Efficient k-NN Search on Vertically Decomposed Data

Arjen P. de Vries[†] Nikos Mamoulis[§*] Niels Nes[†] Martin Kersten[†]

[†]Centrum voor Wiskunde en Informatica
Kruislaan 413, 1098 SJ, Amsterdam
The Netherlands
{arjen,niels,mk}@cwi.nl

[§]University of Hong Kong
Pokfulam Road
Hong Kong
nikos@csis.hku.hk

Abstract

Applications like multimedia retrieval require efficient support for similarity search on large data collections. Yet, nearest neighbor search is a difficult problem in high dimensional spaces, rendering efficient applications hard to realize: index structures degrade rapidly with increasing dimensionality, while sequential search is not an attractive solution for repositories with millions of objects. This paper approaches the problem from a different angle. A solution is sought in an unconventional storage scheme, that opens up a new range of techniques for processing k-NN queries, especially suited for high dimensional spaces. The suggested (physical) database design accommodates well a novel variant of branch-and-bound search, that reduces the high dimensional space quickly to a small candidate set. The paper provides insight in applying this idea to k-NN search using two similarity metrics commonly encountered in image database applications, and discusses techniques for its implementation in relational database systems. The effectiveness of the proposed method is evaluated empirically on both real and synthetic data sets, reporting the significant improvements in response time yielded.

1. Introduction

Nearest neighbor search in high dimensional spaces finds many applications in domains such as image retrieval, multimedia systems, spatial databases, and data mining. For example, in image retrieval [1] images are represented as points (called *feature vectors*) in a high-dimensional space constructed from color distribution (color histograms), texture patterns (like grayness or smoothness), or image structure (shape). Usually, these feature vectors have high dimensionality; color histograms alone vary from 64 to several hundreds of bins. Images are considered similar if they are located 'close' to each other in this high-dimensional space, according to some distance metric.

*This work was done while the author was with CWI.

ACM SIGMOD '2002 June 4-6, Madison, Wisconsin, USA

The aim for image database research is to accommodate interactive search on millions of images, using the feature space as primary filter and guidance. The most typical query type in image databases finds the k most similar images to a given example. Simply comparing the query vector to all feature vectors, while maintaining an array with the best k answers so far, is too expensive for two reasons: (i) inspecting all feature vectors infers a lot of I/O, and (ii) the CPU cost of comparing each pair of vectors is significant, even if you can quickly assess a vector is not a good candidate. Yet, the analysis in [22] shows that indexing methods based on space partitioning methods, generalized from spatial databases, break down in high dimensional spaces, a problem that has been validated empirically. Another important drawback of these indexing methods is that they are based on a static feature space decomposition, where all dimensions are of equal importance. They are thus unable to support efficiently *weighted* k-NN queries, where dimensions can have different arbitrary importance at search, and queries on arbitrary sub-spaces of the full-dimensional space.

This paper considers a novel direction for improving the efficiency of k-NN search in high dimensional spaces, approaching it as a physical database design problem. The main rationale is that the development of efficient query processing techniques for nearest neighbor search may benefit from physical data independence, i.e., on distinguishing between the logical and physical organization of feature vectors. An unconventional physical design alternative is used, that maintains a separate table for each dimension, containing, of *all* vectors in the repository, the coefficients of *that same dimension*. This physical representation accommodates a novel search technique called Branch-and-bound ON Decomposed data (BOND).

In BOND, the distance between the query point and all data vectors is accumulated by scanning these dimensional projections one-by-one. After processing few of them, *partial* distances of each vector to the query are known; then, lower and upper bounds on the *complete* distance of the k-nearest neighbors are exploited to discard safely from further consideration those vectors that cannot possibly participate in the response set. Applying this process iteratively, reduces the candidate set such that the last stages are performed on just a small database sample. The resulting search process is visualized in Figure 1. The first m dimension columns are scanned, and the best partial scores are computed. Vectors with a smaller best-case score than the worst-case score of the k-th most similar vector are pruned. This process is repeated until the candidate set contains exactly k objects, or

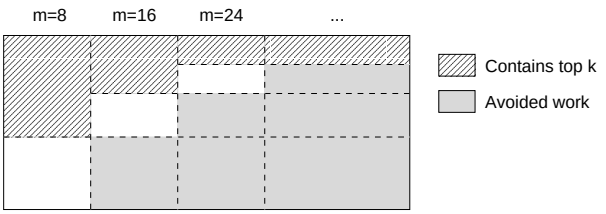


Figure 1: BOND visualized.

all dimensions have been processed.

The advantages of BOND are summarized as follows:

- It avoids a large number of computations compared to a full sequential scan;
- It is conceptually simple, causes practically no storage overhead, and requires no preprocessing of the data;
- Its good performance is robust to increasing dimensionality (assuming a *meaningful* search problem [3]);
- It is a novel technique, orthogonal to previous approaches based on compression of feature vectors (such as [22]).
- On the *same* data representation different variants of k-NN queries can be processed efficiently, including: (i) queries with different weights of importance on the various dimensions, (ii) queries in any dimensional subspace, (iii) queries with various (monotonic) similarity metrics, and (iv) multi-feature queries that combine similarities from various sources [7, 9].

The remainder of this paper is organized as follows. First, related work is presented in Section 2. Section 3 summarizes the notation used throughout the paper. We use image retrieval as an example application of k-NN search, so two metrics frequently used in image retrieval are introduced. Section 4 then presents BOND, and derives the necessary bounds for these two metrics. Sections 5 and 6 discuss optimization techniques for BOND and implementation details. The proposed methodology is evaluated empirically in Section 7 and Section 8 discusses its application to two additional types of k-NN queries. Finally, Section 9 concludes by summarizing the contributions of this work.

2. Related Work

A variety of techniques have been proposed previously to improve upon naive k-NN search. A brute-force solution precomputes for each vector its k nearest neighbors [7], generating a *similarity network*. This method avoids expensive run-time distance computations and ranking, offering perhaps the only viable solution for very large data sets. Yet, updates cannot be done incrementally and fixed constants (like the number of neighbors per object and the similarity metric used) support neither queries with arbitrary k nor weighted ones. Also, it is impossible to query for objects that are not selected from the indexed collection.

Another line of research applied a variety of indexing techniques to speed up the search process. If the number of dimensions is low, a *spatial access method* (SAM) (e.g., the R-tree [10]) stores the feature vectors, such that a k-nearest neighbor search algorithm (e.g., [16]) facilitates efficient search. In practice, however, the number of dimensions in image databases is quite large, and scanning the complete database can be faster [22, 3].

