MULTI-INDEX VOTING FOR ASYMMETRIC DISTANCE COMPUTATION IN A LARGE-SCALE BINARY CODES

Chih-Yi Chiu, Yu-Cyuan Liou, and Sheng-Hao Chou

Department of Computer Science and Information Engineering, National Chiayi University, Taiwan

ABSTRACT

In this paper, we propose a fast search method for asymmetric distance computation in large-scale binary codes. Asymmetric distances take advantage of less information loss at the query side. However, calculating asymmetric distances with the linear scan approach is prohibitive in a large-scale dataset. We present a novel algorithm called multi-index voting, which integrates the multi-index hashing technique with a voting mechanism, to select appropriate candidates and calculate their asymmetric distances. Substantial experimental evaluations are given to demonstrate that, guided by the voting mechanism, the proposed method can yield an approximate accuracy to the linear scan approach while accelerating the run time with a significant speedup. For example, one result shows that in a dataset of one billion 256-bit binary codes, examining only 0.5% of the dataset can reach 95~99% close accuracy to the linear scan approach but can accelerate over 73~128 times.

Index Terms**—** Nearest neighbor search, asymmetric distance computation, multi-index hashing and voting

1. INTRODUCTION

Searching the nearest neighbors (NN) is a problem of finding the closest data with respect to a given query. It is a fundamental requirement in many applications, such as information retrieval, signal processing, pattern recognition, and computer vision. However, within a high-dimensional feature space and a large-scale dataset volume, the NN search becomes a very challenging task when faced with the accuracy, efficiency, and memory issues simultaneously. A number of binary embedding algorithms have been recently proposed to address these issues [1][3][4][7][8][10]. By embedding the original data into the binary space, the binary embedding algorithms enable a faster NN search by hardware-supported bit operations and manageable memory consumption with the compact binary representation.

However, the use of binary embedding causes severe information loss after embedding. Even though these binary embedding algorithms try to bridge the gap through learning, an inevitable large information loss exists in mapping a real-valued vector to a binary code. Besides, the Hamming distance between two binary codes allows only a few distinct values. Hence, the NN search in the binary space causes a weaker discrimination than that in the real-valued space.

A novel concept called asymmetric distance matching for binary codes has recently been proposed, and has been shown to be more accurate than Hamming distance matching [2][5]. "Asymmetric" means that the distance is computed between two different spaces. For example, in our case, the query is a real-valued vector while the reference data are binary codes. Since asymmetric matching does not embed the query to the binary space, it takes advantage of a lesser amount of information loss at the query side. The NN search in the real-valued space would yield a more discriminating result.

We follow Gordo et al.'s formulation of asymmetric distances for binary embedding codes [2]. Assume that a binary embedding function $h(x_i) = q\big(f(x_i)\big)$ can be decomposed into two functions f and q, where $f(x_i)\colon \{\mathbf{R}\}^s \rightarrow \{\mathbf{I}\}^t$ transforms s-dimensional real-valued vector x_i to a t-dimensional real-valued vector y_i, and $q(y_i)\colon \{\mathbf{I}\}^t \rightarrow \{\mathbf{B}\}^t$ transforms y_i to a t-dimensional binary code z_i.[1] The asymmetric distance between a real-valued vector (query) and a binary code (reference) is calculated in the intermediate space $\{\mathbf{I}\}^t$. In Gordo et al.'s approach, they adopted the linear scan scheme to exhaustively calculate the asymmetric distances of all reference data. However, this is prohibitive when searching in a large-scale dataset.

In this paper, we propose a fast search method for asymmetric distance computation in large-scale binary codes. For a reference dataset of binary codes, we divide each code into m disjoint binary subvectors and store them in m different index tables. Given a query of the real-valued vector, it is transformed and divided into m intermediate real-valued subvectors. We calculate the Euclidean distances in the intermediate space between each query subvector and centroids of all entries in an index table, and rank these entries from near to far. We then sequentially scan the ranked entries across the index tables for voting. A reference binary code that has multiple subvectors close to the query, i.e., it gets enough votes from the query, is selected as a candidate. Finally, we calculate candidates' asymmetric distances and output NNs for the query.

