Efficient Approximate Nearest Neighbor Search with Integrated Binary Codes
Wei Zhang¹,², Ke Gao¹, Yongdong Zhang¹, and Jintao Li¹
¹Advanced Computing Research Laboratory, Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China
²Graduate University of the Chinese Academy of Sciences, Beijing, China
{zhangwei, kegao, zhyd, jtli}@ict.ac.cn

ABSTRACT
Nearest neighbor search in Euclidean space is a fundamental problem in multimedia retrieval. The difficulty of exact nearest neighbor search has led to approximate solutions that sacrifice precision for efficiency. Among such solutions, approaches that embed data into binary codes in Hamming space have gained significant success for their efficiency and practical memory requirements. However, binary code searching only finds a big and coarse set of similar neighbors in Hamming space, and hence expensive Euclidean distance based ranking of the coarse set is needed to find nearest neighbors. Therefore, to improve nearest neighbor search efficiency, we proposed a novel binary code method called Integrated Binary Code (IBC) to get a compact set of similar neighbors. Experiments on public datasets show that our method is more efficient and effective than state-of-the-art in approximate nearest neighbor search.

Categories and Subject Descriptors
H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

General Terms
Algorithms, Performance.

Keywords
Similarity Search, Approximate Nearest Neighbor Search.

1. INTRODUCTION
Nearest neighbor search for large-scale data in high-dimensional space is a fundamental requirement in many multimedia retrieval applications. Currently most nearest neighbor search approaches are on the basis of Euclidean metric. Focusing on the Euclidean space, given a dataset \( X \subseteq \mathbb{R}^D \), the nearest neighbor of a vector \( x \) from \( X \) is defined as \( \text{NN}(x) = \{ x' : \min_{x' \in X} d(x, x') \} \), where \( d \) is a Euclidean distance metric. The difficulty of exact NN (nearest neighbor) search has led to various indexing methods. The early tree-structure based NN methods such as kd-tree, R-tree, etc. are suffered from “curse of dimensionality” [1]. As noises and approximate factors always exist in high-dimensional data extraction processes in multimedia, Approximate Nearest Neighbor (ANN) algorithms that sacrifice precision for efficiency achieve remarkable success in multimedia retrieval field.

One of the most famous ANN methods is Locality Sensitive Hashing (LSH) [2]. LSH uses hash-table structures to filter most false positives, and then re-ranks the leftover data by Euclidean distances. LSH can achieve ANN search efficiently with a controllable high probability. However, the heavy memory consumption is a barrier to apply LSH to large-scale database.

Consequently, in recent years many approaches [3-8] use binary codes to limit memory consumption. These approaches embed high-dimensional data into Hamming space, satisfying the conditions that neighborhood relationships are retained in the embedded space as much as possible. Then distance computations of binary codes can be executed by hardware, which enable fast retrieval.

For these binary code methods, searching nearest neighbors in Euclidean space is approximated by searching similar neighbors in terms of Hamming distances between codes. However, binary codes are generated with information loss. Thus re-ranking by exact Euclidean distances is needed to find nearest neighbors, which consumes time heavily and has become the efficiency bottleneck.

Therefore, we put forward a novel binary code method called Integrated Binary Code (IBC) to get a compact set of similar neighbors and hence improve query efficiency ultimately. The IBC is based on two novel and complementary binary code methods called as Uni-bicode and Quanti-bicode respectively. Figure 1 demonstrates our method. The set of similar neighbors obtained using the IBC is the intersection of the two sets returned by Uni-bicode and Quanti-bicode. In this way, Euclidean distance computations of the similar neighbors can be reduced significantly. As a result ANN search efficiency can be improved.

*Area Chair: Marcel Worring

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MM’11, November 28–December 1, 2011, Scottsdale, Arizona, USA.
Copyright 2011 ACM 978-1-4503-0616-4/11/11...$10.00.
2. RELATED WORK

All these approaches retrieve similar neighbors by restricting Hamming distances between codes. Ideally, the nearest neighbors in Euclidean space should be returned. However, binary code searching strategy returns a coarse set of similar neighbors in Hamming distance, which will obviously decrease the search quality. An example is cited to explain this specifically. With 10k SIFT [9] vectors from INRIA BIGANN [7] database (refer to section 4.1 for detail), 32-bit codes are generated using source codes provided by the authors of SH [5]. The searching results by restricting Hamming distances between codes are showed in Table 1.

