What Is the Most Efficient Way to Select Nearest Neighbor Candidates for Fast Approximate Nearest Neighbor Search?

Masakazu Iwamura, Tomokazu Sato and Koichi Kise
Graduate School of Engineering, Osaka Prefecture University
{masa, kise}@cs.osakafu-u.ac.jp

Abstract

Approximate nearest neighbor search (ANNS) is a basic and important technique used in many tasks such as object recognition. It involves two processes: selecting nearest neighbor candidates and performing a brute-force search of these candidates. Only the former though has scope for improvement. In most existing methods, it approximates the space by quantization. It then calculates all the distances between the query and all the quantized values (e.g., clusters or bit sequences), and selects a fixed number of candidates close to the query. The performance of the method is evaluated based on accuracy as a function of the number of candidates. This evaluation seems rational but poses a serious problem; it ignores the computational cost of the process of selection. In this paper, we propose a new ANNS method that takes into account costs in the selection process. Whereas existing methods employ computationally expensive techniques such as comparative sort and heap, the proposed method does not. This realizes a significantly more efficient search. We have succeeded in reducing computation times by one-third compared with the state-of-the-art on an experiment using 100 million SIFT features.

1. Introduction

Finding the nearest neighbor (NN) of a given query, called nearest neighbor search (NNS), is a simple but important task. An approximate nearest neighbor search (ANNS) is an NNS in which an approximation is introduced. ANNS problems are actively studied and widely applied to many applications such as near-duplicate detection [13], large-scale object recognition [21, 26, 11, 14, 12], document image retrieval [25], camera-based character recognition [10, 9]. The required features for ANNS methods are accuracy in finding the true NNs, and efficiency in time and space. In this paper, we focus on the relationship between accuracy and computational efficiency.

Taking into account the relationship between accuracy and computation times is inherent in NNS problems. If the computation time is not an issue, NNS problem can always be solved by a brute-force search, i.e., calculating all the distances between the query and the data. This naive solution is not practical because it can take a long time, especially for large datasets. A way to reduce computation times is to reduce the data submitted to the brute-force search. That is, a limited number of data, where we call *NN candidates*, is selected in the selection process followed by the brute-force search. Only in the selection process is there occasion to introduce improvements in the NNS problem and it needs to be fast and accurate. Lengthy computation times in selecting the NN candidates negates the advantage in reducing the data. By not being accurate in selecting NN candidates, the number of NN candidates needs augmenting to compensate for the low accuracy. Computation times though would lengthen in the brute-force searches.

The performance of most existing methods is evaluated based on accuracy as a function of the number of candidates. The number of candidates determines computation times for brute-force searches. However, it is independent of the computation times of the selection process. Hence papers ignoring the computation time of the selection process tackle only a part of the ANNS problem[1].

Some papers take into account computation cost properly (e.g., [20, 12, 2]). Among them, the most efficient method is the inverted multi-index (IMI) [2]. However, on the question, "*What is the most efficient way to select NN candidates?*," we found that the method does not capture the essence of the NN candidate selection. An intuitive notion of what we mean is the following. Imagine that there are n data and a task to find NN candidates of size k ($k < n$). The worst approach is sorting them all, a task which takes time $O(n \log n)$. A better way is to use a partial sort technique using a priority queue. That is, each datum is added to the priority queue and the best k data are kept. This is

[1] One might believe that computation times for the selection process can be ignored in comparison with that for the brute-force search. However, a good ANNS method reduces the computation times for brute-force searches. As a result, these computation times are not the dominant factor.

known to take time $O(n \log k)$. IMI takes this approach. However, there is a faster sort algorithm—the bucket sort. It can complete the task in time $O(n)$ because it does not perform comparisons between data. Thus if we can prepare appropriate buckets, the selection of NN candidates can be much faster. This makes sense because our purpose is just finding NN candidates; there is no need to arrange buckets in the ascending order of their distances.

Towards a practically fast ANNS method for large scale datasets, we propose a more efficient ANNS method compared with others, which involves the most efficient way of finding NN candidates. The proposed method is named bucket distance hashing (BDH). In the experiments, we compared the proposed method with various representative ANNS methods on the same platform with respect to the criterion, *recall as a function of computation time*, in addition to the commonly used criterion, *recall as a function of the number of candidates*. Although we focus on the NN search, the same technique is directly applicable to the k-NN search problem.

2. Related Work

For an efficient selection of NN candidates, data are usually indexed in advance. Based on the structure for indexing, ANN methods can be divided into tree-based and hash-based. Hash-based methods are also classed either as data independent or data dependent. The former does not use data for indexing though the latter does.

2.1. Tree-based approach

FLANN is a representative tree-based method [20]. It automatically selects the best method among the randomized kd-tree [24], the hierarchical k-means [21] and the brute-force search, and tunes parameters for a given dataset. In the experiments, randomized kd-tree and hierarchical k-means were compared with the proposed method as FLANN itself did not work for large datasets.