The two-step evaluation technique applied in [8, 13, 18] alleviates these problems. Each original feature vector is mapped onto a small number of dimensions (e.g., 16), such that a (possibly different) distance metric in the low-dimensional space lower-bounds the actual distance in the high-dimensional space (*dimensionality reduction*). The resulting (low-dimensional) vectors are organized in a SAM, which is used to (i) compute a worst-case distance of the k-th nearest neighbor from the query vector and (ii) prune objects with larger distances in the low-dimensional space. A disadvantage of this methodology is the problem of finding a *proper* mapping that preserves enough information from the original space, to filter as many objects as possible using the SAM. Another drawback is the system overhead introduced by the additional set of feature vectors, affecting negatively its space requirements and update speed.

Given the efficiency of sequential scan for high-dimensional search, the Vector Approximation File (VA–File) [22] uses a smaller, approximate representation of the feature vectors (e.g., 8 bits per dimension instead of a double) for an initial filter step; an idea similar to the use of signature files for searching textual data. To compute the final answers, a refinement step using the complete feature vectors is performed. The filter step is fast because it requires small bandwidth, and since the refinement step processes much less data, computing the top-k answers is cheaper than sequential scanning the original vectors. This approximation technique has also been applied to adapt R-trees for high-dimensional problems, by compressing the leaf nodes (the IQ-tree [2]) or storing a compressed representation of bounding boxes of child nodes in the inner nodes (the A-tree [17]).

Summarizing, three approaches have been followed to tackle the curse of dimensionality: (1) indexing based on space partitioning, (2) dimensionality reduction, and (3) data compression. This paper considers techniques from physical database design to represent a collection of feature vectors in the database system. We use the decomposition storage model [5] (also known as *vertical fragmentation*), which was initially proposed for the physical storage of data in relational DBMSs in order to reduce the I/O cost of frequently observed query patterns. It has proven its value in various data management problems since, being applied effectively in commercial products (Sybase-IQ [14]) and research database systems [4].

BOND exploits the possibility of independent access to each dimension provided by the vertical fragmentation. Instead of partitioning the space using all dimensions and building an index for the data, it considers a dynamic order of the dimensional fragments and prunes objects while computing their partial distances to the query object. Furthermore, BOND combines transparently with the advantages of compression, resulting in additional performance improvements. Thus, by combining a new query processing technique (BOND) with a (rather neglected) physical database storage schema, a novel way to solving k-NN search problems has been discovered.

3. Definitions

In this section we introduce the notation used in this paper, as well as the two image search problems used to illustrate the approach.

3.1 Notation

Let $\mathcal{X}$ be a collection of sequences (i.e., N-dimensional vectors) $\mathbf{x} = x_1 \dots x_N$. Let m be an integer: $1 \leq m \leq N$. Operators $^-$ and $^+$ on $\mathbf{x}$ define sequences with the first m and the last $N - m$ elements, respectively. Thus, $\mathbf{x}^- = x_1 \dots x_m$ and $\mathbf{x}^+ = x_{m+1} \dots x_N$. Let $T(\mathbf{x})$ be the sum of the elements in the sequence, and let this function apply also on the partial sequences $\mathbf{x}^-$ and $\mathbf{x}^+$. Let $\mathcal{S}$ be an associative and monotonic aggregate function, defined for sequences $\mathbf{x}$ of any length N. Let $\mathbf{X}$ denote $[\mathbf{x}_1, \dots, \mathbf{x}_{|\mathcal{X}|}]^T$, a vector representing collection $\mathcal{X}$. We assume throughout the paper an implicit mapping between collections $\mathcal{X}$, $\mathcal{H}$, $\mathcal{V}$, and their corresponding vectors $\mathbf{X}$, $\mathbf{H}$, $\mathbf{V}$.

Operations are often applied to each element of a collection. $\mathbf{S} = \mathcal{S}(\mathbf{X})$ denotes the result of element-wise application of aggregate function $\mathcal{S}$ to each $\mathbf{x}_i$ in $\mathbf{X}$. Analogously, operators $^-$ and $^+$ apply also on $\mathbf{X}$ and $\mathbf{S}$, e.g., $\mathbf{X}^- = [\mathbf{x}_\mathbf{1}^-, \dots, \mathbf{x}_{|\mathcal{X}|}^-]^T$ and $\mathbf{S}^- = \mathcal{S}(\mathbf{X}^-)$.

Symbols $\mathbf{S}^+_{\min}$ and $\mathbf{S}^+_{\max}$ denote the minimum and maximum bounds for each element in $\mathbf{S}^+$. Similarly, $\mathbf{S}_{\min}$ and $\mathbf{S}_{\max}$ are the minimum and maximum bounds of $\mathbf{S}$, provided that $\mathbf{S}^-$ has been computed. Finally, let $\kappa_{\min}$ ($\kappa_{\max}$) be the k-th largest (smallest) element of $\mathbf{S}_{\min}$ ($\mathbf{S}_{\max}$). Table 1 summarizes the notation used throughout the paper. Some symbols are implicitly parameterized by an index m or a collection $\mathcal{X}$.[1]

Table 1: Notation.

$\mathbf{x}$	Sequence $x_1 \dots x_N$.
$\mathbf{x}^-$	Sequence $x_1 \dots x_m$.
$\mathbf{x}^+$	Sequence $x_{m+1} \dots x_N$.
$T(\mathbf{x})$	$\sum_{i=1}^{N} x_i$
$T(\mathbf{x}^-),\ T(\mathbf{x}^+)$	$\sum_{i=1}^{m} x_i,\ \sum_{i=m+1}^{N} x_i$
$\mathcal{X}$	Collection of sequences $\{\mathbf{x}\}$.
$\mathbf{X}$	Vector of sequences $[\mathbf{x}_1, \dots, \mathbf{x}_{\lvert\mathcal{X}\rvert}]^T$.
$\mathcal{S}(\mathbf{x})$	Aggregate function $\mathcal{S} : \mathbf{x} \to \mathbb{R}$.
$\mathbf{S}$	Vector $\mathcal{S}(\mathbf{X})$.
$\mathbf{S}^+_{\max}$, $\mathbf{S}^+_{\min}$	Element-wise bounds for $\mathbf{S}^+$.
$\mathbf{S}_{\max}$, $\mathbf{S}_{\min}$	$\mathbf{S}^- + \mathbf{S}^+_{\max}$, $\mathbf{S}^- + \mathbf{S}^+_{\min}$.
$\kappa_{\min}$, $\kappa_{\max}$	The $k-$th element of $\mathbf{S}_{\min}$, $\mathbf{S}_{\max}$.