[1] **R**, **I**, and **B** represent the original real-valued, intermediate, and embedded binary spaces, respectively.

The candidate selection algorithm is called *multi-index voting*. Empirical studies are given through substantial experimental evaluations on large-scale datasets. Results show that, guided by the voting mechanism, the proposed method demonstrates a satisfactory accuracy with a significant speedup. For example, in a dataset of one billion 256-bit binary codes, our method that examines only 0.5% of the reference dataset can reach 95~99% close accuracy to the linear scan approach but can accelerate over 73~128 times [2]. Two state-of-the-art methods for asymmetric NN search, including product quantization [5] and multi-index hashing [9] are implemented for comparison. The proposed method yields the best accuracy by taking advantage of combining multi-index voting and intermediate space computation. In particular, without the voting mechanism, the accuracy is much lower than the linear scan approach; adopting the voting mechanism can generate a burst improvement in accuracy.

The remainder of this paper is organized as follows. In Section 2, we briefly introduce the related work. Section 3 describes and analyzes the proposed multi-index voting algorithm in detail. Section 4 demonstrates and discusses the experimental results. Conclusions are given in Section 5.

2. RELATED WORK

The NN search for binary codes can be divided into two categories: symmetric and asymmetric. For the symmetric approach, a query x_Q is transformed to a binary code through the binary embedding function. The distance from x_Q to a reference data x_i is expressed as the Hamming distance of their binary codes. Matching binary codes can be done efficiently with a couple of machine instructions (e.g., XOR and POPCNT instructions). In particular, Norouzi et al. [9] proposed an efficient multi-index hashing (MIH) scheme to perform an exact search for binary codes. Reference binary codes are divided into m disjoint subvectors and hashed into m different index tables. Given a query and a search radius r, the reference codes that differ from the query by r bits or less can be exactly returned. A key factor for speedup is that for each index table, the search radius is decreased to only $\left\lfloor \frac{r}{m} \right\rfloor$ bits. This is because, when two binary codes differ by r bits or less, at least in one of their m subvectors they differ by at most $\left\lfloor \frac{r}{m} \right\rfloor$ bits.

Rather than the symmetric approach that purely exploits the binary space structure, the asymmetric approach utilizes the original space information of binary codes. However, the asymmetric distance between x_Q and x_i is computed in neither the original space nor the binary space; it is calculated in their intermediate space. Gordo et al. [2] presented two asymmetric distance computation (ADC) schemes based on statistic expectation and lower-bound. Jégou et al. [5] proposed an inverted file system with asymmetric distance computation (IVF-ADC) for approximate NN search based on product quantization. The inverted file system is served as a coarse quantizer. The residual vector between a vector and its corresponding coarse centroid is encoded with a product quantizer, which proceeds by decomposing the high-dimensional space into a Cartesian product of low-dimensional subspaces.

3. MULTI-INDEX VOTING

Suppose we have a reference dataset of n t-dimensional binary codes $\{z_i \in \{\mathbf{B}\}^t \mid i = 1, 2, \ldots, n\}$, whose correspondences are s-dimensional real-valued vectors in the original space $\{\boldsymbol{x_i} \in \{\mathbf{R}\}^s \mid i = 1, 2, \ldots, n\}$ and t-dimensional real-valued vectors in the intermediate space $\{\boldsymbol{y_i} \in \{\mathbf{I}\}^t \mid i = 1, 2, \ldots, n\}$. Every binary code z_i is divided into m disjoint subvectors, each of which comprises of $g = \frac{t}{m}$ bits (assume t is divisible by m). g should be chosen appropriately so that the system can bear O(2^g) data space occupied in memory. We then build m index tables with respect to m subvectors as follows. Denote the jth subvector as $z_i^{(j)}, j \in [1, m]$. A cluster $Z_\beta^{(j)} = \left\{ i \middle| z_i^{(j)} \in \beta \right\}$ is generated to store a set of reference identities whose jth subvectors are β, where $\beta \in \{\mathbf{B}\}^g$ is a g-dimensional binary code. We also compute the cluster mean in the intermediate space of $Z_\beta^{(j)}$:

