Multimedia Systems (2017) 23:139–153
DOI 10.1007/s00530-014-0444-3

SPECIAL ISSUE PAPER

Effective optimizations of cluster-based nearest neighbor search in high-dimensional space

Xiaokang Feng · Jiangtao Cui · Yingfan Liu · Hui Li

Published online: 24 December 2014
© Springer-Verlag Berlin Heidelberg 2014

Abstract Nearest neighbor (NN) search in high-dimensional space plays a fundamental role in large-scale image retrieval. It seeks efficient indexing and search techniques, both of which are simultaneously essential for similarity search and semantic analysis. However, in recent years, there has been a rare breakthrough. Achievement of current techniques for NN search is far from satisfactory, especially for exact NN search. A recently proposed method, HB, addresses the exact NN search efficiently in high-dimensional space. It benefits from cluster-based techniques which can generate more compact representation of the data set than other techniques by exploiting interdimensional correlations. However, HB suffers from huge cost for lower bound computations and provides no further pruning scheme for points in candidate clusters. In this paper, we extend the HB method to address exact NN search in correlated, high-dimensional vector data sets extracted from large-scale image database by introducing two new pruning/selection techniques and we call it HB+. The first approach aims at selecting more quickly the subset of hyperplanes/clusters that must be considered. The second technique prunes irrelevant points in the selected subset of clusters. Performed experiments show the improvement of HB+ with respect to HB in terms of efficiency (I/O cost and CPU response time) and also demonstrate the superiority over other exact NN indexes.

Keywords Image retrieval · Nearest neighbor search · High-dimensional indexing · Hyperplane bounds

X. Feng · J. Cui · Y. Liu · H. Li (✉)
School of Computer Science and Technology, Xidian University, Xi'an 710071, China
e-mail: hli@xidian.edu.cn

X. Feng
e-mail: xkfengxd227@gmail.com

J. Cui
e-mail: cuijt@xidian.edu.cn

Y. Liu
e-mail: yfliu1989@gmail.com

1 Introduction

Large-scale image retrieval plays an important role in both the database community and the computer vision community. Recently, researchers have diverted more attention to supervised (or weakly supervised) hashing methods due to the concern of semantic similarity. These methods enable users to specify query through a natural language description of the visual concepts of interest [1], which appears to be more precise in the semantic perspective. However, this does not cause the unsupervised similarity retrieval methods to be disfavored. In fact, unsupervised methods use just the unlabeled data to quantify the given points and usually show good performance with metric distances. The most typical paradigm of unsupervised similarity retrieval is the nearest neighbor (NN) search or k-nearest neighbor (k-NN) search problem. Figure 1 shows a brief introduction of image retrieval progress supported by NN search.

According to Fig. 1, we can see that image retrieval is composed of two processing stages. The first one is the image representation stage (the blue arrow part) during which we extract high-quality feature vectors to represent the original images with sufficient accuracy. It directly determines the accuracy of image retrieval. Images from the image database are represented by kinds of feature vectors which are usually correlated and high dimensional, such as color histogram, textures, shape, graphlet or multimodal features. All these extracted feature vectors form

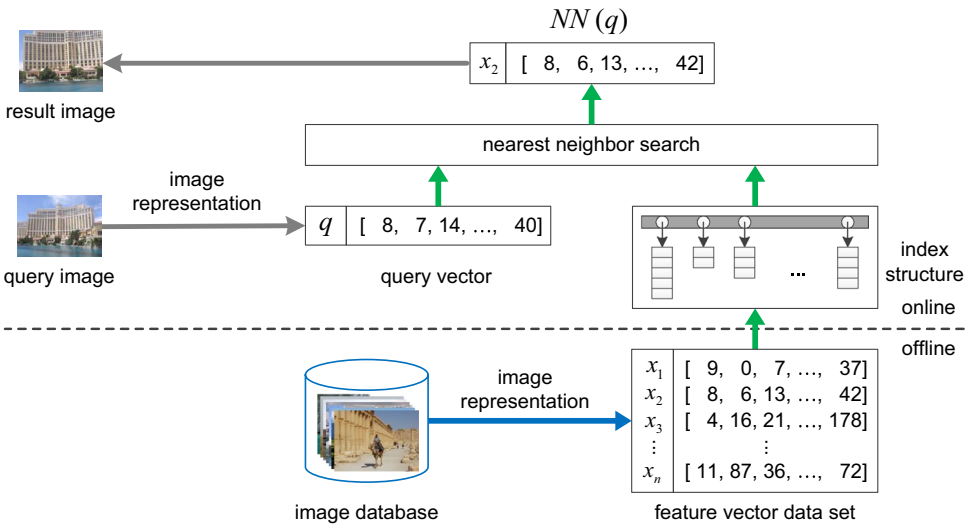


Fig. 1 Brief introduction of the image retrieval process (colour figure online)

a feature vector data set which will be handed over to the second processing stage (i.e. NN search, the green arrow parts in Fig. 1) to deal with. Therefore, image retrieval in large-scale image database is conveniently transformed into NN search in large-scale high-dimensional vector space. In other words, NN search is a fundamental paradigm of image retrieval. It supports image retrieval and directly affects the efficiency and accuracy of image retrieval.

We devote this paper to solving the NN search problem. The following is a formulated description of it:

$$NN(q) = \arg\min_{x \in D} \|q, x\|_p, \tag{1}$$

where q represents a query vector extracted from a query image, D represents the feature vector data set where $D \subset \mathbb{R}^d$ and $\|q, x\|_p$ is the p-norm distance between q and x. The goal of NN search is to find the closest point $NN(q)$ to the query from the data set.

It is noted that NN search in large-scale high-dimensional vector space usually consists of two procedures as depicted in Fig. 1. The first one is the off-line procedure, which seeks efficient high-dimensional index structures for the data set which will reduce expensive I/O operations. The other one is the online procedure, which seeks efficient searching methods that will cut down CPU computation tasks. Both of these are essential for NN search.

Research on NN search problem can be divided into exact retrieval and approximate retrieval. The approximate nearest neighbor (ANN) search problem has been adequately addressed. Hashing is demonstrated to be effective for similarity search in multimedia databases. Locality-sensitive hashing (LSH) is one typical hashing method [2]. Its basic idea is to use a family of locality-sensitive hashing functions based on random projections. Several heuristic variants of LSH have also been suggested [3–5]. Vector quantization, especially using product quantization and residual quantization, can also support efficient ANN search combined with an inverted file system [6].

In comparison, exact NN search has much slower progress due to the strict requirement of returning accurate nearest neighbors. This causes inevitable expensive distance computation on real data points, which makes I/O a major performance bottleneck in current storage system. Techniques that address this issue have to seek efficient index structure and search methods simultaneously, which brings considerably more difficulty to algorithm designing.

In this paper, we focus on the exact k-NN search algorithm and the corresponding index structures.[1] The conventional approach to support k-NN search in high-dimensional vector space is to use a multidimensional index structure [7]. However, designing an efficient index structure is a big challenge for researchers. Recently, the "filter-and-refine" index strategy has been proved to be an efficient scheme to solve NN search in correlated, high-dimensional feature space. To perform the "filter-and-refine" scheme, the data set is divided into partitions with space-based or data-based techniques in advance, and index structures for those partitions are then built accordingly. With the index structures, lower bounds of distances between the query point and all partitions can be achieved conveniently. Finally, only partitions with small enough lower bounds to the query are accessed, the remaining are safely pruned. Notably, the performance of "filter-and-refine" approaches depends on the tightness of the lower bounds. In other words, the tighter the lower bounds, the more will be the clusters pruned. Traditionally, lower

[1] In the sequel, we shall use "k-NN search" to refer to "exact k-NN search" for simplicity.

Table 1 Notations

Symbol	Notations
D	Data set
K	Number of clusters
$H(n,p)$	A hyperplane with parameters n and p
q	Typical query
$x, x_1, x_2, \ldots$	Typical elements of the data set
$\{C_i\}_{=1}^{K}$	K clusters of the data set
$\{c_i\}_{=1}^{K}$	K centroid of all clusters
H_{ij}	Hyperplane separating C_i and C_j
$d(x,H)$	Distance between a point x and a hyperplane H
$d(H,C)$	The minimum distance between a cluster C and one of its hyperplane bounds H
$LB(x,C)$	The lower bound distance between a point x and a cluster C
k	Number of nearest neighbors to return
$\|\cdot\|_2$	2-norm of a vector
$\|x_1, x_2\|_2$	Euclidean distance between x_1 and x_2

bounds are computed with bounding rectangles or spheres of the partitions [8–11]. Unfortunately, they do not produce sufficiently tight lower bounds for high-dimensional data and thus have to access the majority of the data set.