3.2 Two Common Metrics

Applications in image retrieval frequently use *histogram intersection* as a metric for image similarity [21], summing the overlap between the two histograms in each dimension. Two images are considered similar if their histogram intersection is large. Another commonly used metric is the *Euclidean distance*: images are considered similar if their distance in the feature space is small.

Histogram Intersection

Let $\mathcal{H}$ be a collection of normalized image histograms (N-dimensional vectors $\mathbf{h}$, $\forall \mathbf{h} \in \mathcal{H} : T(\mathbf{h}) = 1$).

[1]We valued the readability of $\mathbf{S}^+_{\max}$ over the preciseness of $\mathbf{S}^{\max}_{m+1}(\mathcal{X})$.

DEFINITION 1. *Given two normalized histograms* $\mathbf{h}$ *and* $\mathbf{q}$*, we define* ***histogram intersection*** *as a measure of similarity between them:*

$$Sim(\mathbf{h}, \mathbf{q}) = \sum_{i=1}^{N} \min(h_i, q_i) \tag{1}$$

Using histogram intersection assumes that the different dimensions are uncorrelated. This metric was reported in [20] to be superior to Euclidean distance for color histograms, mainly because of its ability to reduce the contribution of the irrelevant vectors in the query result. The intersection of two histograms is approximately one if the histograms are much alike, because $\forall i, 1 \leq i \leq N : \min(h_i, q_i) \simeq h_i$ and $T(\mathbf{h}) = 1$. If the histograms differ significantly, their scalars differ significantly in each dimension, and their intersection is small.

Euclidean distance

Let $\mathcal{V}$ be a collection of N-dimensional feature vectors $\mathbf{v}$ in the unit hyperbox ($\forall \mathbf{v} \in \mathcal{V} : 0 \leq v_i \leq 1$).

DEFINITION 2. *The* ***squared Euclidean distance*** *between two vectors* $\mathbf{v}$ *and* $\mathbf{q}$ *of dimensionality* N *is defined as follows:*

$$\delta(\mathbf{v}, \mathbf{q}) = \sum_{i=1}^{N} (v_i - q_i)^2 \tag{2}$$

The actual Euclidean distance is the square root of $\delta(\mathbf{v}, \mathbf{q})$. Using squared distance δ reduces computations; obviously, the relation between the actual Euclidean distance and $\delta(\mathbf{v}, \mathbf{q})$ is monotonic. Two images are considered similar if the distance between them is small. So, we define the following similarity metric:

$$Sim(\mathbf{v}, \mathbf{q}) = 1 - \sqrt{\frac{1}{N}\delta(\mathbf{v}, \mathbf{q})} \tag{3}$$

4. BOND: Branch-and-bound ON Decomposed data

This section describes the generic search strategy applied in BOND, followed by the derivation of algorithms for k nearest neighbor search using histogram intersection and Euclidean distance.

Algorithm 1 is a brute-force approach to finding the k sequences with the largest value of aggregate $\mathcal{S}$. Step 1 represents a naive loop over all elements of $\mathcal{X}$. In practice, N and $|\mathcal{X}|$ are large, and sequential search becomes very expensive.

ALGORITHM 1. *Sequential-Search($\mathcal{X}$,k)*

1. *Compute* $\mathbf{S} = \mathcal{S}(\mathbf{X})$*;*
2. *Rank* $\mathbf{S}$ *and return the k highest values.*

Assuming that aggregate $\mathcal{S}$ is monotonically increasing, we propose BOND; the following *branch-and-bound* alternative to improve efficiency:

ALGORITHM 2. *BOND($\mathcal{X}$,k,m)*

1. *Compute* $\mathbf{S}^- = \mathcal{S}(\mathbf{X}^-)$*;*
2. *Determine* $\mathbf{S}_{\max}$ *and* $\mathbf{S}_{\min}$*;*

3. *Determine* $\kappa_{\min}$ *from* $\mathbf{S}_{\min}$*;*

4. *Create candidate set* $\mathcal{C}$*, by removing from* $\mathcal{X}$ *the* $\mathbf{x}_i$ *for which* $\mathbf{S}_{\max}[i] < \kappa_{\min}$*;*

5. *Apply iteratively steps 1-4 on* $\mathcal{C}$ *for a larger* m *until* $|\mathcal{C}| = k$*, or all dimensions have been processed.*

Pruning step 4 states formally which sequences $\mathbf{x}$ *may* still reach the top k while aggregating their remaining values $x_{m+1} \dots x_N$. It is derived from the fact that each partial score increases with $\mathbf{S}^+_{\min}$ at least, but never with more than $\mathbf{S}^+_{\max}$. Algorithm 2 is meant for finding the k elements with the largest values in $\mathbf{S}$. When we are interested in the k smallest values of $\mathbf{S}$, step 4 prunes each $\mathbf{x}_i$ for which $\mathbf{S}_{\min}[i] > \kappa_{\max}$.

4.1 Pruning bounds for histogram intersection

The remaining problem is to derive computationally inexpensive rules for determining $\mathbf{S}^+_{\max}$ and $\mathbf{S}^+_{\min}$ for our similarity metrics. Let aggregate $\mathcal{S}()$ be the histogram intersection, as given in Equation 1. We first break the sum into partial sums $\mathcal{S}(\mathbf{h}^-, \mathbf{q}^-)$ and $\mathcal{S}(\mathbf{h}^+, \mathbf{q}^+)$:

$$\mathcal{S}(\mathbf{h}, \mathbf{q}) = \underbrace{\sum_{i=1}^{m} \min\{h_i, q_i\}}_{\mathcal{S}(\mathbf{h}^-, \mathbf{q}^-)} + \underbrace{\sum_{j=m+1}^{N} \min\{h_j, q_j\}}_{\mathcal{S}(\mathbf{h}^+, \mathbf{q}^+)} \quad (4)$$

The next inequality provides a rather straightforward upper bound for each $\mathcal{S}(\mathbf{h}^+, \mathbf{q}^+)$:

$$\mathcal{S}(\mathbf{h}^+, \mathbf{q}^+) \leq \sum_{j=m+1}^{N} q_j = T(\mathbf{q}^+) = 1 - T(\mathbf{q}^-) \quad (5)$$

The obvious lower bound for $\mathcal{S}(\mathbf{h}^+, \mathbf{q}^+)$ is 0. Thus, $\mathbf{S}^+_{\max}$, $\mathbf{S}^+_{\min}$ can be considered as arrays containing these constant values, and $\mathbf{S}_{\max}$, $\mathbf{S}_{\min}$ can be obtained trivially from the already computed $\mathbf{S}^-$. $\kappa_{\min}$ is then the k-th largest element of $\mathbf{S}^-$, and no histogram $\mathbf{h}_i$ with:

$$\mathcal{S}(\mathbf{h}_i^-, \mathbf{q}^-) + (1 - T(\mathbf{q}^-)) < \kappa_{\min} \quad (6)$$

can ever end up in the top k best vectors. We denote the resulting criterion with H_q, since it only depends on the query vector. Note that, in this special case, the derived bounds are the same for each image.