2.2. Hash-based data independent approach

Locality sensitive hashing (LSH) is a representative hash-based data independent method [4, 1]. There are many improvements (e.g., [19, 30]). However, its performance is known to be worse than the data dependent indexing methods. We also confirmed that LSH is far slower than the data dependent methods that follow.

2.3. Hash-based data dependent approach

The data dependent methods can be categorized into those performing the selection of NN candidates in the Euclidean space and the Hamming space.

Selection in the Euclidean space In this category, the methods use quantization and data compression techniques. Vector quantization (VQ) is used in some ANNS methods such as VQ-index [27] and IVFADC [12]. Product quantization (PQ) is used in IMI [2]. As mentioned above, IMI outperforms IVFADC. Transform coding is used in [3]. The process can be regarded as equivalent to scalar quantization (SQ). From the category, we selected IVFADC and IMI for the evaluation; both are reviewed in the next section. Although we implemented transform coding, it was not comparable to those two.

Selection in the Hamming space In this category, data are represented by binary codes. Their use helps reduce memory usage and computation times in Hamming-distance calculations because of the efficient bitwise XOR operation. Because of these good properties, many ANNS methods based on binary codes have been proposed (e.g., [29, 16, 17, 22, 6, 7, 28, 15]). Among them, spectral hashing (SH) [29] is a representative method.

To achieve high recall, long codes, of length 128 or 256 bits long, are often used. Actually handling such a long code is not easy. A baseline method for finding NN candidates in Hamming space is the linear search which, while fast, is intractable when applied to large datasets. One might think that the tree or hash structure helps to achieve sub-linear searches. However, the number of data having the same Hamming distance to the query explode as the distance increases, which prevents the NN candidate selection process from being efficient [31]. As pointed out in [16], one idea is to use the original LSH for bits [8]. However, the data independent approach is inefficient as mentioned above. Another idea might be to apply more sophisticated approaches used in existing ANNS methods designed for Euclidean space. We applied the proposed selection method of NN candidates to SH. However, the performance did not compare well with other representative methods.

3. Existing Hash-based Indexing Methods Using Vector and Product Quantization

Reviewing IVFADC and IMI in this order helps understand the proposed method BDH. Thus in this section, we present them in greater detail than Sec. 2.

3.1. IVFADC

IVFADC [12] indexes data using VQ in selecting NN candidates efficiently. Data are divided into clusters using the k-means clustering algorithm. Given a query, the clusters close to the query are searched and data belonging to the clusters are selected as NN candidates. There is a possibility that some data close to the query are not included in the NN candidates. However, VQ is known to be the best quantization method in respect to quantization error for the same number of clusters [5]. Thus, it is expected to maximize the accuracy of NN candidate selection.

3.2. Inverted multi-index (IMI)

As a better solution than IVFADC, Babenko and Lempitsky proposed IMI [2], which uses PQ instead of VQ. PQ is an intermediate quantization method between SQ and VQ; a vector space is divided into subspaces and then VQ is applied to the subvectors in each subspace. PQ generally produces larger errors than VQ with the same number of clusters. Nevertheless, computation times to achieve the same recall can be reduced with the multi-sequence algorithm (MSA).

Figure 1 illustrates an overview of IMI. Referring to the figure, there are four clusters in the subspace 1 and three clusters in the subspace 2. The centroids of the clusters are represented by C^i_j, where i labels each subspace and j labels a cluster in the i-th subspace. Initially, the squared distances between the query denoted by a star and the centroids in the subspaces are calculated. The squared distance in the original space is given by the sum of the squared distances in the subspaces. Then, if all the distances in the original space are calculated, the centroids close to the query in the original space are found. However, it is not necessary to calculate all the distances in the original space, because centroids in subspaces with large distances cannot contribute to the process. Hence, they are ignored in MSA.

Figure 2 provides an overview of MSA. In MSA, the squared distances in each subspace are sorted first. Then, the centroids in the subspaces are examined in ascending order of distance. As shown in Fig. 2(a), the first cluster examined is the direct product $C^1_1 \times C^2_1$ whose squared distance is 2. The neighboring clusters $C^1_2 \times C^2_1$ and $C^1_1 \times C^2_2$ are selected as candidates to be examined in the next turn. Figure 2(b) shows the next step when the candidate cluster $C^1_2 \times C^2_1$ whose squared distance is 3 is selected because its squared distance is the smallest among the candidates. Then, the cluster $C^1_3 \times C^2_1$ whose squared distance is 6 is selected as a new candidate. The reason that the cluster $C^1_2 \times C^2_2$ (with squared distance of 5 but hidden in the figure) is not selected as a candidate is that at least one candidate is guaranteed to have a smaller distance than the cluster (see [2] for more detail). This process lasts until a sufficient number of NN candidates is obtained.