$$y_\beta^{(j)} = \frac{1}{\left| Z_\beta^{(j)} \right|} \sum_{i \in Z_\beta^{(j)}} y_i^{(j)},$$

where $|\cdot|$ denotes the cardinality of the set. For any subvector $z_i^{(j)} = \beta$, its correspondence in the intermediate space can be considered to be $y_\beta^{(j)}$. In other words, we build a quantizer $u^{(j)}$ of the jth subvector in the intermediate space: $u^{(j)}(y_i) = y_\beta^{(j)}$ for $i \in Z_\beta^{(j)}$. $Z_\beta^{(j)}$ and $y_\beta^{(j)}$ are calculated for all j and β offline.

An online kNN search algorithm for the given query $x_Q \in \{\mathbf{R}\}^s$ is presented hereafter. The query x_Q is projected to the intermediate space as y_Q and then divided into m subvectors. For the jth subvector $y_Q^{(j)}$, we sort the clusters $\left\{ Z_\beta^{(j)} \right\}$ based on its Euclidean distances to the cluster means $\left\{ y_\beta^{(j)} \right\}$ in ascending order. Denote the sorted cluster indexes as $\left\{ B_r^{(j)} \middle| r = 1, 2, \ldots, 2^g \right\}$, where $y_{B_r}^{(j)}$ is the rth nearest to $y_Q^{(j)}$ in the intermediate space. Started from the nearest clusters of all m subvectors $\left\{ Z_{B_1}^{(j)} \middle| j = 1, 2, \ldots, m \right\}$, if a reference data x_i is stored in $Z_{B_1}^{(j)}$, we denote x_i is voted by x_Q. When the vote count of x_i accumulates more than the vote threshold T_v, x_i is put in a candidate set. The process is repeated for the next nearest clusters until the size of the candidate set is greater than the candidate threshold T_c. The search algorithm, termed *multi-index voting*, is listed in the following:

Algorithm: Multi-index voting

Input: n reference binary codes $\{z_i\}$, m index tables $\left\{Z_\beta^{(j)}\right\}$, and the query x_Q.

Output: k NN binary codes for x_Q.

```
1. Generate a t-dimensional vector in the intermediate space for $x_Q$: $y_Q = f(x_Q)$.
2. For each subvector index $j$ and binary code $\beta$, calculate the Euclidean distance between $y_Q^{(j)}$ and $y_\beta^{(j)}$:
       $ed\left(y_Q^{(j)}, y_\beta^{(j)}\right) = \left\| y_Q^{(j)} - y_\beta^{(j)} \right\|_2$.
3. For each $j$, sort the above Euclidean distances to obtain an index list $\left\{B_r^{(j)} \middle| r = 1,2,\dots,2^g\right\}$, where $y_{B_r}^{(j)}$ is the $r$th nearest to $y_Q^{(j)}$ in the intermediate space.
4. Initialize an empty candidate set C and vote counts for all $x_i$:
       $x_i.voteCount = 0,\ i = 1, 2, \dots, n$.
5. Let |C| be the size of C, $T_v$ and $T_c$ be the thresholds of the vote count and the candidate size respectively. Do the loop:
   do
     for $r = 1$ to $2^g$
       for $j = 1$ to $m$
         for $i \in Z_{B_r}^{(j)}$
           $x_i.voteCount = x_i.voteCount + 1$.
           if $x_i.voteCount \geq T_v$
             Add $x_i$ to C.
   while $|C| < T_c$.
6. For each $x_i \in$ C, calculate the asymmetric distance between $x_Q$ and $x_i$ in the intermediate space:
       $ad(x_Q, x_i) = \sum_{j=1}^{m} ed\left(y_Q^{(j)}, y_{\beta'}^{(j)}\right)$,
   where $i \in Z_{\beta'}^{(j)}$. Build a maxheap to store the $k$ smallest asymmetric distances seen so far.
7. Apply the heap sort algorithm to the maxheap and output the $k$ NN binary codes.
```