Cluster-based techniques are demonstrated to be very effective to produce tighter lower bounds, because they can exploit interdimensional correlations within data sets and generate more compact representations. Recently, Ramaswamy et al. [12] proposed a new method called HB (hyperplane bounds) based on these techniques. It generates lower bounds by exploiting separating hyperplanes between the query point and all the clusters. With tight enough lower bounds, HB filters out most of the unrelated clusters effectively and obtains a pretty good query performance. Besides, its index structure is space saving (requires only $O(Kd)$ space, where K and d denote the number of clusters and the dimensionality of the data set, respectively).

However, the tightness of lower bounds in HB is highly related to the number of clusters K and more clusters directly help to produce tighter lower bounds. Unfortunately, the CPU cost for lower bound computation increases quadratically with the growth of K, since its time complexity is $O(K^2 d)$. Besides, HB does not provide any further pruning method inside candidate clusters. All points in a candidate cluster will be verified by expensive distance computation, which will decrease the performance of the search algorithm dramatically.

In this paper, we focus on addressing the two drawbacks of HB above and propose our new index structure called HB+, which can reduce nearly 20 % I/O cost and more than 30 % CPU cost on average compared to HB. Our extensive experiments on three visual feature data sets demonstrate the superiority of HB+ over HB and three other popular indexing methods in the database community including VA-File [7], iDistance [13] and a newly proposed method FNN [14].

2 Preliminaries of HB

Our solutions will leverage on HB as the building brick. In this section, we first present an important technique of HB that is necessary for our discussion. Then we discuss the limitations of HB to further clarify the motivation of our methods. A list of notations that we will use subsequently is presented in Table 1.

2.1 Hyperplane and lower bound

We use Euclidean distance as the similarity measure between points in this paper. In d-dimensional space, a hyperplane can be represented as

$$H(n,p) = \left\{ x^T n + p = 0 \mid x \in \mathbb{R}^d \right\}. \tag{2}$$

Here, n is the normal vector of the hyperplane and p is just a real number. For simplicity, we use H to denote a hyperplane in the rest of this paper. The distance between a point x and a hyperplane H can be obtained as follows:

$$d(x,H) = \left| \frac{x^T n + p}{\|n\|_2} \right|. \tag{3}$$

We are interested in separating hyperplanes (i.e. hyperplanes lie between the query and clusters) because only separating hyperplanes generate lower bound. As shown in Fig. 2, we can obtain a lower bound distance between the query, namely q, and a cluster, namely C_i by adding $d(q,H)$, the distance between q and a separating hyperplane H, and

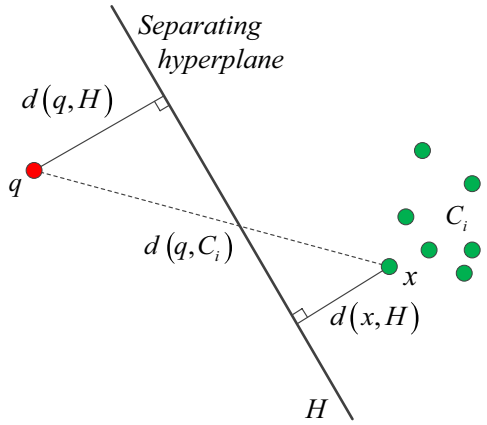


Fig. 2 Lower bound with separating hyperplanes

$d(x, H)$, the minimum of distance between the points in the cluster and H.

We treat the lower bound[2] computing between the query and the cluster as a kind of competition among all the separating hyperplanes. The formal definition is as follows:

$$LB(q, C) = \max_{H \in H_{\text{sep}}} \left\{ d(q, H) + \min_{x \in C} d(x, H) \right\}. \tag{4}$$

A loosening measurement for Eq. 4 is taken in [12] to achieve a tighter lower bound. The following is the final expression of lower bound computing:

$$LB(q, C) = \max_{H \in H_{\text{sep}}} d(q, H) + \min_{x \in C} \min_{H \in H_c} d(x, H), \tag{5}$$

where H_{sep} is a set of separating hyperplanes between q and C and H_c represents all the hyperplane bounds around C . For Eq. 5, the first element to the right side of the equal mark [i.e. $\max_{H \in H_{\text{sep}}} d(q, H)$] represents the maximum distance between q and all the separating hyperplanes of C. It is calculated online. The second one to the right side of the equal mark represents the minimum distance between all data points in C and all C's hyperplane bounds. It seems like a "inner gap" between the whole data points inside C and all its boundaries. We define a new notation to represent it:

$$IG_c = \min_{x \in C} \min_{H \in H_c} d(x, H), \tag{6}$$

where IG_c is short for "inner gap w.r.t a cluster". Besides, it is obvious that IG_c for each cluster can be computed offline and stored in advance with only $O(Kd)$ space. Therefore, the main cost of lower bound computing resides only in the computation of the distances between the query and all the separating hyperplanes.

[2] In the sequel, we shall use the term "lower bound" to refer to the lower bound distance between a query and a cluster for simplicity.

2.2 Refining separating hyperplanes

In this subsection, we discuss how to select the set of separating hyperplanes between the query and a cluster, which is crucial to the tightness of the lower bound according to Eq. 5. Firstly, we assume that a data set is partitioned into K clusters, which are represented as $\{C_i\}_{=1}^{K}$. The centroid of these clusters can be denoted as $\{c_i\}_{=1}^{K}$.

As defined in [12], the hyperplane H_{ij} between two clusters C_i, C_j ($1 \le i, j \le K$ and $i \ne j$) are defined as follows:

$$H_{ij} = H(-2(c_i - c_j), \|c_i\|_2^2 - \|c_j\|_2^2). \tag{7}$$

Hence, for a cluster, there are $K - 1$ hyperplanes around it. However, not all of them are separating hyperplanes for a specific query, since a separating hyperplane just locates between the query and the cluster. Thus, the set of separating hyperplanes for a query q and a cluster C_i is defined as follows:

$$H_{\text{sep}}(q, C_i) = \left\{ H_{ij} \middle| \|q, c_i\|_2 > \|q, c_j\|_2 \right\}. \tag{8}$$

Equation 8 enables us to pick out the real separating hyperplanes from all hyperplane bounds for a cluster. This refinement of separating hyperplanes works as follows. Let $d_{qc}[\cdot]$ store the distances between query q and all K clusters. We can figure out a sequence of these clusters $o_c[\cdot]$ by sorting all the K distances in ascending order. Thus, according to Eq. 8, for a cluster C_i, it may get a corresponding order number j ($1 \le j \le K$) in $o_c[\cdot]$ where $i = o_c[j]$, and only clusters with order numbers smaller than j will form separating hyperplanes for cluster C_i. The following is an observation about the number of separating hyperplanes.

Observation 1 The average number of separating hyperplanes for all clusters in HB is $(K - 1)/2$, approximately 0.5 K in high-dimensional space.

Proof Based on the orders of all the K clusters $o_c[\cdot]$, the numbers of the corresponding separating hyperplanes for each cluster are $0, 1, 2, \ldots, K - 1$, respectively, which means the cluster with the farthest centroid from the query gets $K - 1$ separating hyperplanes, while the closest one gets no separating hyperplane. Therefore, to obtain all the K lower bounds, there are totally $K(K - 1)/2$ distances between the query and separating hyperplanes that have to be computed. On average, each cluster gets $(K - 1)/2$ separating hyperplanes, nearly 0.5 K in high-dimensional space.

2.3 Limitations of HB

The k-NN search algorithm performed by HB in the Algorithm 2 in [12] follows the "filter-and-refine" strategy. For every query, the hyperplane bounds are calculated in

advance. These lower bounds are then used to sort all the clusters. Clusters are accessed in ascending order, respectively, and in the meanwhile k-NNs within each cluster are identified. When the k-th nearest neighbor's distance found so far is smaller than the lower bound of the next cluster, the search terminates as all k-NNs have been found.