Table 1. Nearest Neighbors in Euclidean Space returned by restricting Hamming distance

<table>
<thead>
<tr>
<th>Hamming distance</th>
<th>Similar Neighbors returned (%)</th>
<th>Nearest Neighbor returned (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.0089</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>0.1641</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>5.6769</td>
<td>80</td>
</tr>
<tr>
<td>10</td>
<td>16.4539</td>
<td>96</td>
</tr>
</tbody>
</table>

Searching binary codes does not guarantee the nearest neighbors can be found, except within a large Hamming distance which also increases false positives. As is shown in Table 1, when the nearest neighbor is returned with probability 96%, there will be 16.4% similar points returned. The situation exists for other binary code methods. As reported by [4], Hamming distance restriction keeps 94% of the 5-NN and filters 77% points, there are still 23% points returned as search result.

Consequently, for ANN search re-ranking process by Euclidean distance computations should be provided to find nearest neighbors. Nevertheless, such computations consume time heavily and hence reduce efficiency. This motives us to increase the proportions of nearest neighbors to search results that are returned by Hamming distance restriction of binary codes. In the experimental section, Hamming Embedding (HE) [4] and SH [5] will be used as baseline for IBC evaluation.

3. INTEGRATED BINARY CODE
In this section, we will introduce two novel binary code methods: Uni-bicode and Quanti-bicode. Uni-bicode utilizes percentiles to partition data in principal component space while Quanti-bicode uses clustering techniques. Based on the complementary properties of the two approaches, Integrated Binary Code (IBC) method is proposed and demonstrated in section 3.3.

3.1 Percentiles Based Uni-bicode
In order to generate compact codes of high quality, Uni-bicode transforms data into principal component space as other high-dimensional indexing algorithms [4, 6, 11] using Principle Component Analysis (PCA). Assigning the number of bits is known as bit allocation. In transform space, an adaptive bit allocation algorithm is adopted. Assume a bit allocation array \( B = \{ b_1, b_2, \ldots \} \) stores the assignment on each component. Instead of allocating equal bits to different components as in [12], components with larger variances should be assigned more bits to produce a better partition. In practice, variances can be estimated as Eigen values of covariance matrix of training data. As discussed in [6], an optimal allocation is achieved when \( b_i \sim \log_2 \lambda_i \), where \( \lambda_i \) is the corresponding Eigen value. We adopt this allocation algorithm for its simplicity and effectiveness, and the details can be referred in [6].

After bit allocation, binary codes are generated by partitioning thresholds on each principal component. The thresholds are produced in Algorithm 1.

Algorithm 1. Thresholds Generation for Uniform Partition

Input: Bits Allocation Array \( B \), training dataset \( D \) after PCA
Output: \( T \) as a two dimensional array of thresholds
For each \( B(i) > 0 \) on component \( i \)
    Estimate percentile of \( \left[ 100 \times \left( j / (B(i) + 1) \right) \right] \) from \( D \);
    save \( T(i)(j) = \text{percentile} \),
End
Percentile provides a way of estimating proportions of the data that should fall below a certain percent. Percentiles split a set of ordered data into hundredths. Usage of percentiles can ensure the partition works well.

After thresholds generation, binary codes of query points or dataset points can be computed as follows: (1) on each component \( i \), set the value of bit on position \( B(i-1)+j \) to 1 if data point is larger than \( T(i)(j) \) and otherwise 0; (2) concatenate all bits together.