Stricter upper and lower bounds for $\mathcal{S}(\mathbf{h}^+, \mathbf{q}^+)$ can be defined using information from $\mathbf{h}$:

$$\begin{aligned} \mathcal{S}(\mathbf{h}^+, \mathbf{q}^+) &\leq \min\{T(\mathbf{h}^+), T(\mathbf{q}^+)\} \\ &= 1 - \max\{T(\mathbf{h}^-), T(\mathbf{q}^-)\} \quad (7) \\ \mathcal{S}(\mathbf{h}^+, \mathbf{q}^+) &= \sum_{i=m+1}^{N} \min\{q_i, h_i\} \\ &\geq \sum_{i=m+1}^{N} \min\{q_{\min}, h_i\} \\ &\geq \min\{q_{\min}, T(\mathbf{h}^+)\} \\ &= \min\{q_{\min}, 1 - T(\mathbf{h}^-)\}, \quad (8) \end{aligned}$$

where q_{min} is the minimum element of $\mathbf{q}^+$. In other words, as long as $T(\mathbf{h}^+)$ is larger than $q_{\min}$, the histogram intersection of the remaining values is at least $q_{\min}$; otherwise, it is equal to $T(\mathbf{h}^+)$.

Table 2: Example collection $\mathcal{H}$.

H	$h \in \mathcal{H}$	$\mathbf{S}^-$	$\mathbf{S}_{\min}$	$\mathbf{S}_{\max}$	$\mathcal{S}$
$\mathbf{h}_1$	$< 0, 0.1, 0, 0.9 >$	0.1	0.15	0.25	0.15
$\mathbf{h}_2$	$< 0.05, 0.05, 0.9, 0 >$	0.1	0.15	0.25	0.2
$\mathbf{h}_3$	$< 0.8, 0.1, 0.05, 0.05 >$	**0.8**	0.85	**0.9**	**0.9**
$\mathbf{h}_4$	$< 0.2, 0.6, 0.1, 0.1 >$	0.35	0.4	0.5	0.5
$\mathbf{h}_5$	$< 0.7, 0.15, 0.15, 0 >$	**0.85**	0.9	**1**	**0.95**
$\mathbf{h}_6$	$< 0.925, 0, 0, 0.025 >$	**0.7**	0.725	0.725	0.725
$\mathbf{h}_7$	$< 0.55, 0.2, 0.15, 0.1 >$	**0.7**	0.75	**0.85**	**0.85**
$\mathbf{h}_8$	$< 0.05, 0.1, 0.05, 0.8 >$	0.15	0.2	0.3	0.25
$\mathbf{h}_9$	$< 0.45, 0.5, 0.05, 0.05 >$	**0.6**	0.65	0.7	0.7

These (stricter) bounds differ for each $\mathbf{h}_i$. Thus, $\mathbf{S}_{\max}$ and $\mathbf{S}_{\min}$ cannot be determined only from $\mathbf{S}^-$; we need partial sum $T(\mathbf{h}^-)$ for each image as well. Equations 7 and 8 then define $\mathbf{S}^+_{\max}$ and $\mathbf{S}^+_{\min}$, respectively. Now, if $\kappa_{\min}$ is the k-th largest element of $\mathbf{S}_{\min}$, a stricter pruning criterion H_h for histogram intersection can be defined as follows:

$$\underbrace{\mathcal{S}(\mathbf{h}_i^-, \mathbf{q}^-) + 1 - \max\{T(\mathbf{h}_i^-), T(\mathbf{q}^-)\}}_{\mathbf{S}_{\max}[i]} < \kappa_{\min} \quad (9)$$

The advantage of rule H_q over H_h is that it is computationally cheaper and requires less bookkeeping information. Using H_q, we maintain only the essential table $\mathbf{S}^-$ of partial similarities at each iteration, which accumulates to the similarity of the final solutions. Using H_h requires also keeping the partial sums of the values accessed so far (i.e., $T(\mathbf{h}^-)$ for each $\mathbf{h}$). Nevertheless H_h is more precise, so it is expected to identify a larger number of disqualifying vectors.

4.2 An Example

A simple example illustrates how Algorithm 2 works for histogram intersection. Consider collection $\mathcal{H}$ as shown in Table 2, query histogram $\mathbf{q} =< 0.7, 0.15, 0.1, 0.05 >$, and a search problem in which we like to find the three nearest neighbors ($k = 3$). The three best matches are $\{\mathbf{h}_3, \mathbf{h}_5, \mathbf{h}_7\}$, which could be found by computing $\mathcal{S}(\mathbf{h}_i, \mathbf{q})$ for each $\mathbf{h}_i \in \mathcal{H}$, sorting the resulting sums, and returning the three best results.

First, consider pruning rule H_q. With $m = 2$, our algorithm first computes the partial sums for each histogram (column $\mathbf{S}^-$). The trivial lower bound equals zero, we use the third highest value $\kappa_{\min} = 0.7$ for the pruning step. Histograms $\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_4, \mathbf{h}_8\}$ can be removed from the candidate set, because $\mathcal{S}(\mathbf{h}_i^-, \mathbf{q}^-) < \kappa_{\min} - 0.15 = 0.55$; the resulting candidate set is printed boldface in column $\mathbf{S}^-$. Only $\mathbf{h}_6$ and $\mathbf{h}_9$ take part in the next step without contributing to the final result set.

Rule H_h takes advantage of the information in $\mathbf{h}^-$ as well. It computes columns $\mathbf{S}_{\min}$ and $\mathbf{S}_{\max}$ as shown in the table, determines a (higher) $\kappa_{\min} = 0.75$ from $\mathbf{S}_{\min}$, and selects the histograms $\mathbf{h}_i$ with $\mathbf{S}_{\max}[i] < \kappa_{\min}$, which are shown in boldface again. H_h removes $\mathbf{h}_6$ and $\mathbf{h}_9$ from the candidate set as well, already identifying the three best results. Obviously, in this small example, we have already seen half of the data, so a good reduction of the data set is not so surprising. The experiments in Section 7 demonstrate that this branch-and-bound strategy works on real data sets indeed.

