A NOVEL INDEX STRUCTURE FOR LARGE SCALE IMAGE DESCRIPTOR SEARCH

Jiangbo Yuan, Xiuwen Liu

Florida State University
Department of Computer Science
Tallahassee, FL, USA

ABSTRACT

This paper presents a $k$-means based algorithm for approximate nearest neighbor search. The proposed Embedded $k$-Means algorithm is a two-level clustered index structure which consists of two groups of centroids; additionally, an inverted file is used for recording of the assignments. The coarse-to-fine structure achieves high search efficiency using multi-assignment operations on the coarse level. At the query stage, pruning strategies are utilized to balance the trade-off between search qualities and speeds. Our algorithm is able to explore the neighborhood space of a given query efficiently. Due to its good recall/selectivity and memory efficiency, the proposed algorithm is scalable and is able to process very large databases. Experimental results on SIFT and GIST image descriptor datasets show search performance better and comparable to the state-of-the-art approaches with lower memory usage and complexity.

Index Terms— $k$-means, approximate nearest neighbor search, multi-assignment, pruning strategies, image descriptor indexing

1. INTRODUCTION

Fast nearest neighbor search in high-dimensional space is a fundamental problem for large scale content based image/video/audio retrieval systems in computer vision and multimedia communities. The scalability and accuracy of traditional nearest neighbor search algorithms are severely limited by the curses of dimensionality for data in high dimensional. As a result, real-world applications, where high dimensional data are ubiquitous, require new approaches and algorithms to overcome the limitations in needed computation time, index storage, and search quality. Recently, Approximate Nearest Neighbor (ANN) search approaches on image descriptors (SIFT [1] and GIST [2]) have demonstrated to be effective in handle large-scale image retrieval or other similar problems.

In the last decade, Local Sensitive Hashing (LSH) is one of the best known ANN methodologies. Its variants (e.g., [3, 4]) have been applied in different applications; however, LSH algorithms often require large memory usage. To overcome the memory limitation, algorithms which use limited memory (within memory-based index structures) are being proposed since they provide a better solution to meet the requirements of large scale real-time search systems. Those algorithms are often designed with more consideration of data distributions (i.e., Multiple KD-trees [5], Hamming Embedding [6], Hierarchical $k$-Means Trees [7, 8] and Product Quantization [9, 10]).

We introduce a new coarse-to-fine clustering-based index structure, Embedded $k$-Means (EMKM), which combines with multi-assignment and pruning strategies to enable exploring of the limited neighborhood space while achieves most true neighbors of a given query. By performing clustering in the residual space [9], EMKM forms a virtual tree-structure similar to hierarchical $k$-means but using an uniform clustering on the second level so with limited centroids/reference points. At the same time, performing multi-assignments on the first level increases the probabilities to have the nearest neighbors falling into the same cells while the embedded clustering avoids to fully visiting those enlarged cells (see Fig. 1). Moreover, the pruning strategy limits the number of candidate points for exhaustive search to balance the search speeds and search qualities.

The remainder of the paper is organized as follows. Section 2 introduces our new ANN algorithm in details. Section 3 discusses performance comparison between our approach and several state of the art methods, then provides certain experimental results on large SIFT and GIST datasets.

2. EMBEDDED $k$-MEANS

The index structure of Embedded $k$-Means consists of two parts. One of them contains two level of centroids and another one is an inverted file. The inverted file, which is used for recording of the clustered point groups, is built up in the same way as [7] to fit to the hierarchical structure.

2.1. Index structure construction

The EMKM index structure is mainly constructed by the following steps:

1. cluster all database points by $k$-means to get the first level coarse centroids, refer to $k = k_1$;
2. assign each database point to multiple, say $m$, of its nearest coarse centroids;
3. mix all the residual data points (by subtracting each of the assigned coarse centroids) and to be clustered by $k$-means, which forms a second level fine centroids; refer to $k = k_2$;
4. assign each residual data point to its nearest fine centroid.