As seen above, MSA compares the distances between clusters and arranges them in ascending order of distance. This is not required in selecting the NN candidates. In this regard, the method does not capture the essence of NN candidate selection. The figures 1 and 2 illustrate the case when the feature space is divided into two. If it is divided into more than two, the computational cost expands. Thus it is reported that IMI performed best when the feature space is divided into two. We shall show that space division into more than two achieves better performance with a more efficient algorithm of selecting NN candidates.

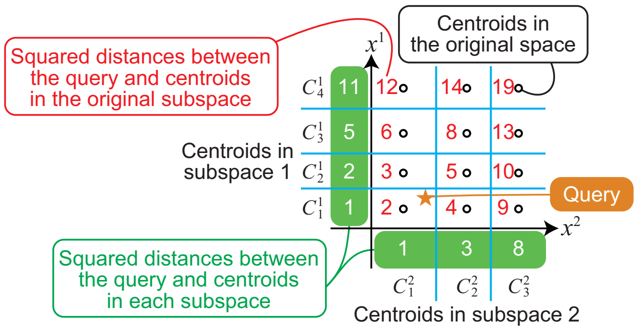

Squared distances between the query and centroids in the original subspace
Centroids in the original space
x^1
C^1_4
C^1_3
C^1_2
C^1_1
Centroids in subspace 1
11
5
2
1
12
14
19
6
8
13
3
5
10
2
4
9
Query
x^2
1
3
8
C^2_1
C^2_2
C^2_3
Squared distances between the query and centroids in each subspace
Centroids in subspace 2

Figure 1: Overview of the inverted multi-index (IMI) when the feature space is divided into two.

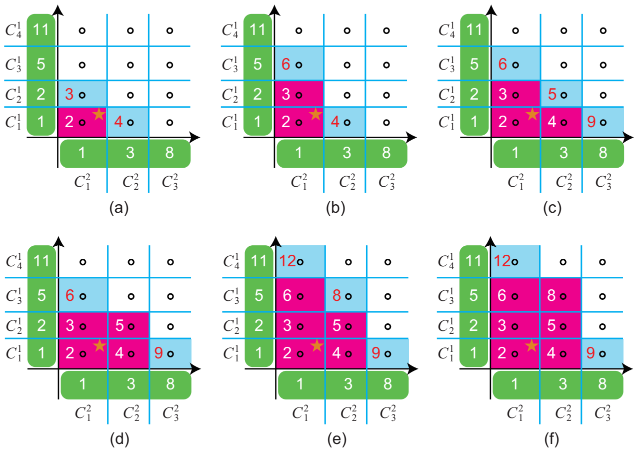


Figure 2: Overview of the multi-sequence algorithm (MSA). The algorithm proceeds in the order from (a) to (f). The clusters in magenta represent those already selected and those in cyan are to be examined in the next turn.

4. Proposed Method

4.1. Overview

As mentioned in Sec. 1, the key idea of the proposed method is to select NN candidates without comparing data or clusters. We apply the branch and bound algorithm to the problem to which MSA is applied.

Figure 3, corresponding to Fig. 1, provides an overview of the proposed method. In the figure, paths from query (Q) to the centroids (C) in the original space are drawn. The left and right halves of the paths represent the squared distances $\{d^i_j\}$ between the query and centroids in subspaces 1 and 2, respectively. d^i_j shares its indexes with C^i_j. Then, an upper bound of the squared distance is determined. The

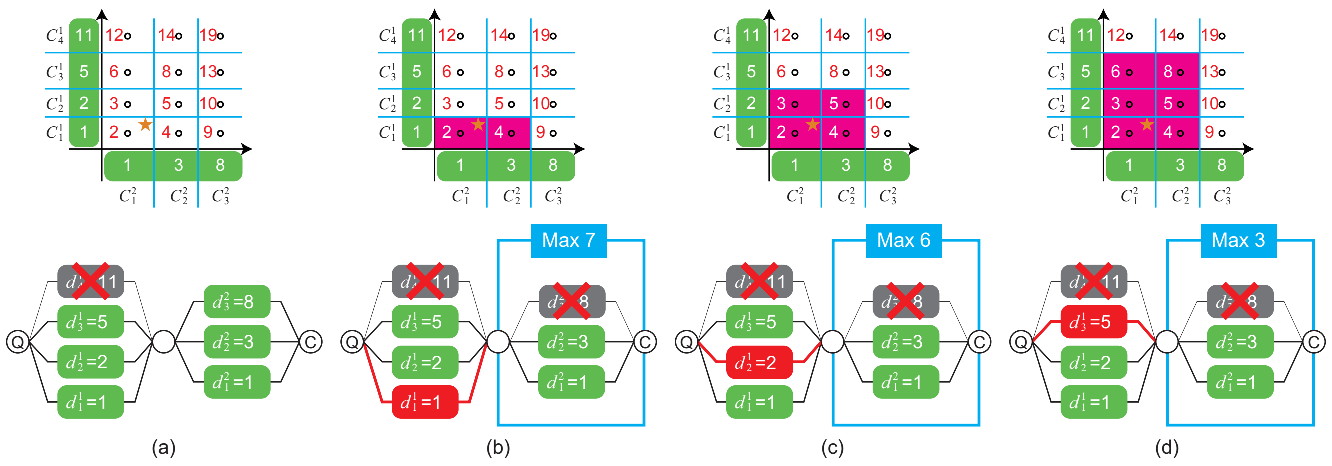
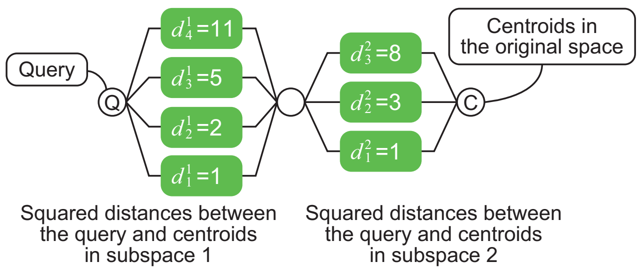


