, but still maintains relatively high performance. Based on this point, we say the dynamic building strategy waives the building cost of an agglomerative clustering tree.

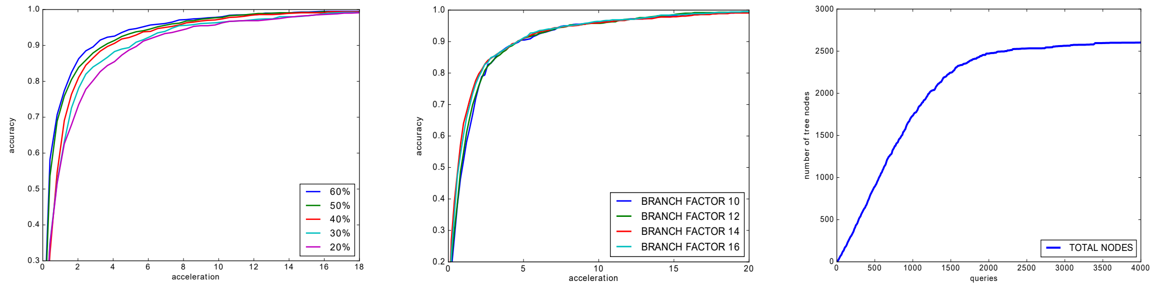

Figure 8. From left to right: i. ANN search performance of DAOCTs with different overlapping thresholds. ii. ANN search performance of DAOCTs with different branch factors. iii. Total nodes in the DAOCT as the dynamic query phase proceeds on a static database.

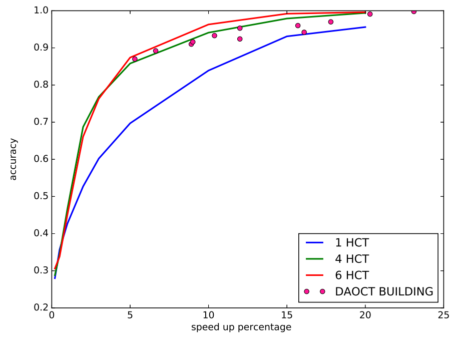

Figure 9. DAOCT performance at building phase. Four pre-built hierarchical clustering trees are used. Tested with different number of data points to examine(L_{max} in pseudo codes).

4.2. Clustering Quality

For vector quantization based algorithms, there are numerous methods to measure cluster quality. Generally, a good clustering function should let the similarity within a cluster be high and the inter cluster similarity be low. When we use a function to define the clustering quality, the clustering quality measuring function f should be defined in accordance with following three requirements: i. Scale invariance. For every clustering C of (X, d), and any positive λ, $f(C, X, d) = f(C, X, \lambda d)$, where X is the clustered dataset and $d(*)$ is the distance function. ii. For any distance measurement d_v, by which $d_v(x_i, c_i) < d(x_i, c_i)$ for any x_i belongs to clustering center c_i, and $d_v(x_j, c_j) > d(x_j, c_j)$ for x_j is not a member of clustering centroid c_j, the clustering quality measure function f should satisfy $f(C, X, d_v) > f(C, X, d)$. iii. When measure any non-trivial clustering C on X, there exists a distance function d_w allows $C = argmax f(C, X, d_w)$. As mentioned above, the nearest neighbor retrieval accuracy tests are not scale independent measurements. A scale independent clustering quality measuring strategy is needed for conducting universal comparison of divisive clustering method and the agglomerative clustering method introduced in this paper. Here we choose relative margin (RM) measurement, which is a scale independent clustering quality measure function defies the influence of databases and some predetermined parameters. The relative margin here is defined as

Branch Factor	RM(HCT)	RM(DAOCT)
14	1.07541	0.947338
18	1.05867	0.961507
20	1.06563	0.949256

Table 1. Relative margin of divisive method HCT and agglomerative method DAOCT.

$$RM_{X,d}(C) = avg_{x \in X} \frac{d(x, c_{opt})}{d(x, c_{sub})} \tag{2}$$

where c_{opt} is the clustering centroid which x belongs to, and c_{sub} is the second closest clustering center to x. The better a cluster function C is, the smaller relative margin will it have. Experiments results show the clustering in DAOCT has smaller relative margin than the clustering in HCT.

5. Conclusion

As the data generated by modern internet applications grow ever bigger, it becomes unprecedentedly important for us to develop efficient algorithms for fast information retrieval in order to relieve the heavy burden on the computer processors. The proposed Dynamic Agglomerative Overlapping Clustering Tree aimed to solve the binary descriptors ANN searching problem by introducing a low time cost building method for agglomerative clustering tree. The overlapping strategy is very effective to deal with the $bordereffect$ and makes concrete progress to the agglomerative clustering tree. The DAOCT algorithm very naturally adopted the elimination based centroids choosing strategy. It outperforms several state of the art methods in our experiments.

References

[1] A. Alahi, R. Ortiz, and P. Vandergheynst. Freak: Fast retina keypoint. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 510–517, 2012. 1

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of The ACM*, 51(1):117–122, 2008. 1, 3

[3] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: speeded up robust features. *european conference on computer vision*, 3951:404–417, 2006. 1

[4] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. Brief: Computing a local binary descriptor very fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1281–1298, 2012. 1

[5] M. M. Esmaeili, R. K. Ward, and M. Fatourechi. A fast approximate nearest neighbor search algorithm in the hamming space. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2481–2488, 2012. 3

[6] D. G. L. Jeffrey S. Bei. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1000–1006, 1997. 2

[7] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2130–2137, 2009. 3

[8] S. Leutenegger, M. Chli, and R. Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. *International Conference on Computer Vision*, pages 2548–2555, 2011. 1

[9] D. G. Lowe. Object recognition from local scale-invariant features. *International Conference on Machine Learning*, 2:1150–1157, 1999. 1

[10] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005. 1

[11] M. Muja and D. G. Lowe. Fast matching of binary features. *Conference on Computer and Robot Vision*, pages 404–410, 2012. 3

[12] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014. 1, 2, 3

[13] M. Norouzi and D. M. Blei. Minimal loss hashing for compact binary codes. *International Conference on Machine Learning*, pages 353–360, 2011. 3

[14] M. Norouzi, A. Punjani, and D. J. Fleet. Fast exact search in hamming space with multi-index hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(6):1107–1119, 2014. 3

[15] R. Salakhutdinov and G. E. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 2009. 3

[16] C. Silpaanan and R. I. Hartley. Optimised kd-trees for fast image descriptor matching. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008. 2

[17] T. Trzcinski, V. Lepetit, and P. Fua. Thick boundaries in binary space and their influence on nearest-neighbor search. *Pattern Recognition Letters*, 33(16):2173–2180, 2012. 3

[18] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. *Neural Information Processing System*, pages 1753–1760, 2009. 3