3.2 Clustering Based Quanti-bicode
The Quanti-bicode method is similar to Uni-bicode, except that the thresholds are generated in a different way. Firstly, Quanti-bicode transforms data into principle component space. Then the adaptive bit allocation algorithm (refer to 3.1 for detail) is executed. After generating thresholds by clustering based quantization, Quanti-bicode produces binary codes as Uni-bicode. An unsupervised clustering method is adopted for data quantization, i.e. k-means is executed on each principal component direction. The thresholds are produced as follows:

Algorithm 2. Thresholds Generation for Clustering Partition

Input: Bits Allocation Array \( B \), training dataset \( D \) after PCA
Output: \( T \) as a two dimensional array of thresholds
For each \( B(i) > 0 \) on component \( i \)
    Set the number of clustering centers \( k \) to \( B(i) \);
    Execute k-means on component \( i \) of training data;
    Save the array of k-means centers in \( T(i) \)
End

Binary codes of [6] are generated by assigning data to the index number of clustering centers. The problem of such a method is that Hamming distances between binary codes are not reflecting similarity of original data any more. To exploit similarity in Hamming space, our algorithm takes each cluster center as a threshold and generates binary codes as Uni-bicode method.
3.3 IBC based on Uni-bicode and Quantibicode

Integrated Binary Code (IBC) is a novel binary code search method to improve query efficiency. As is shown in Figure 1, the set of similar neighbors obtained using IBC is the intersection of the two sets from Uni-bicode and Quanti-bicode.

Uni-bicode uses percentiles to generate thresholds for partitioning data. Percentiles make the partitioning quantity of data to be equal. On the other hand, Quanti-bicode uses clustering technique, and hence the spatial information of data distribution is utilized, which is complementary to percentiles. Using data projected on a principle component from [11], four sample thresholds are generated using percentile and k-means clustering respectively. The distinction is exhibited in Figure 2.

Figure 2. Different partitions by Uni-bicode and Quantibicode on a sample principle component from [11]. The blue line indicates Uni-bicode and the red Quantibicode.

As can be observed, the two methods partition data differently. Taking advantage of the complementary properties, IBC search can return a refined set of similar neighbors. Thus, IBC is efficient and effective for ANN search.

3.4 ANN Search Using IBC

The ANN search algorithm using IBC is listed in Algorithm 3. The order of Uni-bicodes and Quantibicodes in IBC searching is almost uninfluential on the search result.

Algorithm 3. ANN Search Algorithm

1. Compute Uni-bicode and Quantibicode of query point y.
2. IBC searching:
   Compute Hamming distances between y and points in dataset S1 using Uni-bicodes. After filtering points whose Hamming distances are larger than dist, a sub-set S'1 is obtained. Then Compute Hamming distances between y and points in S'1 using Quantibicodes. If Hamming distance between p ∈ S'1 and y is less than dist, add p to candidate list S2.
3. For ANN search, re-rank S2 by exact Euclidean distances, and return the nearest one.

The ANN algorithm requires only a few parameters. As for IBC searching, the number of total bits and Hamming distance threshold are trade-offs between query efficiency and accuracy. Take Hamming distance dist for example. If dist is large, there will be more candidates returned and the query will achieve higher accuracy. Besides, if dist is small, there will be fewer neighbors to be processed and the query will be efficiency.

4. EXPERIMENTAL RESULTS

Our algorithms are evaluated on two datasets. One is from INRIA BIGANN dataset [7], which is a public benchmark for approximate nearest neighbor search. We use one sub-dataset of 1M 128-d SIFT descriptors for performance comparison. The other is [10] used for image retrieval. This dataset contains about 2680 images, and are categorized into 8 outdoor scene categories. Filtered-Ratio and NN-Recall are defined to illustrate the search quality. The higher of Filtered-Ratio means more noises are filtered after searching and query efficiency is improved. And higher NN-Recall means more correct nearest neighbors are found and effectiveness is improved.

\[ \text{Filtered-Ratio} = \frac{\text{Number of Points Filtered After Search}}{\text{Size of Dataset}} \]

\[ \text{NN - Recall} = \frac{\text{Number of Nearest Neighbors Returned by Search}}{\text{Size of Query Set}} \]

The experiments are performed on a workstation with 24G memory, 2.13GHz Intel CPU and 64-bit Operating system.