Figure 4: Proposed algorithm when the upper bound of the squared distance is 8. The algorithm proceeds in the order from (a) to (d).



Query
Centroids in the original space
Squared distances between the query and centroids in subspace 1
Squared distances between the query and centroids in subspace 2

Figure 3: Overview of the proposed method when the feature space is divided into two. It corresponds to Fig. 1. Paths from query (Q) to centroids (C) in the original space are drawn.

upper bound is incremented throughout the process. All the paths whose total distances are less than an upper bound are selected in the following process.

Figure 4 illustrates the proposed algorithm when the upper bound of the squared distance is 8. The algorithm proceeds in the order from (a) to (d). In (a), when the upper bound is set at 8, the path with $d_4^1 = 11$ is immediately removed because the path distance is larger than the upper bound. Hereafter, each path in the subspace 1 (left half) is examined. In (b), the path with $d_1^1 = 1$ is examined. As the difference between the path distance and the upper bound is 7, the paths with $d_1^2 = 1$ and $d_2^2 = 3$ on the subspace 2 (right half) are selected as NN candidates. In (c), the path with $d_2^1 = 2$ is examined. As the difference between the path distance and the upper bound is 6, the paths with $d_1^2 = 1$ and $d_2^2 = 3$ on the subspace 2 (right half) are selected as NN candidates. In (d), the path with $d_2^1 = 2$ is examined. As the difference between the path distance and the upper bound is 3, the paths with $d_1^2 = 1$ and $d_2^2 = 3$ on the subspace 2 (right half) are selected as NN candidates. Finally, the same results are obtained as in Fig. 2.

4.2. Preparation

To explain the proposed method in more detail, some definitions and preparations are given. Let us assume both query, denoted by $\boldsymbol{q}$, and data are represented as vectors. Let N and D be the number of data, and the dimensionality of query and data, respectively. We apply the principal component analysis (PCA) to the data so as to minimize the estimation error of distance, and obtain the largest u principal components (eigenvalues) $l_1, l_2, \ldots, l_u$ in descending order. Let $\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \ldots & \boldsymbol{v}_u \end{bmatrix}$ be the matrix of eigenvectors corresponding to the u eigenvalues. u corresponds to the dimensionality of the multi-dimensional hash table. We introduce a way to automatically determine u later. Hereafter a cluster refers to a bucket.

4.3. Indexing and parameter tuning

PQ divides the feature space into p-dimensional subspaces, i.e., a u-dimensional vector $\boldsymbol{x}$ in the subspace spanned by u eigenvectors is represented as $\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$, where $m = u/p$. Then, k-means clustering is applied to each p-dimensional subspace.

With the exception of p, we present an automatic parameter tuning algorithm. Let us begin with the automatic parameter selection for k-means clustering. The purpose here is to divide data into k_i clusters in the i-th subspace, and determine $\{k_i\}$ and the centroids of the clusters in subspaces. Let $\boldsymbol{x}_{is}$ be the i-th subvector of the s-th datum, and C_j^i be the j-th centroid in the i-th subspace. Then, the quantiza-