4. EXPERIMENTAL RESULT

We carried out the experiments on a public large-scale dataset: ANN_SIFT1B [6]. It contains one billion SIFT vectors (each with 128-dimensional real values) and 10,000 queries along with the ground truth of 1,000 nearest neighbors for each query. The real-valued vectors are transformed to 128/256 bits binary codes by using locality sensitive hashing (LSH). Experiments were run on a PC using Windows 7, with Intel Core i7 3.4GHz CPU (only one thread is used), 256KB L2 cache, and 64GB RAM.

We implement five NN search methods for comparison, including exhaustive symmetric distance computation (abbreviated as SDC), exhaustive asymmetric distance computation (ADC) [2], inverted file system with asymmetric distance computation (IVF-ADC) [5], multi-index hashing with asymmetric distance computation (MIH-ADC) [9], and the proposed multi-index voting with asymmetric distance computation (MIV-ADC). The former two methods perform an exact but exhaustive search. They can be considered the baselines of the symmetric and asymmetric approaches, respectively. The latter three methods act as the approximate but speedup versions of ADC. The main difference among them is in the inverted indexing: IVF-ADC employs a single index table; MIH-ADC and MIV-ADC adopt multiple index tables but with different candidate selection mechanisms. MIH-ADC selects candidates that are hit by the query only once and within $\left\lfloor \frac{r}{m} \right\rfloor$ bits Hamming distance to the query in the binary space. MIV-ADC applies a voting mechanism to select candidates if they are hit by the query at least T_v times in different index tables, and candidates are selected in order according to their Euclidean distances to the query in the intermediate space.

Figure 1 plots mean average precision (MAP) for SDC, ADC, and MIV-ADC. The abscissa axes represent T_v, and the ordinate axes represent MAP. Solid curves represent various T_c = {10K, 50K, 100K, 500K, 1M} of MIV-ADC, while dash lines represent SDC and ADC. In general, ADC stands for the accuracy upper bound for MAP in asymmetric binary code search. MIV-ADC gets an improvement on accuracy with an increasing voting threshold T_v. However, it turns out to be degenerated when T_v is close to the number of index tables m due to the cumulative quantization error in the intermediate space.

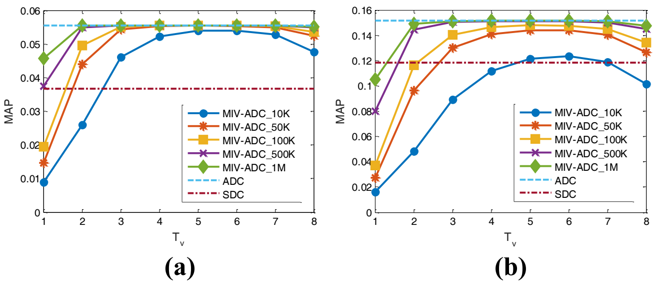


Fig. 1. MAP for querying the 1B **(a)** 128-bit and **(b)** 256-bit datasets. Solid curves represent various T_c = {10K, 50K, 100K, 500K, 1M} of MIV-ADC, while dash lines represent SDC and ADC.