We can see that HB is theoretically efficient. However, there are two limitations within the algorithm, too many query–hyperplane distance computations and no effective approach to prune false positives in the candidate clusters.

To reduce the storage cost, HB only stores the minimum of the distances between each cluster and all the hyperplanes around it. All the lower bounds between the query and clusters need to be calculated online, which leads to exhaustive computations with $O(K^2d)$ time complexity of high-dimensional distance according to Sect. 2.2. Especially with an excessively large K, which is necessary for high filtration rate, the computation cost of lower bounds becomes very high and even exceeds the cost of distance computation. Conversely, if K is too small, the filtering ability of HB will decrease and more candidate points need to be accessed and computed.

Besides, HB only provides a novel bounding technique to filter irrelative clusters. However, it takes a simple method to deal with the false positives inside the candidate clusters. It loads all members of a candidate cluster into memory to perform expensive distance computations to identify k nearest neighbors. Unfortunately, as there may exist numerous false positives in the candidate cluster, this approach may lead to considerable unnecessary I/O cost and CPU cost.

3 HB+

To address the two limitations of the above HB method, we propose our new approach, namely HB+, which will be described in detail in this section.

3.1 Accelerate lower bound computing

Lower bound depends on separating hyperplanes. As mentioned in Sect. 2.2, to achieve tight lower bounds, HB takes a large number of distance computing between the query and separating hyperplanes. We can figure out the equation for query–hyperplane distance computing by combining Eqs. 3, 7 and 8:

$$d(q, H_{ij}) = \frac{\|q, c_i\|_2^2 - \|q, c_j\|_2^2}{2\|c_i, c_j\|_2}. \tag{9}$$

We can see that it constitutes two kinds of distance computing. Both are very expensive in high-dimensional space. However, according to Eq. 5, the lower bound is actually determined by only one separating hyperplane. This means that most of the separating hyperplanes are superfluous, but inevitable. We propose a novel way to accelerate the lower bound computing from two aspects as follows: reducing the amount of calculation in Eq. 9 as well as partially selecting separating hyperplanes.

For the first aspect, as mentioned above, there are two computing tasks in Eq. 9 the distances computing between q and K cluster centroids and the distance computing between each pair of centroids. The former one contains K distance computing, which should be obtained online in advance, since they cost few response times and each of them will be used several times during the process of computing the K lower bounds. As for the latter one (i.e. $\|c_i, c_j\|_2$), there are totally $K(K-1)/2$ distances. To save storage as HB always promotes, we plan to solve this problem online. To solve this issue, we introduce random projection techniques [15–17] which are commonly used for fast estimation of distance between points in machine learning community.

Formally, we represent the set of K cluster centroids as a $K \times d$ matrix A and a random matrix R with size $d \times m$ which is generated to perform the dimension reduction as follows:

$$B = \frac{1}{\sqrt{m}} AR. \tag{10}$$

According to this equation, a d-dimensional point is transformed to an m-dimensional point. Besides, each element r_{ij} of R is independently randomly drawn from the following distribution [15]:

$$r_{ij} = \begin{cases} \sqrt{3} & \textit{with prob. } 1/6 \\ 0 & \textit{with prob. } 2/3 \\ -\sqrt{3} & \textit{with prob. } 1/6 \end{cases} \tag{11}$$

With dimension reduction, we can estimate the distances between each pair of centroids with their corresponding reduced ones, thus leading to an estimation for Eq. 9. It is noted that m is an important parameter for HB+ since it affects the tightness of lower bounds and the efficiency of the estimation simultaneously. As m increases, more information will be retained in the points after dimension reduction and the estimation of distances between cluster centers will be more precise. However, this will also introduce more CPU cost during the estimation. Actually, m is usually a small value, much smaller than the dimensionality of the data set d. We will discuss the choice of m in Sect. 4.2.

For the second aspect, instead of selecting all possible hyperplanes obtained by Eq. 8, we can smartly choose only T separating hyperplanes that hold the largest T estimates of distance from the query. Because of the well similarity preserving ability, the random projection techniques we performed above can ensure high probability that the separating hyperplane which provides the maximum distance to the query point is retained in the T selected ones. Let $\alpha = \frac{T}{K}$ be

the proportion of selected hyperplanes; hence, $0 < \alpha < 1$. More precisely, the similarity preserving ability of the random projection techniques is well enough that α is usually much smaller than 0.5 (the average number of separating hyperplanes for each cluster in HB is almost 0.5 K according to Observation 1) according to the experimental result. Actually, T and α are derived from the same concept.[3] Based on this, the cardinality of the set of separating hyperplanes for a query and a cluster is reduced significantly, which helps to accelerate the computation of lower bounds further. A fast algorithm to compute the lower bound with our aforementioned acceleration method is outlined in Algorithm 1.

Algorithm 1 LowerBound

Require: $\{c_i\}_{=1}^{K}$, q, α, K, $IG_c[\cdot]$

Ensure: $LB[\cdot]$

1: $d_{qc}[\cdot] \leftarrow \text{dist}(q,\{c_i\}_{=1}^{K})$;
2: $\{d_{qc}^{sort}[\cdot], o[\cdot]\} \leftarrow \text{sort}(d_{qc}[\cdot], \text{'ascend'})$;
3: $T \leftarrow \alpha \cdot K$;
4: $LB[o[1]] \leftarrow 0$;
5: **for** $i \leftarrow 2 : K$ **do**
6: $\quad H_{sep}[\cdot] \leftarrow \phi$
7: $\quad i_c \leftarrow o[i]$;
8: $\quad$ **for** $j \leftarrow 1 : i-1$ **do**
9: $\quad\quad$ add $H_{i_c o[j]}$ to $H_{sep}[\cdot]$;
10: $\quad$ **end for**
11: $\quad$ **if** $i-1 > T$ **then**
12: $\quad\quad d_{qH}[\cdot] \leftarrow \text{estimate\_dist}(q, H_{sep}[\cdot])$;
13: $\quad\quad \{d_{qH}^{sort}[\cdot], o_H[\cdot]\} \leftarrow \text{sort}(d_{qH}[\cdot], \text{'descend'})$;
14: $\quad\quad H_{sep}[\cdot] \leftarrow H_{sep}[o_H[1:T]]$;
15: $\quad$ **end if**
16: $\quad LB[i_c] \leftarrow \max_{H \in H_{sep}[\cdot]} d(q,H) + IG_c[i_c]$;
17: **end for**
18: **return** $LB[\cdot]$;

In Algorithm 1, distances between the query point q and all cluster centroids will be computed first and kept in a list $d_{qc}[\cdot]$ (line 1). Then they are sorted in ascending order (line 2) to generate an ordered list $o[\cdot]$ of these clusters. It is obvious that there is no separating hyperplane around the first cluster in $o[\cdot]$ whose centroid is closest to q. We just set its lower bound to be 0 (line 4). For the other clusters, C_{i_c} ($i_c \neq o[1]$) as an example, we can also obtain the separating hyperplanes easily with the help of $o[\cdot]$ according to Eq. 8, because all the separating hyperplanes of C_{i_c} must lie between q and C_{i_c}. This means a cluster (C_{i_c} except) that wants to form a separating hyperplane of C_{i_c} must satisfy the condition that the distance between q and itself should be smaller than that between q and C_{i_c} according to Eq. 8. $o[1:i-1]$ is the set of cluster IDs that satisfy this condition, where i is the rank number of C_{i_c} in the ordered list $o[\cdot]$. Thus, for a cluster whose rank number in $o[\cdot]$ is i ($1 \leq i \leq K$), the number of separating hyperplanes around it will be $i-1$. According to this, the farther the centroid of a cluster away from q, the more separating hyperplanes the cluster gets. Afterwards, we select at most T separating hyperplanes for each cluster to perform the precisely, but expensively lower bound computing in Line 16. According to the above, clusters with rank numbers smaller than $T+1$ can acquire a direct promotion to Line 16. For the other clusters that hold more than T separating hyperplane bounds, we perform an estimation of the distances between the query and the separating hyperplanes for each cluster with the help of random projection as mentioned before (line 12). Then we sort these estimates of distance in descending order and choose the most promising T ones as the final candidate separating hyperplanes (line 13 to line 14) of one cluster. In Line 16, with carefully chosen separating hyperplanes, we obtain the lower bound for each cluster. Here, as mentioned in Sect. 2.1, IG_c for each cluster is computed off-line and stored in advance. It consists of all the minimum distances between each cluster and the hyperplanes around it. Finally, the lower bounds of all clusters will be returned.