Algorithm 1: Adaptive quantization with automatic parameter tuning. This algorithm finds the number N_{bkt} of buckets which are closest to the number N of data.

```
    Input: $p$, $D$, $N$
    Output: $N_{\text{bkt}}$, $\{k_i\}$, $\{C^i_j\}$
 1  $m \leftarrow \lfloor D/p \rfloor$;  $k_i \leftarrow 1$, for $i = 1, \ldots, m$;  $N_{\text{bkt}} \leftarrow 1$;
 2  Calculate the initial $\{C^i_j\}$, which are means in subspaces;
 3  repeat
        // Preserve the current values of $N_{\text{bkt}}$, $\{k_i\}$ and $\{C^i_j\}$
 4      $N'_{\text{bkt}} \leftarrow N_{\text{bkt}}$;  $\{k'_i\} \leftarrow \{k_i\}$;  $\{C^{i\prime}_j\} \leftarrow \{C^i_j\}$;
        // Increment by 1 the number of clusters in the subspace
           having the largest error
 5      Calculate $\{E_i\}$;  $t \leftarrow \underset{i}{\operatorname{argmax}}\, E_i$;  $k_t \leftarrow k_t + 1$;
 6      Update the centroids in the $t$-th subspace;
        // The number of buckets is determined by the product
           of the number of clusters
 7      $N_{\text{bkt}} \leftarrow \prod_i^m k_i$;
 8  until $N_{bkt} > N$;
 9  if $N/N'_{bkt}$ is closer to 1 than $N/N_{bkt}$ then
        // Revert to the previous values of $N_{\text{bkt}}$, $\{k_i\}$ and $\{C^i_j\}$
10      $N_{\text{bkt}} \leftarrow N'_{\text{bkt}}$;  $\{k_i\} \leftarrow \{k'_i\}$;  $\{C^i_j\} \leftarrow \{C^{i\prime}_j\}$;
11  end
```

tion error E_i in the i-th subspace is defined as

$$E_i = \sum_{s=1}^{N} \left(\boldsymbol{x}_{is} - C^i_{q_i(\boldsymbol{x_{is}})} \right)^2 , \tag{1}$$

where $q_i(\boldsymbol{x})$ is the assignment function given as

$$q_i(\boldsymbol{x}) = \arg\min_j \left(\boldsymbol{x} - C^i_j \right)^2 . \tag{2}$$

Because a large quantization error causes a large estimation error in distance, we take up the strategy of minimizing the largest quantization error E_{max} in the subspaces. Algorithm 1 outlines the proposed algorithm. It increases the number of clusters in the subspace having the largest quantization error, and ends when the total number of buckets approaches the number of data. As a result, the feature space is divided into N_{bkt} buckets, which corresponds to the hash size of the multi-dimensional hash table. We empirically found the stopping condition from experiments over different numbers of artificial data following the normal and uniform distributions and real ones such as SIFT and GIST features.

The algorithm also determines the dimensionality u of the multi-dimensional hash table. If the number k_i of clusters in a subspace is 1, the quantization error in the subspace is less than E_{max}. In such a case, the subspace need no dividing. Assuming the eigenvalues are arranged in descending order, there exists m' which satisfies $k_i > 1$ for $i = 1, \ldots, m'$ and $k_i = 1$ for $i = m' + 1, \ldots, \lfloor D/p \rfloor$. Then, ignoring the latter half ($i \geq m' + 1$), u is set to $m'p$.

Algorithm 2: Incremental search region algorithm to efficiently obtain the minimum size c of NN candidates.

```
    Input: $\{d^i_j\}$, $c$
    Output: a list of buckets containing NN candidates
 1  $n \leftarrow 0$;  $L \leftarrow 0$;  $U \leftarrow \sum_{b=1}^m \min_j d^b_j + \Delta$;
    // Pre-calculate the minimum and maximum partial bucket
       distances in from the $(i+1)$-th to the $m$-th subspaces
 2  for $i = 1$ to $m - 1$ do
 3      $d^i_{\text{min}} \leftarrow \sum_{b=i+1}^m \min_j d^b_j$;
 4      $d^i_{\text{max}} \leftarrow \sum_{b=i+1}^m \max_j d^b_j$;
 5  end
 6  while $n < c$ do
        // Find new NN candidates
 7      $n \leftarrow n +$ CircuitBucketsFunction$(1, 0)$;
        // Extend the search region
 8      $L \leftarrow U$;  $U \leftarrow U + \Delta$;
 9  end
```

Algorithm 3: Function CircuitBucketsFunction(i, d).

```
    Input: $i$: index of subspace, $d$: partial bucket distance
    Output: $n$: the number of NN candidates
 1  if $i < m$ then
        // Find an index in the $i$-th subspace satisfying the
           condition, and then continue the search in the
           $(i+1)$-th subspace
 2      forall the $j$ do
 3          if $L \leq d + d^i_j + d^i_{\text{max}}$ & $d + d^i_j + d^i_{\text{min}} < U$ then
 4              $h_i \leftarrow j$;
 5              CircuitBucketsFunction$(i + 1, d + d^i_j)$;
 6          end
 7      end
 8  else // Return the bucket indexed by the $m$ indexes
 9      forall the $j$ do
10          if $L \leq d + d^m_j$ & $d + d^m_j < U$ then
11              Return the bucket whose hash value is
                $\boldsymbol{H} = \{h_1, h_2, \cdots, h_m\}$ and the number of
                data in the bucket as $n$;
12          end
13      end
14  end
```

4.4. Efficient selection of NN candidates

Let c be the number of NN candidates required. We propose an algorithm to select at least c NN candidates with the smallest estimated distances in a step-by-step manner. Algorithms 2 and 3 show the detailed procedure, a part of which is already presented in Sec. 4.1. Algorithm 2 finds buckets whose distances are in the range between the lower bound L and upper bound U of the estimated distance, calling recursively Algorithm 3. The initial values of L and U are 0 and the squared distance between the query and the nearest bucket, respectively. As long as the number n of the

NN candidates obtained so far is less than c, the process is repeated with updated search regions. Δ is the increment for L and U. We used 1/100 of the sum of the eigenvalues as Δ.