Tables I and II summarize the run time for the 128-bit and 256-bit datasets, respectively. In general, as the candidate size T_c increases ten times, the run time increases much less than ten times. It implies the proposed method runs in a sublinear time complexity. In addition, the speedup of MIV-ADC over ADC is more prominent in a longer-bit dataset. As an example of the 256-bit dataset, when T_v = 3 and T_c = 500K (0.5% of the reference dataset), MIV-ADC can reach 99% close MAP to ADC but run over 73 times faster than ADC; if we change T_v = 2, MIV-ADC can reach 95% close accuracy to ADC with 128 times acceleration over ADC.

Table I. Run time (second) for querying the 1B 128-bit dataset

		$T_v = 1$	$T_v = 2$	$T_v = 3$	$T_v = 4$	$T_v = 5$	$T_v = 6$	$T_v = 7$	$T_v = 8$
MIV-ADC	$T_c = 10\text{K}$	0.0777	0.0888	0.1755	0.3586	0.6918	1.2009	2.1547	4.3299
	$T_c = 50\text{K}$	0.0870	0.1196	0.2668	0.5714	1.1317	1.9413	3.4150	6.4879
	$T_c = 100\text{K}$	0.0962	0.1475	0.3655	0.7313	1.3706	2.3637	4.1901	7.8855
	$T_c = 500\text{K}$	0.1622	0.3136	0.7169	1.4289	2.5819	4.1562	7.0476	12.7760
	$T_c = 1\text{M}$	0.2424	0.4930	1.0277	1.9269	3.3307	5.4102	8.9847	15.7816
E-SDC		3.5865							
E-ADC		14.8508							

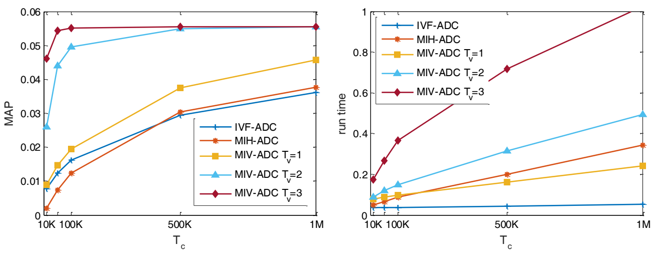
Table II. Run time (second) for querying the 1B 256-bit dataset

		$T_v = 1$	$T_v = 2$	$T_v = 3$	$T_v = 4$	$T_v = 5$	$T_v = 6$	$T_v = 7$	$T_v = 8$
MIV-ADC	$T_c = 10\text{K}$	0.1261	0.1392	0.2171	0.4161	0.9049	1.2564	2.2172	4.1633
	$T_c = 50\text{K}$	0.1526	0.1839	0.3359	0.6543	1.3167	1.9484	3.4854	7.1273
	$T_c = 100\text{K}$	0.1700	0.2352	0.4303	0.8206	1.6043	2.4331	4.2483	7.9912
	$T_c = 500\text{K}$	0.3053	0.5294	0.9369	1.6614	2.7775	4.3287	7.2132	12.9476
	$T_c = 1\text{M}$	0.4639	0.8415	1.3413	2.2694	4.0543	5.7418	9.5166	16.2088
E-SDC		4.6469							
E-ADC		68.0036							

Next, we compare and discuss the three approximate ADC methods: IVF-ADC, MIH-ADC, and MIV-ADC. Figure 2 shows MAP and run time for querying the 128-bit dataset. All methods apply the same feature decomposition scheme by partitioning the original SIFT vector into eight disjoint subvectors. In the 128-bit dataset, IVF-ADC employs one index table, whereas MIH-ADC and MIV-ADC employ eight index tables; each index table comprises of 2^{16} entries (clusters). We do not show the result of the 256-bit dataset because MIH-ADC requires more memory beyond the 64GB RAM machine.