4.3 Pruning bounds for Euclidean distance

Similar pruning rules can be derived when Euclidean distance is used as similarity metric. Assume that the aggregate function $\mathcal{S}()$ is defined by equation 2, i.e., it is the squared Euclidean distance[2]. Unlike histogram intersection, we are interested in the objects with the smallest values in $\mathbf{S}$. A simple pruning rule E_q depends on the query vector $\mathbf{q}$ and the partially computed distances $\mathbf{S}^-$ only. Obviously $\mathbf{S}_{\min} = \mathbf{S}$, i.e., the lower bound of the distance for each vector is the already computed distance, since $\mathbf{v}^+$ may be equal to $\mathbf{q}^+$. The upper bound of $\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+)$ is also constant. Geometrically, it is the distance between $\mathbf{q}^+$ and the furthest corner in the hyperspace defined by the remaining dimensions:

$$\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+) \leq \sum_{i=m+1}^{N} \max\{q_i, 1-q_i\}^2 \tag{10}$$

Now define stricter bounds for $\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+)$, assuming that $T(\mathbf{v}^+)$ is known.

LEMMA 1. *Assume that the values of $\mathbf{q}^+$ are in decreasing order, i.e., $q_i > q_{i+1}, \forall i > m$. Let $l = N - \lfloor T(\mathbf{v}^+) \rfloor$, $u_l = T(\mathbf{v}^+) - \lfloor T(\mathbf{v}^+) \rfloor$. The upper bound of $\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+)$ is then defined by:*

$$\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+) \leq \sum_{i=m+1}^{l-1} q_i^2 + (u_l - q_l)^2 + \sum_{i=l+1}^{N} (1-q_i)^2 \tag{11}$$

Proof. Lemma 1 states that the distance is maximized when the values of $T(\mathbf{v}^+)$ are distributed such that the dimensions in increasing order of value in $\mathbf{q}^+$ have the largest possible value. Let $m = N-2$ and $q_{N-1} \geq q_N$. First assume that $T(\mathbf{v}^+) \leq 1$. According to the lemma the additional distance is maximized if $h_{N-1} = 0$ and $h_N = T(\mathbf{v}^+)$. It suffices to prove that if we 'move' a part x of $T(\mathbf{v}^+)$ from h_N to h_{N-1} the distance decreases. This can be shown by evaluating the following inequality: $(q_{N-1} - x)^2 + (q_N - T(\mathbf{v}^+) + x)^2 \leq q_{N-1}^2 + (q_N - T(\mathbf{v}^+))^2$. The proof is similar for $1 < T(\mathbf{v}^+) \leq 2$, where for any $x, 0 < x \leq 2 - T(\mathbf{v}^+)$ the following inequality holds: $(T(\mathbf{v}^+) - 1 + x - q_{N-1})^2 + (1 - x - q_N)^2 \leq (T(\mathbf{v}^+) - 1 - q_{N-1})^2 + (1 - q_N)^2$. By induction, inequality 11 is proven for every $m < N - 2$. □

LEMMA 2. *The lower bound of $\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+)$ is defined by:*

$$\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+) \geq \frac{(T(\mathbf{v}^+) - T(\mathbf{q}^+))^2}{N - m} \tag{12}$$

Proof. Lemma 2 suggests that the increase of δ is minimized if the differences in each of the remaining dimensions are all minimal and equal. This stems from the basic fact that when $\sum_{i=1}^{n} x_i$ is constant, then $\sum_{i=1}^{n} x_i^2$ is minimized when $\forall i, x_i = (\sum_{i=1}^{n} x_i)/n$. □

Figure 12 in the appendix visualizes geometrically the special case in which only the last two dimensions remain, i.e. $m = N - 2$, and $T(\mathbf{v}^+) \leq 1$. Lemmas 1 and 2 provide the required bounds to apply Algorithm 2.[3] Notice that $T(\mathbf{v}^+)$ is needed for each vector in order to define the precise bounds. Unlike the histogram intersection case, for which $T(\mathbf{v}^+)$ equals $1 - T(\mathbf{v}^-)$, $T(\mathbf{v}^+)$ cannot be computed from $T(\mathbf{v}^-)$ only, since $T(\mathbf{v})$ differs for each vector $\mathbf{v}$. A simple solution materializes and uses this extra table. $T(\mathbf{v}^+)$ is then initially a copy of $T(\mathbf{v})$ and it is updated at each step. In the rest of the paper, criterion E_v refers to pruning using the lemmas and $T(\mathbf{v})$.

[2]The reason we do not use the metric defined by equation 3 is that it is far more complex and gives essentially the same result.

[3]Let $diff = (T(\mathbf{v}^+) - T(\mathbf{q}^+))/(N-m)$ and assume that the values in $\mathbf{q}^-$ are in decreasing order (like in the definition of lemma 1). In the special cases $(i)\, diff < 0 \wedge |diff| < q_N$ and $(ii)\ diff > 0 \wedge q_{m+1} + |diff| > 1$, we use a stricter lower bound than inequality 12. Details are omitted for sake of readability.

5. Optimization Issues

Algorithm 2 can be applied directly using the bounds derived above. This section discusses some optimization techniques that enhance its efficiency. More specifically, it investigates the importance of choosing a good ordering of dimensions, and discusses how to tune the frequency by which pruning is attempted.

5.1 Finding a good order of the dimensions

The aggregates used are not only associative and monotonic, but also commutative: the sequence in which we process the dimensions does not affect the final result, so it is a good idea to define an order that prunes a large percentage of the vectors early. Of course, it is not possible to know a priori the effectiveness of each dimension in pruning. But, a combination of the distribution of values in $\mathbf{q}$ with statistical information about $\mathcal{H}$ guides the definition of a good order.

Without additional knowledge about the distribution in $\mathcal{H}$, condition H_q in histogram intersection can be expected to prune the candidate set most succesfully if the right-hand side of the inequality has the highest value, i.e., processing the dimensions in decreasing order of scalars in $\mathbf{q}$. In other words, had $\mathbf{q}$ in the example of Section 4.1 been $< 0.15, 0.1, 0.7, 0.05 >$, dimensions 3 and 1 should be considered to compute partial scores $\mathbf{S}^-$. Notice that this ordering is not necessarily optimal, and a better estimate could be obtained if more were known about $\mathcal{H}$.