5. Experiments

We performed experiments to compare the proposed method (BDH) with representative ANNS methods, which include IVFADC [12] and IMI [2] as hash-based methods and randomized kd-tree (RKD) [24] and hierarchical k-means (HKM) [21] as tree-based methods from the components of FLANN [20][2]. All the methods were implemented in the C++ language. We implemented the hash-based methods by ourselves, referring to the MATLAB version of IVFADC[3] and C++ source code of IMI[4]. IVFADC, IMI and BDH share most of the source code. This avoids differences in implementation and makes for a fairer comparison. For the tree-based methods, we used the C++ source code of FLANN obtained at the authors' homepage[5]. FLANN itself did not work for large datasets. We experimentally explored the best parameters for each method, and determined ones which achieved the best performance in the criterion, recall as a function of computation time. The parameters used in the experiments are summarized in Table 1.

We used the BIGANN database[6] containing 1 billion 128-dimensional SIFT descriptors [18] and the *80 Million Tiny Images*[7] containing approximately 80 million 384-dimensional GIST descriptors [23]. For the former, 1 million, 10 million and 100 million datasets were used; 1000 data were used as queries. For the latter, 100 thousand, 1 million and 10 million datasets were used; the first data in the first 1000 categories (1000 data in total) were removed from the datasets and used as queries. For both databases, the smallest dataset, which is a subset of larger datasets, is used for training (parameter tuning).

We employed servers where 4 CPUs (AMD Opteron 6174, 2.2GHz, 12 cores) and 256GB memory were installed. All data were stored on memory. Each program was executed as a single thread on a single core.

5.1. Experiment 1: Recall vs computation time

We compared the methods in the criterion, recall as a function of computation time. Computation time is defined as the average computation time required to obtain an answer from the time the query is given. Accuracies for small computation times were relatively low because of limitations in measuring computation times. Figure 5 shows experimental results on the SIFT and GIST datasets. They show that the proposed method performed much better than the others. Figure 5(c) shows that BDH was two times faster than IMI and 4.5 times faster than IVFADC in 90% recall, and 2.9 times faster than IMI and 9.4 times faster than IVFADC in 60% recall. Comparing Figs. 5(a) to 5(c), the advantage of the proposed method became clearer as dataset size increased. This shows scalability of the proposed algorithm. In contrast, comparing Figs. 5(d) to 5(f), the advantage of the proposed method does not change. This might be due to the high dimensionality of the feature vectors.

The fact that IVFADC and IMI did not scale is caused by relatively large computational overhead. Letting G be the number of clusters in the original space, IVFADC needs time G for the distance calculation in selecting NN candidates. IMI needs time $2\sqrt{G}$ for distance calculation, and time $\sqrt{G}\log\sqrt{G}$ in sorting the distances in the subspaces.

5.2. Experiment 2: Recall vs number of candidates

We also compared the methods in the commonly used criterion, recall as a function of the number of candidates. With this criterion, we show the experimental results of hash-based methods. We selected the best parameters subject to the criterion, recall as a function of computation time. Figure 6 shows that BDH was also better than IMI and IVFADC in the standard criterion.

6. Conclusions

In the approximate nearest neighbor search (ANNS) process, only the selection of nearest neighbor candidates has any leeway for improvement. In this paper, we pointed out the importance of evaluating the computational costs of ANNS methods. Because taking the computation time into account is inherent in the ANNS problem, researches ignoring computation time have only a limited impact. We also pointed out that the state-of-the-art ANNS method, IMI, does not capture the essence of NN candidate selection. That is, although the goal of NN candidate selection is simply finding NN candidates, it employs unnecessary processes such as comparative sort and heap.

Finally, we proposed a new ANNS method that takes into account the cost of NN candidate selection. The proposed method is based on the branch and bound algorithm which produces a significantly more efficient search. We also presented an automatic parameter tuning algorithm which, except for one parameter, can automatically determine the best parameters. In an experiment using 100 million SIFT features, the proposed method succeeded in reducing the computation time by one third compared with the state-of-the-art; the proposed method was two times faster in 90% recall, and 2.9 times faster in 60% recall.

[2] Although LSH [4], spectral hashing [29], transform coding [3] were examined, their performance was too poor. Therefore, we have omitted them.