Then we get $k_1$ centroids from the first level and $k_2$ from the second level, after embedding the $k_2$ residual centroids into every coarse centroid, finally we get in total $k_1 	imes k_2$ cells in the Inverted File. Given a database with $N$ points in $D$ dimensions, however, the true storage for centroids is only $(k_1 + k_2) \times D$. Fig. 1 is an illustration of this procedure. We use $k$-means to cluster those 2D random points while represent the clusters by the Voronoi cells.

Multiple assignment One of the limitations of $k$-means based ANN methods is that the choosing $k$ often suffers the trade-off between recall and precision. In other words, a larger $k$ is wanted because it gives finer clustering and filters more points out for further processing. However, it retains fewer target nearest neighbors.
in near cells that results in more cells having to be visited. In our approach, we assign each database point to \( m_a \) of its nearest centroids in the coarse level. But we must notice that the multi-assignment strategy eventually enlarges the cover range of each single cell. It gives very limited advantage if no further processing is involved. We apply a second level \( k \)-means clustering on the \( N \times m_a \) points (or randomly choosing a subset) in the residual space. (A residual point is calculated by subtracting the corresponding assigned centroid from a given data point.) Finally, each of the \( N \times m_a \) residual points are assigned to its nearest fine centroid. Successively, the assignment complexity is \( (k_1 + m_a \times k_2) \times D \) for a single database point. Please see Fig. 2 and Fig. 3 that demonstrated the advantages of our multiple assignment approach on both running time and selectivity. The selectivity means the proportion of the number of points on the final short list which are ready for exhaustive checking.

### 2.2. Query process

Due to the multi-assignment, given a query \( q \) whose coarse nearest centroid is \( C_i \) then it is actually assigned into the **bold** cell whose search range covers all the points in black circles in the fine level (see Fig. 1). In other words, the searching range in a single cell is larger than its possible assignment range for a given query \( q \). Compared with hard assignment approach, this increases the probability for reaching \( q \)'s near neighbors in a single cell. It is obviously that the multi-assignment still cannot guarantee the nearest neighbor of \( q \) is located at its nearest coarse cell. Therefore, multi-probe operations are also employed in our approach for both two levels. However, multi-assignment strategy significantly reduces the probe times have to be made in the coarse level which consequently reduces the operations made on the fine centroids.

**Pruning strategies** The pruning process consists of two steps. Firstly, we limit the amounts of visited cells on both coarse and fine levels. Assuming that in the coarse level we probe the query \( q \) to \( mp_1 \)
To configure good parameters in our approach, \( \{k_1, k_2, m_a\} \) impact on both index structure and query process while \( \{mp_1, mp_2, N_{\text{maxvisit}}\} \) are purely for query process. Empirically, \( k_1 \times k_2 \geq N \) is a good choice since it gives an approximate population \( p_{\text{cell}} = m_a \) for a single cell which is very fine but not too sparse. And \( k_1 \) can be slightly larger than \( k_2 \) considering the multi-probe complexity. At the same time, \( N_{\text{maxvisit}} \) is used as a threshold to control the range of expansion. As the increases of \( mp_1 \) and \( mp_2 \), both of the search accuracy and selectivity will get stuck in a "upper bound" after the shortlist from the pruning stage one overflows. Consequently, it is reasonable to set up the estimated size of shortlist \( \left(mp_1 \times mp_2 \times p_{\text{cell}}\right) \) from the pruning stage one to be slightly larger than \( N_{\text{maxvisit}} \).

### 3.2. Comparison with the state of the arts

**Memory usage** Jegou et al. [9] report that for SIFT 1 million data set, the index structure of FLANN [8] requires more than 250 MB of memory while for IVFADC [9] it occupies less than 25 MB. It is also reported in [7] that the index structure for their hierarchical k-means tree structure requires 143 MB for a tree with 1 million leaf nodes (leaf centroids). However, our method only needs with less than 4.9 MB \( (k_1 = 10000 \text{ and } k_2 = 100) \) for centroid file and less than 23 MB \( (m_a = 5) \) or 15 MB \( (m_a = 3) \) for the inverted file. The memory usage in total is comparable to IVFADC and much more efficient than other similar tree structuring methods. Table 2 reports separately the memory usage complexities of centroids and inverted file while Table 3 gives the total amount of cells achieved and assignment complexity for several k-means based approaches. **HKM** represents Hierarchical k-Means and **PKM** is Product k-Means or IVFADC in [9].