Rules H_h and E_v consider also the distribution of values in $\mathcal{H}$. Figure 2 shows statistics from a real dataset containing 59,619 166-dimensional vectors (color histograms of images from the Corel collection [6]). The upper diagram plots the mean value of each bin, the lower one shows the distribution of values in a histogram if taken in decreasing order. Notice that for a specific image, the histogram values follow a Zipfian distribution. Of course, the bins that take the highest values are not the same in every image, as indicated by the leftmost plot. Given this data distribution, processing the dimensions in decreasing order of the values maximizes the chances to find images that are part of the top k early for rules H_h and E_v as well: the dimensions with high values are skewed, so most images are expected to have low values in these dimensions and will be pruned early. Of course, were the distribution of values different, a different processing order should be considered to take the most skewed dimensions first.

5.2 How many dimensions count?

Choosing a small m prunes the candidate set sooner, possibly avoiding a lot of query processing. However, it adds a non-trivial overhead in computing the k-th element more

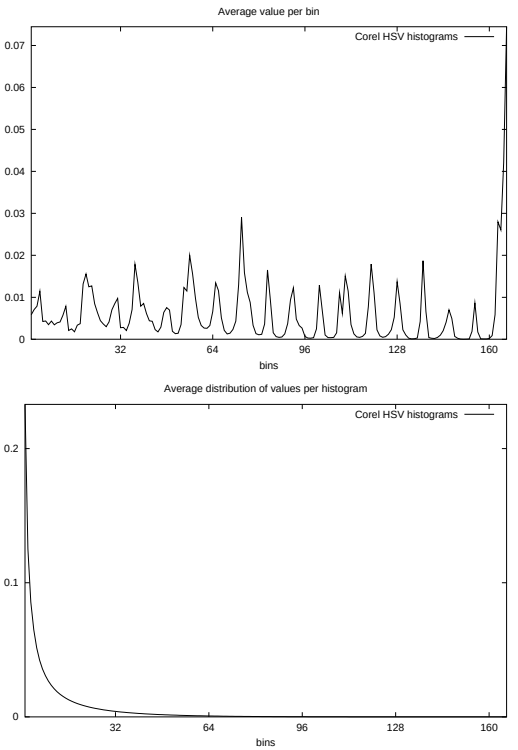


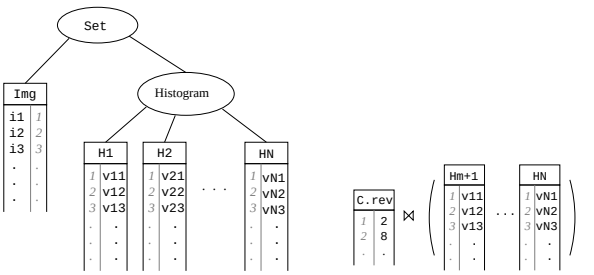
Figure 2: Statistics from a real dataset

frequently as well as updating the candidate set. Thus, m should be sufficiently large for the number of pruned images at each step to be non-trivial, and sufficiently small to have impact on the search speed, compared to full-scanning the images of the previous step.

An optimal choice of m takes into account both data statistics and the query vector. For instance, H_q will not prune any image until the right-hand side of equation 6 is positive. This can happen only when $T(\mathbf{q}^-) > 0.5$, since $\kappa_{\min} \leq T(\mathbf{q}^-)$. Thus, attempting to reduce the data set is futile until this condition applies. As we will see later, after the number of candidates has shrunk to a small superset of the final result, the effect of pruning reduces significantly and the benefit of pruned search is negligible in comparison to performing a full scan on the remaining candidates. Therefore, the significant effects of pruning occur only within a range of dimensions; the optimization problem is reduced to estimating this range and choosing a good m for it. A variant that we have not studied yet is whether m should be adapted dynamically to the expected pruning effect.

6. Implementation

An interesting property of branch-and-bound on decomposed data is that it can be expressed in standard relational algebra; it does not require user-defined types or advanced indexing structures. The proposed query processing techniques have been implemented in Monet [4], a research DBMS which provides a variety of highly efficient implementations of common algebraic operators (join, select, etc.), using strategies such as positional lookup, hash lookup, or binary search. Administration of so-called 'properties' (e.g. sorted, keyed, dense) propagates fragmentation information through operators, to avoid unnecessary joins if possible.



(a) stored histograms (b) step 3 of BOND

Figure 3: Implementation in a relational database.

6.1 Basic algorithm

We vertically fragment the collection of histograms $\mathcal{H}$ into N binary relations `Hi` of length $|\mathcal{H}|$, storing tuples with a histogram identifier and the value of the i-th histogram bin h_i. The resulting tables are shown in Figure 3, in which histogram $\mathbf{h}_2$, with histogram identifier 2, has value $< v_{12},\ v_{22},\ \ldots,\ v_{N2} >$, and belongs to image i_2. Exploiting the known, densely ascending order of histograms, avoids materialization of the histogram identifiers; illustrated in Figure 3 by italic numbers. This serves two goals: it allows positional lookup of scalar values given a histogram identifier, and, it saves storage.[4] BOND with rule H_q is expressed in the Monet Interpreter Language (MIL) as follows:

```
1. for i in 1..m do
     Di := [min](Hi, const Qi);
   Smin := [+](D1, ..., Dm);
2. sumQ := Q1 + ..  + Qm;
   sk := Smin.kfetch( k );
   maxbound := sk + sumQ - 1;
   C := Smin.uselect(maxbound, 1.0);
3. for i in m+1..N do
     Hi := C.reverse.join(Hi);
```

Step 1 computes partial similarity $\mathcal{S}(\mathbf{h}^-)$ for each histogram in the collection. The `[f]()` construct is the *multi-join map*, which performs an implicit equi-join on the left-hand attribute of multiple binary relations, and executes its operator (`f`) on the right-hand arguments of the join result. The `const` keyword denotes its parameter as a constant into all operator executions. Thus, the `[min]` takes the minimum of h_i and q_i for all histograms, whereas the `[+]` joins these results and adds them together; because these tables are aligned, a *positional* join (with negligible cost) is chosen as physical operator.

Step 2 computes the maximum bound derived in rule H_q, and selects the identifiers of the candidate histograms with partial scores that may still end up in the best k results. The `kfetch` operator selects the k-th largest element of `Smin` using a priority queue implemented as a heap, with worst-case cost $O(n \log k)$. The `uselect` operator is the 'unary range select', which returns the left-hand values of tuples from `Smin` with right-hand values in the specified range, setting the right-hand side of the result to a densely ascending range of (virtual) oids. Finally, step 3 reduces the remaining tables

[4] A third of the tablesize, assuming that an `oid` value is represented in 4 bytes and a `dbl` value in 8 bytes.

of $\mathbf{H}^+$ to the candidate set (Figure 3b, assuming histograms 2 and 8 are in the candidate set).