[3] http://www.irisa.fr/texmex/people/jegou/ann.php

[4] http://arbabenko.github.com/MultiIndex/

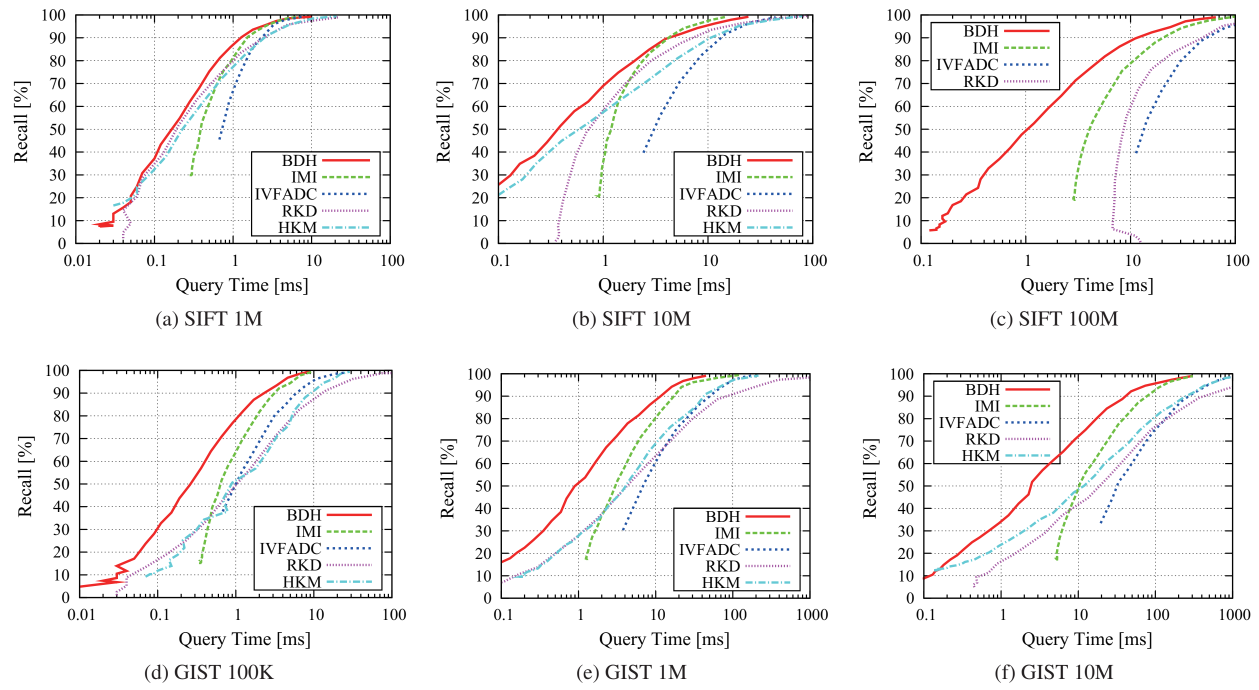
[5] http://www.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN

[6] http://corpus-texmex.irisa.fr/

[7] http://horatio.cs.nyu.edu/mit/tiny/data/

Table 1: Parameters used in the experiments.

Methods	Parameters	SIFT 1M	SIFT 10M	SIFT 100M	GIST 100K	GIST 1M	GIST 10M
BDH	$\log_2 \|C\|$, P	20, 5	26, 3	28, 5	18, 3	22, 5	24, 5
IMI	$\log_2 \|C\|$	14	18	20	16	18	22
IVFADC	$\log_2 \|C\|$	10	12	14	8	12	14
RKD	No. of trees	8	8	8	16	8	16
HKM	k	32	64	Not executable	16	64	32



(a) SIFT 1M (b) SIFT 10M (c) SIFT 100M

(d) GIST 100K (e) GIST 1M (f) GIST 10M

Figure 5: Experiment 1: Recall vs Computation time.

Acknowledgment

This work was supported by JST CREST project, JST Funds for integrated promotion of social system reform and research and development, JSPS KAKENHI #25240028, and JST A-STEP #AS242Z02205H.

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proc. FOCS*, 2006. 2

[2] A. Babenko and V. Lempitsky. The inverted multi-index. In *Proc. CVPR*, 2012. 1, 2, 3, 6

[3] J. Brandt. Transform coding for fast approximate nearest neighbor search in high dimensions. In *Proc. CVPR*, 2010. 2, 6

[4] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. 20th annual symposium on Computational geometry*, 2004. 2, 6

[5] A. Gersho and R. Gray. *Vector Quantization and Signal Compression*. Kluwer Acadmic Publishers, 1992. 3

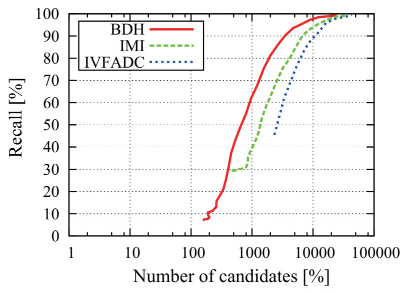
[6] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proc. CVPR*, 2011. 2