3.2 Efficient pruning in candidate clusters

We have discussed in Sect. 2.3 that the search algorithm of HB suffers from checking all data points in a candidate cluster which may contain lots of false positives and thus lead to a bad search performance. Figure 3 illustrates an example. Suppose the current search radius is r. In the ideal case, there is only one point x_1 that needs to be checked; thus, it is unnecessary to access all the points in this cluster for the query.

The reason that HB cannot avoid such false positives relies on the granularity of the "inner gap" concept used in HB. In other words, IG_c is able to tell us whether a cluster should be loaded into the main memory or not during the k -NN search, because it is distinguishable on the granularity of a cluster. However, it is unable to determine whether a certain data point should be loaded.

To prune data points inside the candidate clusters in HB+, we extend the "inner gap" concept to the granularity of a data point and introduce a new notation IG_p ("inner gap w.r.t a point") accordingly. IG_p is defined as:

$$IG_p(x) = \min_{H \in H_c} d(x,H), \tag{12}$$

[3] This concept refers to our suggestion of partially selecting separating hyperplanes. In fact, in experimentation we have tested the empirical results by varying the value of α, the proportion of selected separating hyperplanes, during the experiments.

where x denotes any point in cluster C, and H_c represents all the $K-1$ hyperplane bounds around C as mentioned before. Therefore, the minimum distances between each point and all the hyperplanes should be obtained because

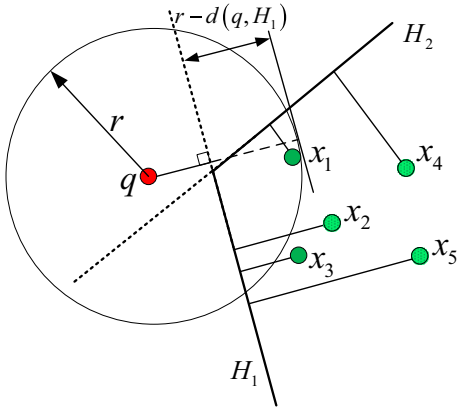


Fig. 3 An illustration of a cluster containing false positives

IG_p is related to every data point. As shown in Fig. 3, the length of the solid line connecting the data point and its nearest hyperplane boundary is equal to $IG_p(x)$. Utilizing this variable, we can further get a tighter lower bound distance between the query point q and a data point x as follows:

$$LB(q,x) = \max_{H \in H_{\text{sep}}} d(q,H) + IG_p(x) \leq \|q,x\|_2. \tag{13}$$

All the data points of a cluster are stored and accessed in ascending order of IG_p. During the query process, a point to be accessed can be rejected if its lower bound to the query is larger than the search radius. In this way, if we find a point that can be pruned, all the rest points in this cluster can be pruned too. Recall the example shown in Fig. 3. It is noted that the data points sequence of this cluster is x_1, x_3, x_2, x_4, x_5 and the maximum distance between the query and separating hyperplanes is $d(q,H_1)$. According to the above, since $IG_p(x_2)$ is larger than $r-d(q,H_1)$, x_2 can be pruned safely without distance computation. Consequently, x_4 and x_5 can also be rejected.

Algorithm 2 k-NNSearch

Require: $\{c_i\}_{=1}^{K}, IG_c[\cdot], IG_p[\cdot], \alpha, k, q$

Ensure: $kNNs[\cdot]$ // $kNNs$ is a max-heap holding at most k elements, each element of it is a $\langle point, distance \rangle$ pair and it is maintained according to the $distance$ value

1: set each element of $kNNs[\cdot]$ to be $\langle NULL, MAXREAL \rangle$, set $kNNs.count$ to be 0;
2: $LB[\cdot] \leftarrow$ lowerbound($\{c_i\}_{=1}^{K}, q, \alpha, K, IG_c[\cdot]$);
3: $\{LB^{sort}[\cdot], o[\cdot]\} \leftarrow$ sort(LB,'ascend');
4: $i \leftarrow 1$;
5: **while** $i \leq K$ and $LB^{sort}[i] < kNNs[1].distance$ **do**
6: $\quad i_c \leftarrow o[i]$;
7: $\quad Candidate_p[\cdot] \leftarrow \phi$;
8: $\quad$ **for** point x in the i_c-th cluster **do**
9: $\quad\quad$ **if** $kNNs[1].distance \geq LB[i_c] - IG_c[i_c] + IG_p(x)$ **then**
10: $\quad\quad\quad$ load x from the disk and add it to $Candidate_p[\cdot]$;
11: $\quad\quad$ **else**
12: $\quad\quad\quad$ break;
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: $\quad$ **for** point x in $Candidate_p[\cdot]$ **do**
16: $\quad\quad$ **if** $kNNs.count < k$ or $\|q,x\|_2 < kNNs[1].distance$ **then**
17: $\quad\quad\quad$ max-heap-insert $\langle x, \|q,x\|_2 \rangle$ into $kNNs[\cdot]$ and ensure that $kNNs[\cdot]$ holding at most k elements;
18: $\quad\quad$ **end if**
19: $\quad$ **end for**
20: $\quad i++$;
21: **end while**
22: **return** $kNNs[\cdot]$;

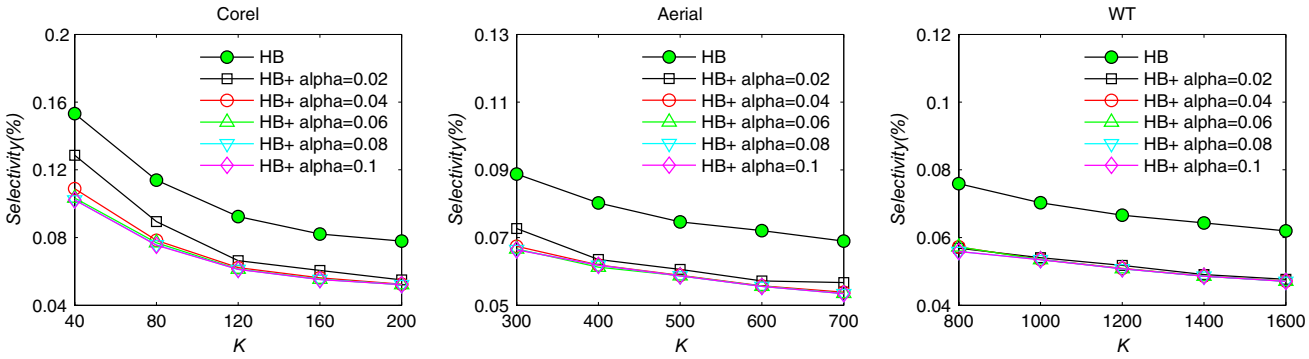


Fig. 4 Comparison of selectivity between HB and HB+

Based on this, we propose our new pruning algorithm. More details are depicted in Algorithm 2. It is noted that the process of computing IG_p is performed offline, which does not add to the computation cost during the query. Besides, only 1-dimensional value is added when storing data points, and no additional index is used to manage these points in a cluster.

Moreover, the index to manage data points for a cluster of iDistance is based on distance computation [13]. The one-dimensional distances are indexed with a B+-tree. During the search, it locates the first point to be accessed in a cluster and then a bidirectional scan will be conducted. However in HB+, the search in a cluster is a one-directional linear scan, which can save more I/O cost, because the linear scan avoids seeking the specified page and is much faster than the bidirectional scan of data pages.

Figure 4 illustrates the effectiveness of the pruning operation. We implement 10-NN search on three high-dimensional data sets, Corel, Aerial and WT (specific details of these data sets can be found in Sect. 4). We compare the selectivity between HB and HB+ where the selectivity of an approach is defined as the number of points that are accessed during the search with respect to the cardinality of the data set. As mentioned above, we tune both the proportion of selected separating hyperplanes α and the cluster number K to see the performance trend.