Positional joins construct the results of each iteration of the algorithm cheaply. In early iterations, however, when selectivity is still rather low, copying a large proportion of the table into the result consumes too many resources. As a more efficient alternative, we initially use another physical implementation of `uselect`, creating a bitmap index on the histogram identifiers to represent the pruned candidate set. After several iterations, when the candidate set has reduced significantly, the query processor switches to the 'standard' positional joins approach, resulting in much smaller base tables for the subsequent iterations. Another advantage of the bitmap index is its usage to speed up complex queries involving both k-NN and other predicates, by initializing the bitmap with the result of a prior selection predicate such as 'photographs taken in 1992'. CPU cost is reduced as distances only need be computed for those candidates satisfying the predicate.

6.2 Updates

By their nature, large collections of high-dimensional vectors (e.g., image databases) is relatively static. Yet, in case of updates, or more likely when appending new images to extend a collection, the cost for our storage scheme are the 'normal' cost of updating vertically fragmented collections. As argued already in [5], update performance will approximate the efficiency of updates in a normal relational storage scheme, especially when using differential files and performing mainly batch updates. In our implementation, the same bitmap as used in step 3 marks the deleted image histograms, until periodic reorganization of the collection.

7. Experiments

This section evaluates pruning efficiency and run-time cost for various instances of Algorithm 2. Experiments on a real dataset verify the pruning effects of the four criteria. The effects (on performance) of the choice of k and the ordering of dimensions are measured on the same data set. The next experiment validates BOND's robustness to dimensionality. A run-time cost comparison between BOND and sequential scan follows, with and without compression. Finally, the effectiveness of pruning is validated on synthetic datasets with varying data distributions. Unless otherwise stated, in all experimental instances k was set to 10. Experiments were run on a PC with an AMD Athlon MP 1500+ (1333MHz) processor.[5]

7.1 Pruning effects of the criteria

We evaluated the pruning criteria using a 166-dimensional dataset created from the Corel image database (59,619 images) [6]. The histograms were created using the methodology and parameters described in [19]. The HSV values of all pixels were extracted and quantized to a space that consists of (18 hues)·(3 saturations)·(3 values) + (4 grays) = 166 bins. The values of each histogram were then normalized to sum up to 1. Figure 2 provides statistical information about this dataset.

For each pruning metric, we ran 100 queries randomly selected from the collection. The dimensions are ordered by decreasing values in $\mathbf{q}$ and $m = 8$. Figure 4 plots the best, average, and worst pruning efficiency using H_q and H_h. Our technique manages to shrink fast the search space; more than 98% of the images are discarded after on average just 1/5 of the dimensions. Observe, that the average pruning efficiency of H_q is close to the one of H_h, which has larger overhead (due to the maintenance of $T(\mathbf{h}^-)$). The best case is a 'perfect one'; every false hit is pruned after just one iteration (8 dimensions). Another interesting statistic is that on the average the top-k images are identified after 64 dimensions, which means that 102 tables need not be accessed at all. This shows another advantage of BOND; even if the worst memory settings apply, dimension-wise pruned search would do much better than sequential scan.

Figure 5 plots the pruning efficiency of BOND with pruning criteria E_q and E_v and Euclidean distance as metric. Since we know for this specific dataset that $T(\mathbf{v}) = 1$ for each $\mathbf{v}$, we replaced the upper bound in E_q (defined by inequality 10) by the stricter bound $\max\{q_{\max}^2, (1 - q_{\min})^2\}$. We again ordered the dimensions in decreasing value in $\mathbf{q}$, because this ordering considers the most skewed dimensions first. In contrast to the small difference between H_q and H_h, E_q prunes hardly any image. This can be explained by the large upper bound of $\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+)$, which cannot be practical without knowledge about $T(\mathbf{v}^+)$. On the other hand, E_v manages to prune well, but not as fast as the histogram intersection methods. In the rest of the paper, we will not consider criterion E_q again. Notice that, although we performed the Euclidean distance experiments on the same dataset for comparability, the actual distance distribution between points in the dataset suggests that histogram intersection is a more appropriate metric for image similarity.

7.2 Effects of *k* and ordering of dimensions

We tested the effect of k in the pruning of H_q by running the sample queries and averaging the number of pruned images per dimension (Figure 6). Observe that even with as large values as 1000, BOND manages to prune the space early. The large difference between k=1 and k=10 is due to the fact that the queries are taken from the dataset, thus for k=1 the top-k element is a perfect match with high pruning efficiency. Recall that no images are pruned until the 15-th dimension, where $T(\mathbf{q}^-)$ becomes larger than 0.5.

The nature of skew in the dataset (few dimensions with large values in $\mathbf{q}$ and many with values close to zero) favors considering the dimensions in decreasing order of value in $\mathbf{q}$ for both similarity metrics. Figure 7 verifies this reasoning. The three lines show the pruning effect of H_q when dimensions are taken (i) in decreasing value in $\mathbf{q}$, (ii) at random, and (iii) in increasing value (worst setting). The fact that the best ordering depends on $\mathbf{q}$ (so it is not static) favors the application of BOND in comparison to sequential scan and other methods, because of its flexibility to consider the dimensions in any order without penalty in access cost.

7.3 Effects of dimensionality

The next experiment validates the robustness of our method to the dimensionality of the dataset. From the Corel image database, we generated four HSV histogram datasets of dimensionality 26, 52, 166 and 260. Figure 8 shows the pruned images as a percentage of processed dimensions, when E_v is used. Observe that the effectiveness decreases with di-

[5]The experiments have been repeated on a SGI Origin 2000 workstation with a MIPS R12000 300MHz processor, with similar results.