**Search performance** Referring to the comparison between IVFADC and FLANN in the paper [9] on SIFT 1 million dataset, we can see that our EMKM approach (refer to Fig. 2) achieves slightly better performance than IVFADC. Moreover, FLANN (including KD-trees and hierarchical k-means trees) is apparently outperformed by both EMKM and IVFADC. Our experimental environment setup is very close to Jegou et al. did in their paper [9] since we use the same YAEL library for fundamental calculations and all results were produced by similar single-core machines. The evaluation metric \( \text{precision}@1 \) is consistent with \( 1-\text{recall}@1 \) in [9] and \( \text{precision} \) in [8].

### 3.3. Large-scale experiments

Searching on large-scale SIFT datasets (SIFT50M and SIFT100M) are performed in order to evaluate the scalability and the efficiency of EMKM (see Fig. 4). We achieved about 96% precision in average of 8ms per query on SIFT50M and about 92% precision in

---

**Table 1.** Datasets used in this paper.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Database</th>
<th>Learning Set</th>
<th>Query Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT1M</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>10,000</td>
</tr>
<tr>
<td>SIFT50M</td>
<td>50,000,000</td>
<td>1,000,000</td>
<td>10,000</td>
</tr>
<tr>
<td>SIFT100M</td>
<td>100,000,000</td>
<td>1,000,000</td>
<td>10,000</td>
</tr>
<tr>
<td>GIST1M</td>
<td>1,000,000</td>
<td>500,000</td>
<td>1,000</td>
</tr>
</tbody>
</table>

---

**Table 2.** Memory usage complexities of different index structures.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Centroids (complexity)</th>
<th>Inverted File (memory in bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-means</td>
<td>( k D )</td>
<td>( N \log_2 N )</td>
</tr>
<tr>
<td>HKM</td>
<td>( \frac{h}{m-a} (k - 1) D )</td>
<td>( N \log_2 N )</td>
</tr>
<tr>
<td>PKM</td>
<td>( m k^a D^a = k^a D )</td>
<td>( N \log_2 N + m \log_2 k^a )</td>
</tr>
<tr>
<td>EMKM</td>
<td>( (k_1 + k_2) D )</td>
<td>( m_a N \log_2 N )</td>
</tr>
</tbody>
</table>

---

**Fig. 3.** Selectivity performance on SIFT1M with parameters of \( k_1 = 10000 \text{ and } k_2 = 100 \) and different \( m_a \) values.


<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Total Cells</th>
<th>Assignment Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-means</td>
<td>$k$</td>
<td>$kD$</td>
</tr>
<tr>
<td>HKM</td>
<td>$b_1 = k$</td>
<td>$b_1 lD$</td>
</tr>
<tr>
<td>PKM</td>
<td>$(k^*)^m$</td>
<td>$k^* D$</td>
</tr>
<tr>
<td>EMKM</td>
<td>$k_1 k_2 = k$</td>
<td>$(k_1 + m_1 k_2) D$</td>
</tr>
</tbody>
</table>

Table 3. Assignment complexity of different index structures.

![Fig. 4](image_url)  
Fig. 4. Search performance on datasets SIFT50M and SIFT100M with parameters of $k_1 = 10000$, $k_2 = 1000$ and $m_a = 5$.

...around 7 ms on SIFT100M. The global descriptors such as GIST take advantage of relatively lower dimensional representation of a single image and they are more attractive for scene recognition, web-scale image search or copy detection tasks [11, 12]. For GIST1M, we achieved about 95% precision took less than 60 ms (see Fig. 5). Compared to other methods, our approach is especially interested for high-precision search solutions.

4. CONCLUSION

Through comparison with the state of the art methods on both memory usage and search efficiency, we believe that the new proposed EMKM index structure within multiple assignment and pruning strategies is a promising scheme to handle searching with very large databases of image descriptors. In the future, it is possible to extend the current two level index structures to multi-levels in order to enable better performance; and a multiple tree structure could be considered in the way of randomized KD-trees does. As an indexing structure, it is also possible to combine it with the different kinds of emerging descriptor coding techniques to improve on both speed and storage.

5. REFERENCES