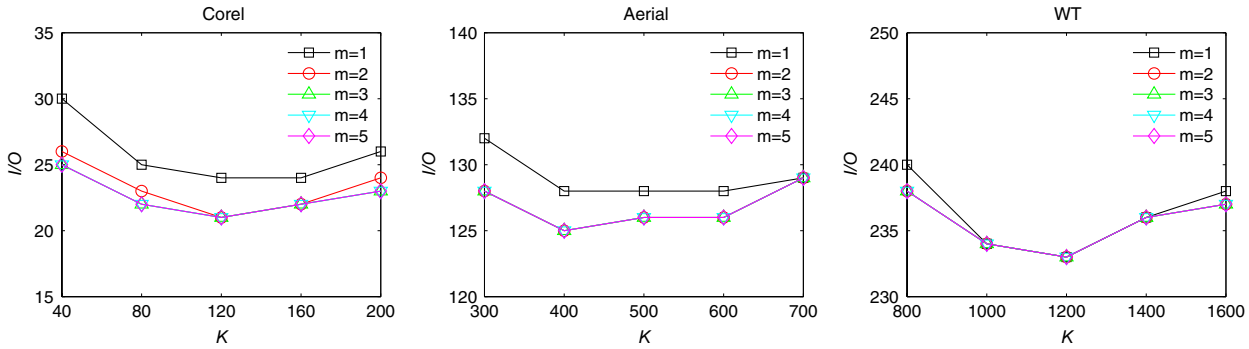
According to Fig. 4, HB+ obviously reduces the number of points accessed compared to HB, which means our new pruning algorithm has made a contribution. Besides, when α is larger than 0.06, increasing α will not provide further pruning power, which means more separating hyperplane cannot enhance the tightness of the lower bounds any more. This indicates a critical point of the amount of selected separating hyperplanes and provides us with a principle to select a proper α for a certain data set.

4 Experiments

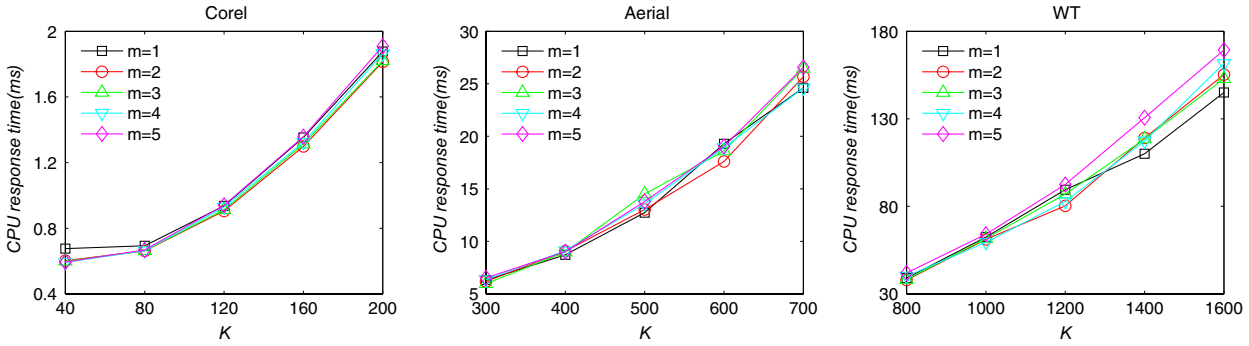
In this section, we experimentally evaluate the performance of our optimizations in HB+. Section 4.1 introduces the metrics for performance evaluating and the details of data sets. Section 4.2 tunes the parameter m for HB+ to observe the influence on the tightness of lower bounds. It offers us a standard for parameter settings of m in HB+. Section 4.3 demonstrates the superiority of HB+ over HB. Finally, in Sect. 4.4, we compare the methods based on hyperplane bounds with popular index schema such as VA-File, iDistance and a newly proposed method FNN in their best performance. Implementations are achieved in C. All experiments are performed on a Linux machine with an Intel® Core™2 Quad CPU 2.83 GHz processor and 4GB RAM running Ubuntu 12.04 LTS.

4.1 Experimental setup

For k-NN search, the most important performance we care about is the efficiency. In this paper, we measure efficiency in two axes: I/O cost and CPU response time. Since our algorithms and index structures are all about external memory where I/O communication between fast internal memory and slower external memory is the major performance bottleneck [18], we treat I/O cost as the major element of performance. For I/O cost, since there are both random I/O operations and sequential I/O operations in the comparing approaches, we compute I/O cost as $IO_r + IO_s/10$ [19], where IO_r is the number of random I/O and IO_s refers to the number of pages sequentially accessed. As for CPU response time, we remove I/O time from it, which means the CPU response time includes time cost of all operations when performing k-NN search, but without I/O operations.



(a) I/O versus m and K



(b) CPU response time versus m and K

Fig. 5 Performance of HB+ by tuning m on different data sets

The experiments are conducted on three representative visual feature data sets: Corel,[4] Aerial[5] and WT[6] Corel contains 68,040 32-dimensional color histograms. Aerial consists of 275,465 60-dimensional texture feature vectors. WT consists of 269,648 128-dimensional wavelet textures with each row representing an image. All of them are extracted from large-scale image databases for image representation and are commonly used as benchmarks for high-dimensional index methods. For each data set, we randomly select 200 data points to form the query set to estimate the performance of different approaches and the results are averaged.

4.2 Tuning m

Firstly, we tune the reduced dimensionality m in HB+, which directly affects the tightness of lower bounds. To observe an obvious performance trend, we set a group of values for m and K. Meanwhile, we fix the number k of returned neighbors as 10 in the following experiments in default, unless specified. As for α, which indicates the cardinality of the set of separating hyperplanes for each lower bound computation, we set it to be 0.06 initially which is a reasonable value. Figure 5 depicts the effect of m for 10-NN search performance on all three data sets.

As shown in Fig. 5a, when m is small ($m = 1$), HB+ consumes significantly more I/Os for different settings of K. However, after a slight increase, the I/O cost of HB+ reaches an optimal value and remains constant during further growth of m. It means that a small value of m (m around 3 according to the experimental performance) is the critical value which is able to produce the best I/O cost for HB+, and increasing m will not enhance the tightness of HB+ further. Naturally, we set m to such a critical point. Actually, the critical value of m is usually far less than d which will sufficiently reduce the computation cost of the denominator of Eq. 9. This is the reason that we insist placing the computation of distances between centroids online in Sect. 3.1. According to the actual results, we set m as 3 for Corel, 2 for Aerial and 2 for WT in the following experiments unless specified. As for CPU response time, the performance lines nearly stick together with each other, which means m may not influence response time obviously. This is easy to understand because the adjustment of m can only

[4] Download from http://archive.ics.uci.edu/ml/datasets/Corel+Image+Features/.

[5] Download from http://vision.ece.ucsb.edu/download.html.

[6] Download from http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm.