4.1 Approximate Nearest Neighbor Search

To show the efficiency of our algorithm, we perform nearest neighbor search on 1M SIFT vector dataset from BIGANN, using SH [5] and HE [4] as baseline.

Firstly, since SH achieves significant improvement on LSH and reports comparable results to [8] with much greater simplicity, we compare Quanti-bicode, Uni-bicode and IBC with SH in Figure 3. When hamming distance is 10, Quanti-bicode filters 94.7% points and finds 97% nearest neighbors, Uni-bicode filters 92.8% and finds 99%, while SH filters 83.5% and finds 100%. With all binary codes set to 32-bit, our two novel binary code methods are more efficient than SH with comparable precision. Moreover, IBC achieves the best Filtered-Ratio with additional memory consumption. The efficiency is improved by integration of the two complementary methods.

Figure 3. Comparison of IBC, Quanti-bicode, Uni-bicode and Spectral Hashing (SH) [5].

Secondly, to compare the binary code methods more objectively, vector quantization with multiple assignments is applied to both HE and IBC using the same parameters. In the following experiments the length of binary codes is evaluated and set to 32 bit and 64 bit respectively. The quantization process filters 92.088% of dataset points and achieves 100% NN-Recall. By varying hamming distances, Figure 4 demonstrates ANN search results of different approaches.

As can be seen from Figure 4, IBC achieves both higher NN-Recall and Filtered-Ratio than HE, i.e. IBC remains more correct nearest neighbors with less false positives. The red ellipse in the figure corresponds to a region that is well balanced between...
efficiency and precision. When the NN-Recall is near to 1.0, points are mainly filtered by quantization and binary code methods filter few points for large hamming distances.

Moreover, IBC with both binary code methods using 32-bit performs better than HE using 64-bit. This indicates that with the same memory consumption, our IBC method achieves both higher Filtered-Ratio and NN-Recall than HE. And thus IBC is more effective and efficient on the dataset.

4.2 Image Retrieval
To further verify our algorithm, we perform image retrieval on an open dataset [10] using the original 960-d GIST [10] descriptor. GIST is a global descriptor and different from SIFT in the pre experiments. To compare these methods in accuracy, we also re-rank the results returned by HE [4] and SH [5] using exact Euclidean distances. The linear scan method computes nearest neighbors using exact Euclidean distances and is used as baseline.

In this experiment, we set dist = 16 for all methods. The length of all binary codes is set to 64-bit. Under these parameters, the search results of our IBC method are almost the same as linear scan. Table 2 shows the average search time and precision of queries. The sample retrieval results are illustrated in Figure 5. Some query results of SH are different from linear scan in Figure 5, but still relevant to the queries. However, the time usage of IBC is only 55.1% of SH.

Table 2. Absolute search time and precision (5NN-Recall) comparisons of Linear Scan, IBC, SH [5] and HE [4] with searching results re-ranked by Euclidean distances. The time is measured in seconds.

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (sec)</th>
<th>IBC+ Re-rank</th>
<th>SH+ Re-rank</th>
<th>HE+ Re-rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Scan</td>
<td>0.0453</td>
<td>0.0076</td>
<td>0.0138</td>
<td>0.00267</td>
</tr>
<tr>
<td>5NN-Recall</td>
<td>1.0</td>
<td>0.982</td>
<td>0.889</td>
<td>0.451</td>
</tr>
</tbody>
</table>

Since HE only quantizes query point to the closest cluster, many nearest neighbors are lost for the quantization borders. With significant improved precision over HE, IBC is only 1.85 times slower. To conclude, our method achieves better performance than SH both in efficiency and effectiveness, and much better search quality than HE on the dataset.

5. CONCLUSION
To find nearest neighbors more efficiently in multimedia retrieval, we have proposed a novel IBC binary code method. The experiments on open datasets showed that the IBC can achieve better performance in terms of the trade-off between search quality and efficiency than existing methods.

6. ACKNOWLEDGMENTS
This work was supported by the National Basic Research Program of China (973 Program, 2007CB311100), National Nature Science Foundation of China (61003163), Co-building Program of Beijing Municipal Education Commission.

7. REFERENCES