Despite the voting mechanism, MIV-ADC yields the best accuracy among these methods. Considering MIV-ADC and MIH-ADC, they both use multiple index tables but apply different candidate selection strategies. MIV-ADC selects candidates in the intermediate space, whereas MIH-ADC does it in the binary space, which is known to be less discriminatory. Therefore, MIV-ADC can obtain higher quality candidates to yield a better accuracy than MIH-ADC. An interesting observation is that, in Figure 2(b), the run time of MIH-ADC and MIV-ADC has a crossover in the interval T_c = {100K, 500K}. In other words, when T_c is raised, MIH-ADC will take more time to collect more candidates than MIV-ADC. This is because the candidate set of MIH-ADC grows rapidly with the order of the binomial coefficient, whereas that of MIV-ADC increases gradually in the proposed voting algorithm.

For the run time, IVF-ADC is the fastest NN search method due to its simple single-index structure. Without the voting mechanism, however, the accuracy is much lower than the accuracy upper bound posed by the linear scan method ADC. It requires a large amount of candidates to be examined to approach the accuracy upper bound, and thus much more time is spent for the candidate examination. MIV-ADC that adopts the voting mechanism can generate a burst improvement in accuracy, which is close to the accuracy upper bound, by examining relatively small amount of candidates. For example in the 256-bit dataset, IVF-ADC yields 0.15 MAP by spends 5.80 seconds to scanning one hundred million candidates (10% of the 1B reference dataset), while MIV-ADC yields the same accuracy by spending 0.93 seconds to scan five hundred thousand candidates (0.5% of the 1B reference dataset) under $T_v = 3$.



(a) **(b)**

Fig. 2. (a) MAP and **(b)** run time for searching the 1B 128-bit SIFT dataset by IVF-ADC, MIH-ADC, and MIV-ADC.

5. CONCLUSIONS

In this paper, we present an approximate asymmetric NN search method for binary embedding codes. By taking advantage of multi-index voting in the intermediate space, the proposed method achieves an outstanding performance in terms of the tradeoff between the accuracy and the efficiency. Experimental results show the proposed NN search method yields a close accuracy to the linear scan approach and a significant speedup by applying a small vote threshold in examining a fraction of the dataset.

ACKNOWLEDGEMENTS

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grants MOST 103-2221-E-415 -009 -MY3 and MOST 104-2628-E-415 -002 -MY2.

REFERENCES

[1] T. Ge, K. He, and J. Sun, "Graph cuts for supervised binary coding," In *Proceedings of European conference on Computer Vision* (Zurich, Switzerland, Sep. 6-12, 2014). ECCV'14.

[2] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik, "Asymmetric distances for binary embeddings," *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 36, No. 1, pp. 33-47, 2014.

[3] K. He, F. Wen, and J. Sun, "K-means hashing: an affinity-preserving quantization method for learning binary compact codes," In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition.* (Portland, USA, Jun. 23-28). CVPR'13.

[4] G. Irie, Z. Li, X. M. Wu, and S. F. Chang, "Locally linear hashing for extracting non-linear manifolds," In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition* (Columbus, USA, Jun. 23-28, 2014). CVPR'14.

[5] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 33, No. 1, pp. 2481-2488, 2011.

[6] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: re-rank with source coding," In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing* (Prague, Czech Republic, May 22-27, 2011). ICASSP'11.

[7] W. Liu, J. Wang, S. Kumar, and S. F. Chang, "Hashing with graphs," In *Proceedings of International Conference on Machine Learning* (Bellevue, USA, Jun. 28-Jul. 2, 2011). ICML'11.

[8] M. Norouzi and D. J. Fleet, "Minimal loss hashing for compact binary codes," In *Proceedings of International Conference on Machine Learning* (Bellevue, USA, Jun. 28-Jul. 2, 2011). ICML'11.

[9] 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*, Vol. 36, No. 6, pp. 1107-1119, 2014.

[10] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," In *Proceedings of Annual Conference on Neural Information Processing Systems* (Vancouver, Canada, Dec. 8-13, 2008). NIPS'08.