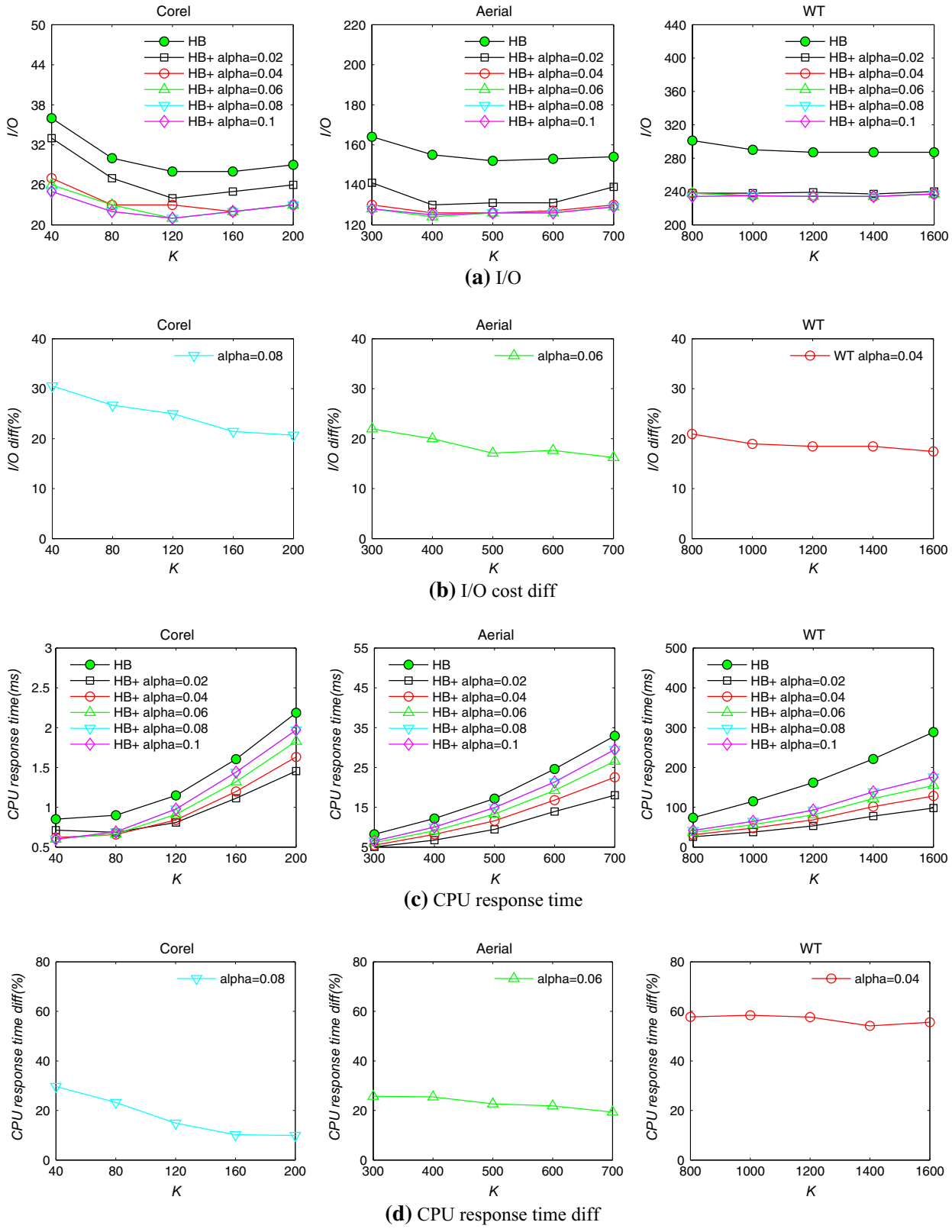


Fig. 6 Comparisons of I/O and CPU performance between HB and HB+

Table 2 Parameters setting for different methods on different data sets

Methods	Parameters setting for data set		
	Corel	Aerial	WT
HB	$K = 120$	$K = 500$	$K = 1{,}200$
HB+	$K = 120, m = 3, \alpha = 0.08$	$K = 500, m = 2, \alpha = 0.06$	$K = 1{,}200, m = 2, \alpha = 0.04$
iDistance	$N_{\text{ref}} = 64$	$N_{\text{ref}} = 64$	$N_{\text{ref}} = 64$
VA-File	$N_{\text{bit}} = 8$	$N_{\text{bit}} = 8$	$N_{\text{bit}} = 8$
FNN	None	None	None

decrease computation of the denominator of Eq. 9 quantitatively, but not change the complexity. In fact, this is just a preliminary operation for CPU cost saving.

4.3 Performance comparisons between HB and HB+

In this subsection, we compare HB+ with HB of both I/O performance and CPU performance. We tune K and α for HB+, which indicate the number of clusters and how many hyperplanes are selected for each cluster, respectively. As for HB, we observe the effects with various K.

Figure 6 depicts the comparisons of both I/O and CPU performance between HB and HB+ in all three data sets. Figure 6a shows the I/O cost comparison between HB and HB+ based on various α and K. We can see that HB+ significantly outperforms HB in I/O performance when $\alpha \geq 0.06$ in all data sets. It indicates that the pruning in candidate clusters is necessary and can promote I/O performance evidently. Besides, each data set encounters a threshold of α beyond which the performance of HB+ changes slightly. This is because increasing α cannot further prune points in candidate clusters when α is large enough. In other words, these small values of α in HB+ can be used to reach the same precision of estimating lower bounds as that by HB. This reflects that our estimation of lower bounds is effective. Hence, we can determine a proper α for each data set. In Fig. 6b, we set α to be 0.08, 0.06 and 0.04 on Corel, Aerial and WT, respectively, and observe specific decreasing quantity of I/O cost for HB+ against HB. We find that HB+ decreases 24.9, 18.6 and 18.9 % I/O operations compared to HB on Corel, Aerial and WT, respectively, which embodies a considerable improvement.

Figure 6c depicts the CPU performance between HB and HB+ with various α and K. We note that α significantly affects CPU performance, and smaller α initiates lower CPU response time. However, since we consider I/O cost to be the major element of performance, we observe specific decreasing degree of CPU response time according to proper α for each data set selected in the I/O cost observation. Obviously, HB+ saves 17.6, 23.0 and 56.7 % of CPU response time compared to HB on Corel, Aerial and WT, respectively. These three degrees reflect an increasing trend of time saving with a decline of α. The promotion of CPU performance indicates that careful selection of separating hyperplanes is aimed at the right direction.

To sum up, our pruning and accelerating measures (including dimension reduction with parameter m and separating hyperplane selection with parameter α) enable HB+ to outperform HB in both I/O and CPU performance.

4.4 Performance comparisons with other popular k-NN methods

In this subsection, we compare the approaches based on hyperplane bounds with other commonly used k-NN methods such as VA-File, iDistance and a newly proposed method FNN in performance. There are various parameters for different methods. HB depends on the number of clusters K. For HB+, it will tune the cluster number K, dimension reduction variable m and the coefficient of separating hyperplanes selecting α. iDistance mainly depends on the number of reference points N_{ref} and VA-File versus the approximate bit length in each dimension N_{bit}. For FNN, according to Algorithm 2 in [14], there is no need for parameter tuning. In our experiments, we set proper values for different parameters in different methods to get the best performance of each method. We reiterate that the major standard for parameter setting is the I/O performance. Table 2 shows the details of parameter setting. Based on the coefficient setting, we perform 10, 40-NN, 70-NN and 100 -NN search of different methods on all three data sets and evaluate both I/O and CPU performance. The corresponding experimental results are depicted in Fig. 7.

According to Fig. 7, it is apparent that approaches based on hyperplane bounds are far more advantageous in I/O cost over the famous two k-NN methods VA-File and iDistance. Besides, HB+ also achieves successfully both the best I/O performance and the best CPU performance in all three data sets. This demonstrates the outstanding ability of filtering irrelevant data points of the hyperplane bound. As for FNN, it shows a fairly good performance in CPU response time and an amazing performance in I/O cost when k is small. However, it does not preserve a good scalability with the number of returned nearest neighbors k. Actually, the I/O cost of FNN increases linearly with the growth of k. In this regard, HB+ maintains both I/O cost