[7] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Trans. PAMI*, 2012. 2

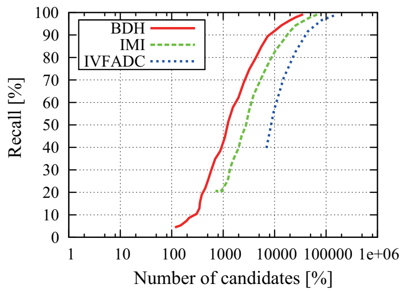
[8] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. In *Proc. 30th Symposium on Theory of Computing*, 1998. 2

[9] M. Iwamura, T. Kobayashi, and K. Kise. Recognition of multiple characters in a scene image using arrangement of local features. In *Proc. ICDAR*, 2011. 1

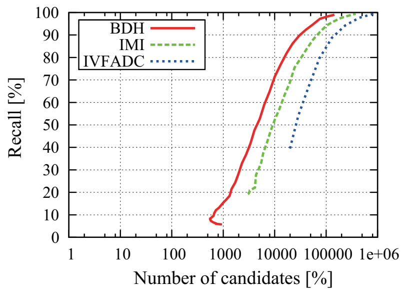
[10] M. Iwamura, T. Tsuji, and K. Kise. Memory-based recognition of camera-captured characters. In *Proc. DAS*, 2010.



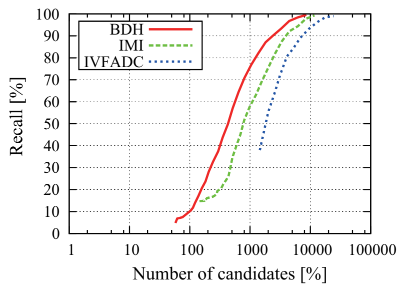
(a) SIFT 1M



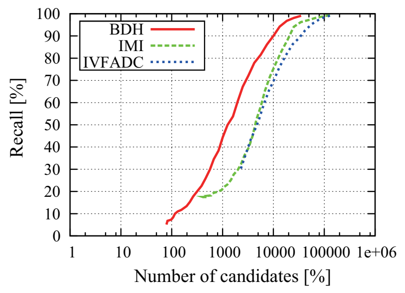
(b) SIFT 10M



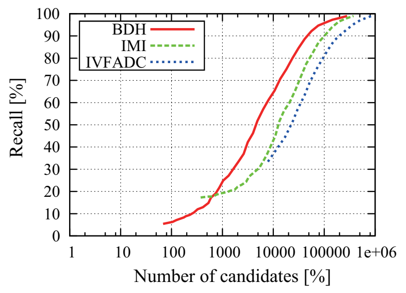
(c) SIFT 100M



(d) GIST 100K



(e) GIST 1M



(f) GIST 10M

Figure 6: Experiment 2: Recall vs Number of candidates.

1

[11] H. Jégou, M. Douze, and C. Schmid. Packing bag-of-features. In *Proc. ICCV*, 2009. 1

[12] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. PAMI*, 33(1):117 –128, 2011. 1, 2, 6

[13] Y. Ke, R. Sukthankar, and L. Huston. Efficient near-duplicate detection and sub-image retrieval. In *Proc. ACM Multimedia*, 2004. 1

[14] K. Kise, K. Noguchi, and M. Iwamura. Robust and efficient recognition of low-quality images by cascaded recognizers with massive local features. In *Proc. WS-LAVD*, 2009. 1

[15] W. Kong and W.-J. Li. Isotropic hashing. In *Proc. NIPS*, 2012. 2

[16] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Proc. NIPS*, 2009. 2

[17] R.-S. Lin, D. A. Ross, and J. Yagnik. SPEC hashing: Similarity preserving algorithm for entropy-based coding. In *Proc. CVPR*, 2010. 2

[18] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2), 2004. 6

[19] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proc. VLDB*, 2007. 2

[20] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. VISSAPP*. INSTICC Press, 2009. 1, 2, 6

[21] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *Proc. CVPR*, 2006. 1, 2, 6

[22] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *Proc. ICML*, 2011. 2

[23] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 42, 2001. 6

[24] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *Proc. CVPR*, 2008. 2, 6

[25] K. Takeda, K. Kise, and M. Iwamura. Memory reduction for real-time document image retrieval with a 20 million pages database. In *Proc. CBDAR*, 2011. 1

[26] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large dataset for non-parametric object and scene recognition. *IEEE Trans. PAMI*, 30(11):1958–1970, 2008. 1

[27] E. Tuncel, H. Ferhatosmanoglu, and K. Rose. VQ-index: an index structure for similarity searching in multimedia databases. In *Proc. ACM Multimedia*, 2002. 2

[28] Y. Weiss, R. Fergus, and A. Torralba. Multidimensional spectral hashing. In *Proc. ECCV, Part V (LNCS 7576)*, 2012. 2

[29] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proc. NIPS*, 2008. 2, 6

[30] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu. Complementary hashing for approximate nearest neighbor search. In *Proc. ICCV*, 2011. 2

[31] L. Zhang, Y. Zhang, J. Tang, K. Lu, and Q. Tian. Binary code ranking with weighted hamming distance. In *Proc. CVPR*, 2013. 2