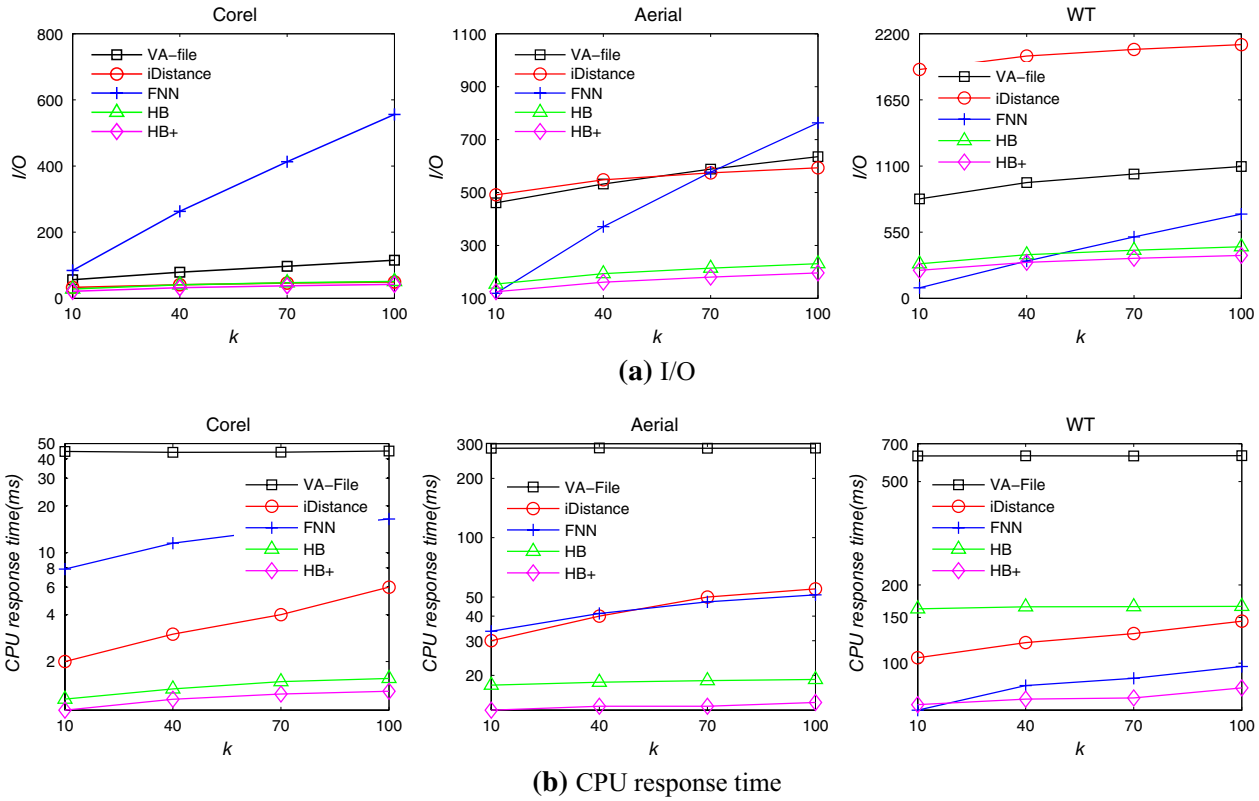


Fig. 7 Performance comparisons between HB+, HB with VA-File, iDistance and FNN

and CPU response time at a stable low amount when k increases. Hence, we demonstrate the superiority of HB+ over the other four approaches.

5 Related work

It can be seen from Fig. 1 that methods of different processing stages of image retrieval carry out different missions. NN search methods focus on enhancing the efficiency of dealing with the large-scale multi-dimensional feature vector data set to answer online image queries from the user efficiently. On the other hand, there exist a series of image representation methods which seek better representations of the original image to improve the accuracy of image retrieval.

5.1 k-NN search in large-scale high-dimensional data set

k-NN search is well understood in low-dimensional space [20]. In early studies, lots of tree-based indexing structures based on data partitioning [8–11, 21] or space partitioning [22, 23] have been developed. However, recent studies [7, 20] show that these tree-based indexing methods will, as dimensionality increases, always degenerate to a sequential scan or be eventually outperformed by sequential scan. To tackle this phenomenon named "curse of dimensionality", more approaches were proposed, mainly in two directions. One is to improve the index structure with efficient organization to promote IO operations. For exact k-NN search in high-dimensional space, there mainly exist two categories of index structure improving methods, data size reduction and dimensionality reduction. The other is vector approximation. Popular approaches usually combine these two methods to simultaneously improve the IO and CPU performance.

VA-File is the representative high-dimensional index for data size reduction. It suggests accelerating sequential scan by data compression and filtering feature vectors. The approximation of high-dimensional vector can be seen as a scale quantization. VA-File divides the data space into 2^b hyper-rectangular cells. It allocates a unique bit-string of length b to each cell, and approximates the data points that fall into a cell by the relevant bit-string w.r.t the cell. VA-File is simply an array of the compact approximations which may help to reduce the I/O cost during the query. One of the drawbacks of VA-File is that it needs to examine the entire VA-File. The other one is that it will bring a large number of random I/O accesses. Some extensions of VA-File have been proposed, such as

IQ-tree [24], which achieves better query performance by combining a tree structure with VA-File. VA+-file improves the approximate ability of VA-File by transforming the data points into PCA space [25]. Another technique PCVA devotes to reducing the amount of candidate approximate vectors that need to be accessed by sorting one-dimensional projections on the first principal component [26].

Dimensionality reduction (DR) indexes the data set in the reduced-dimensionality space [27, 28]. The linear DR approach first condenses most of information in a data set to a few dimensions by applying principal component analysis or other techniques. Two strategies for dimensionality reduction have been presented [19]. The first one is global dimensionality reduction (GDR), in which all the data are regarded as a whole and reduced down to a few dimensions. The other strategy, called local dimensionality reduction (LDR), is a cluster-based DR technique which divides the whole data set into separate clusters on the basis of the correlation of data, and then reduces the dimensionality in each cluster. To improve the performance of indexing method which requires pre-computing distances, a general cost model of LDR for range queries was presented [29]. According to this cost model, a new LDR method, called PRANS, has been proposed. One of the goals of the model is to prune more irrelative data points using triangle inequality.

One-dimensional indexing approaches are also typical dimensionality reduction methods. They provide another direction for high-dimensional indexing, which is one-dimensional mapping or transformation [10, 30]. The one-dimensional mapping methods also use the "filter-and-refine" strategy. Data points can be filtered according to the one-dimensional values, and the real nearest neighbors are verified in the set of candidates. B+-tree is usually deployed to index the transformed one-dimensional values [31]. A query in the original data space is mapped to a region determined by the mapping method, which is the union of one-dimensional ranges. The typical example is iDistance [10]. According to this model, the data set is partitioned and a reference point of each partition is defined. Then, data points are mapped into one-dimensional values based on their distance to the reference point. Under the use of partitioning techniques, iDistance works well in low- or medium-dimensional spaces (up to 30–50 dimensions). However, its performance is sensitive to the selection of reference points and too many random accesses of data pages are required. LDC [32] combines the idea of one-dimensional mapping and vector approximation, and also uses a B+-tree to index the one-dimensional distance. Since LDC is tailor-made for the high-dimensional space, its performance has no superiority in the low- to medium-dimensional space.

Vector approximation maps a sequence of continuous or discrete vectors into a digital sequence which is suitable for communication over or storage in a digital channel. A well-known approximation technique is vector quantization (VQ). According to the classical vector quantization, we map a d-dimensional vector $x \in \mathbb{R}^d$ to another vector $q(x) \in C = \{c_i | c_i \in \mathbb{R}^d, 1 \leq i \leq K\}$. Here, the set C is known as a codebook [33], c_i is a codeword, and K is the number of codewords. The similarity between two vectors can be approximated by the "distance" of their codewords, which will accelerate distance computations. Besides, for k -NN search, to filter irrelevant points efficiently, the codewords can also be used to generate lower bounds of query point, which is helpful to accelerate the search.

The vector approximation used in VA-File can be seen as a scalar quantization. It ignores the dependencies across dimensions, and each point is bounded with a hyper-rectangular cell. HB benefits from the vector quantization. It brings a new cluster-adaptive distance bound by exploiting the separating hyperplane boundaries of Voronoi clusters to get tighter lower distance bounds. Both of these endow HB with two advantages. Firstly, it can compress the data size dramatically and outperform the other NN search approaches. Secondly, compared with bounding sphere or rectangle, HB provides a much better filtering ability for the irrelevant clusters.

5.2 Image representation methods

On the other hand, there exist series of research works in image classification and processing which may also involve evaluation of distances between images; we introduce some representative ones. The first one [34] devotes to selecting graphlets that have highly discriminative and low redundancy topologies to represent the original images. It selects graphlets from each aerial image that can be quantized into a feature vector for further processing. Another one in [35] advocates better characterizing images by exploiting high-order potentials. It manages to integrate multimodal features by introducing a shared entropy measurement and a feature correlation hypergraph (FCH) to respectively capture and model the high-order relations among multimodal features. Besides, works in [36] incorporate image-level semantic information into graphlets with the help of a hierarchical Bayesian network to learn semantic associations between graphlets. It enables the extracted feature vectors containing semantic information which can enhance the semantic accuracy of image representation. What is more, [37] introduces a concept of cellet to construct a fine-grained representation of the spatial layout of images. The cellet can be constructed with representative cells of the original images, which can provide us with an effective representation of the feature of these images.

In fact, these works all aim at finding better representations of image feature vectors, such that images can be better understood. In contrast, our work in this paper aims at optimizing the search process for the best matches given an arbitrary query image which is represented as feature vectors using some image representation methods. Hence, how to better represent the image as feature vectors (i.e. image representation methods) is beyond the scope and is independent of this work. Therefore, in our framework, many image representations methods can be adopted in the images.

6 Conclusion

In this paper, we investigate the unsupervised similarity search in large-scale image database, as well as the essential importance of effective indexing and searching methods for nearest neighbor search in high-dimensional space. According to a newly proposed index method based on separating hyperplanes, namely HB, we focus on addressing its two limitations, huge cost for computing lower bounds and the lack of efficient pruning method for false positives in candidate clusters, respectively. We see that both of these significantly affect the efficiency during the search procedure. Consequently, we propose our novel solutions towards these limitations and develop a new index approach, namely HB+. Firstly, we evaluate all the bounding hyperplanes for one cluster using random projection and select a small part of separating hyperplane boundaries to compute lower bounds. Secondly, we design a new pruning algorithm to eliminate irrelevant points in the candidate clusters. Extensive experiments conducted on three visual feature data sets demonstrate the superiority of HB+ over HB, as well as a series of other popular index schemas such as VA-File, iDistance and a newly proposed method FNN. In future, we will explore the potential of the hyperplane boundary and seek for more effective index structure to solve NN search on more challenging data sets.

Acknowledgments Project supported by the National Natural Science Foundation of China (Grant No. 61173089, 61202179 and 61472298), SRF for ROCS, SEM and Fundamental Research Funds for the Central Universities.

References

1. Carneiro, G., Chan, A.B., Moreno, P.J., Vasconcelos, N.: Supervised learning of semantic classes for image annotation and retrieval. Patt. Anal. Mach. Intell. IEEE. Trans. **29**(3), 394–410 (2007)
2. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proceedings of the 25th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco 518–529 1999
3. Athitsos, V., Potamias, M., Papapetrou, P., Kollios, G.: Nearest neighbor retrieval using distance-based hashing. In: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, Washington, IEEE Computer Society 327–336 (2008)
4. Tao, Y., Yi, K., Sheng, C., Kalnis, P.: Quality and efficiency in high dimensional nearest neighbor search. In: Proceedings of the 35th SIGMOD international conference on Management of data, New York, USA SIGMOD '09, pp. 563–576, ACM (2009)
5. Gan, J., Feng, J., Fang, Q., Ng, W.: Locality sensitive hashing scheme based on dyanmic collision counting. In: Proceedings of the 38th SIGMOD international conference on Management of data. SIGMOD'12, pp. 541–552, ACM (2012)
6. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell. **33**, 117–128 (2011)
7. Weber, R., Schek, H-Jörg., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: VLDB '98 Proceedings of the 24rd International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc. San Francisco, USA pp. 194–205 (1998)
8. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD 84 Proceedings of the 1984 ACM SIGMOD international conference on Management of data, New York, USA pp. 47–57, ACM (1984)
9. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. SIGMOD Rec. **19**(2), 322–331 (1990)
10. David A., Ramesh, W.J.: Similarity indexing with the ss-tree. In: Proceedings of the 12th International Conference on Data Engineering. ICDE'96, pp. 516–523 (1996)
11. Katayama, N., Satoh, S.: The sr-tree: an index structure for high-dimensional nearest neighbor queries. In: Proceedings of the 23rd ACM SIGMOD international conference on Management of data, SIGMOD'97, pp. 369–380 (1997)
12. Ramaswamy S., Rose, K.: Adaptive cluster distance bounding for high-dimensional indexing. IEEE Trans. Knowl. Data. Eng., vol. 23, no. 6, pp. 815–830 June (2011)
13. Jagadish, H.V.: Ooi, B.C., Tan, K.-L., Yu, C., Zhang, R.: idistance: an adaptive B^{+}-tree based indexing method for nearest neighbor search. ACM Trans. Database. Syst. **30**(2), 364–397 (2005)
14. Hwang, Y., Han, B., Ahn, H.-K.: A fast nearest neighbor search algorithm by nonlinear embedding, in Computer Vision and Pattern Recognition (CVPR), IEEE Conference on. IEEE pp. 3053–3060 (2012)
15. Achlioptas, D.: Database-friendly random projections:johnson-lindenstrauss with binary coins. J. Comput. Syst. Sci. **66**(4), 671–687 (2003)
16. Bingham E., Mannila, H.: Random projection in dimensionality reduction: applications to image and text data in SIGKDD pp. 245–250 (2001)
17. Li, P., Hastie, T.J., Church K.W.: Very sparse random projections in SIGKDD, pp. 287–296 (2006)
18. Jeffrey Scott Vitter: Algorithms and data structures for external memory. Found. Trends. Theor. Comput. Sci. **2**(4), 305–474 (2008)
19. Chakrabarti K., Mehrotra, S.: Local dimensionality reduction: A new approach to indexing high dimensional spaces in VLDB '00 In: Proceedings of the 26th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, USA pp. 89–100 (2000)
20. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. ACM Comput. Surv. **33**(3), 322–373 (2001)
21. Berchtold, S., Keim, D.A., Kriegel, H.-P.: The X-tree: an index structure for high-dimensional data. In VLDB '96: Proceedings

of the 22th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, USA pp. 28–39 (1996)
22. Robinson, J.T. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In SIGMOD '81: Proceedings of the 1981 ACM SIGMOD international conference on Management of data, ACM, New York, USA pp. 10–18 (1981)
23. Jon Louis Bentley: Multidimensional binary search trees used for associative searching. Commun. ACM. **18**(9), 509–517 (1975)
24. Berchtold, S., Bohm, C., Jagadish, H.V., Kriegel, H.-P., Sander, J.: Independent quantization: an index compression technique for high-dimensional data spaces In ICDE '00: Proceedings of the 16th International Conference on Data Engineering, IEEE Comput. Soc. Washington, USA 2000, p. 577 (2000)
25. Ferhatosmanoglu, H., Tuncel, E., Agrawal, D.: Vector approximation based indexing for non-uniform high dimensional data sets. In CIKM '00: Proceedings of the ninth international conference on Information and knowledge management, ACM. New York, USA pp. 202–209 (2000)
26. Cui, J., Zhou, S., Sun, J.: Efficient high-dimensional indexing by sorting principal component. Pattern Recogn. Lett. **28**(16), 2412–2418 (2007)
27. Ravi, K.V., Divyakant Agrawal, K., Singh, A.: Dimensionality reduction for similarity searching in dynamic databases. SIGMOD Rec. **27**(2), 166–176 (1998)
28. Van der Maaten, L.J.P., Postma, E.O., Van Den Herik, H.J., Dimensionality reduction : a comparative review. October, vol. 10, no. February, pp. 35, (2009)
29. Lian X., Chen, L.: A general cost model for dimensionality reduction in high dimensional spaces. In ICDE '07: Proceedings of the 23th International Conference on Data Engineering, pp. 66–75 (2007)
30. Berchtold, S., Böhm, C., Kriegal, H.-P.: The pyramid-technique: towards breaking the curse of dimensionality. SIGMOD Rec. **27**(2), 142–153 (1998)
31. Christos F., Searching multimedia databases by content, vol. 3, Springer, New York (1996)
32. Nick K., Beng C.O., Heng T.S., Tung A.K.H.: Ldc: Enabling search by partial distance in a hyper-dimensional space. In ICDE '04: Proceedings of the 20th International Conference on Data Engineering, IEEE Computer Society, Washington, USA p. 6 (2004)
33. Gray, R.M.: Vector quantization. ASSP Magazine. IEEE. vol. 1, no. 2, pp. 4–29 (1984)
34. Zhang, L., Han, Y., Yang, Y., Song, M., Yan, S., Tian Q.: Discovering discriminative graphlets for aerial image categories recognition. IEEE. transac. Image Processing: a publication of the IEEE Signal Processing Society, vol. 22, no. 12, pp. 5071–5084 (2013)
35. Zhang, L., Gao, Y., Hong, C., Feng, Y., Zhu, J., Cai, D.: Feature correlation hypergraph: exploiting high-order potentials for multimodal recognition. Cybernetics. IEEE. Trans. **44**(8), 1408–1419 (2013)
36. Zhang, L., Yang, Y., Gao, Y., Yi, Y., Wang, C., Li, X.: A probabilistic associative model for segmenting weakly-supervised images. Image. Processing. IEEE. Trans.**23**(9), 4150–4159 (2014)
37. Zhang, L., Gao, Y., Xia, Y., Dai, Q., Li, X.: A fine-grained image categorization system by cellet-encoded spatial pyramid modeling. Ind Electron. IEEE. Trans. (2014)