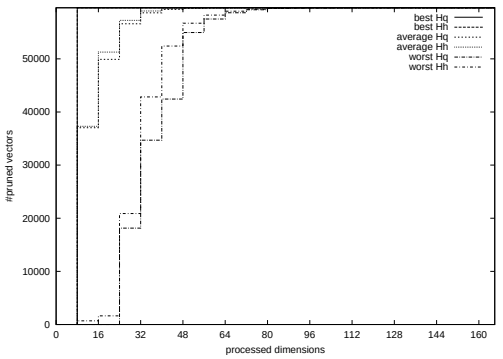


Figure 4: Pruning effects of H_q and H_h

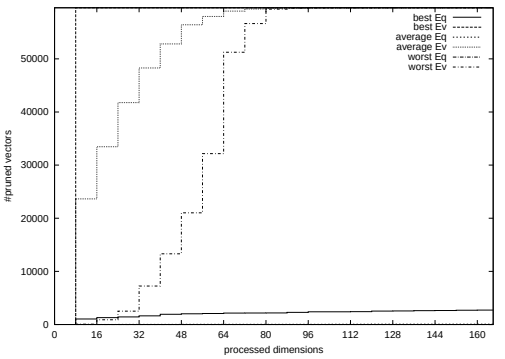


Figure 5: Pruning effects of E_q and E_v

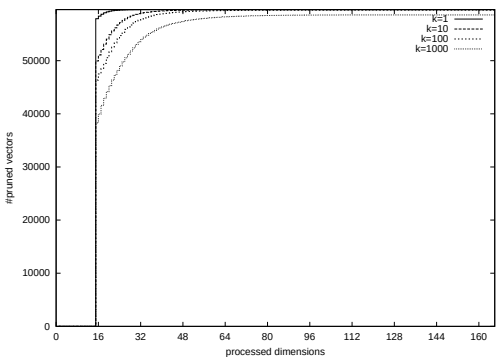


Figure 6: Effects of k in search ($m = 1$)

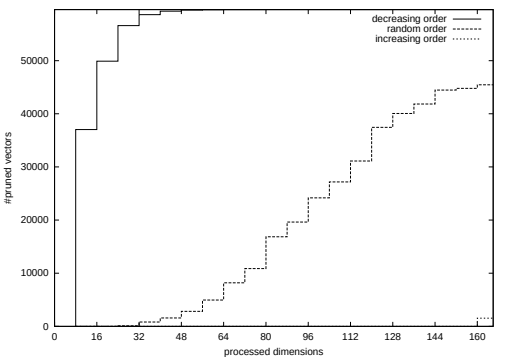


Figure 7: Effects of dimensional orderings

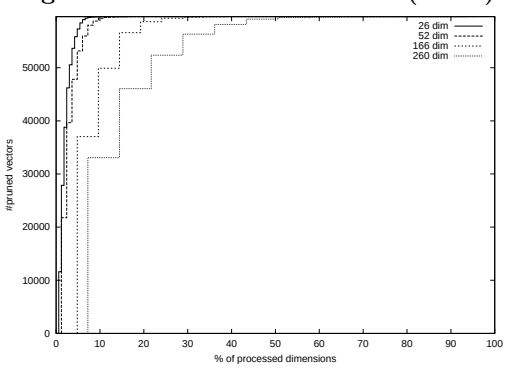


Figure 8: Impact of dimensionality

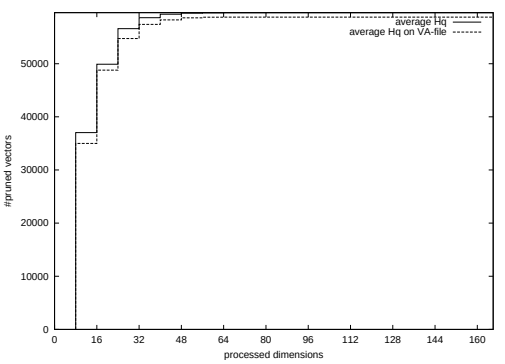


Figure 9: Pruning with approximated vectors.

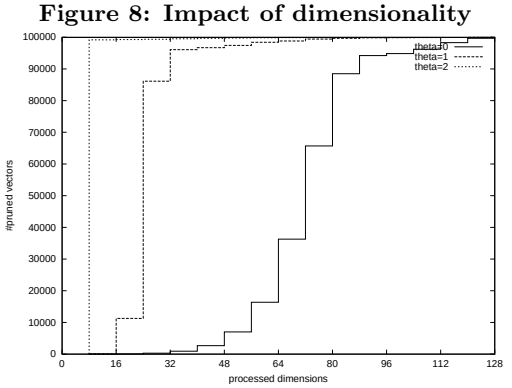


Figure 10: Effects of skew on the data (E_v)

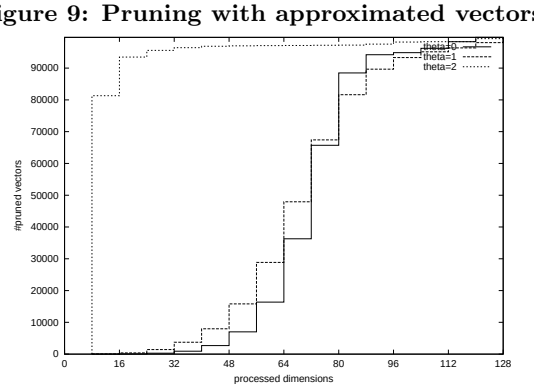


Figure 11: Effects of skew on the weights (E_v)

mensionality, although not dramatically. With the increase of dimensionality the search space becomes less appropriate for nearest neighbor search: the distance ratio between the nearest and the furthest vector from a random point in space drops [3]. In our case, the scores of the best-k matches become lower and more indistinguishable as dimensionality increases. Nevertheless, experiments on synthetic datasets with well-defined clusters (see also Section 7.5) have shown that the pruning efficiency of the proposed method does not degrade with dimensionality.

7.4 Search performance improvement

The practical value of BOND is evaluated by measuring the response time of the implementation described before. The measurements reported have been acquired from executing 100 sample queries on the dataset with 166-dimensional HSV histograms, using both similarity metrics with pruning criteria H_q, H_h, and E_v. These response times are compared to the times measured with an optimized implementation of sequentially scanning a single table with all vectors. For each vector $\mathbf{v}$, sequential scan computes its similarity with $\mathbf{q}$ and adds it to a heap of the k-best matches so far.[6] The two versions of this method for histogram intersection and Euclidean distance are denoted as SS_H and SS_E, respectively.

Table 3 presents statistics for the measured response times; all reported times are in milliseconds. H_q is the best pruning heuristic for histogram intersection due to its simplicity and very good pruning efficiency. Although H_h prunes more effectively than H_q, the difference is not large enough for the additional bookkeeping to pay off. Both criteria reduce significantly the cost of sequential scan, up to an order of magnitude. E_v is not that efficient in terms of CPU-cost, mainly due to the relatively complex bounds of $\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+)$, which add computational overhead. In addition, it prunes less space on the average as shown in Figure 5. However, this method is still much faster than sequential scan.

Table 3: BOND vs. sequential scan; times in msec.

method	min	max	average	median
H_q	21	121	40	35
H_h	27	140	50	45
SS_H	215	242	229	231
E_v	29	224	108	108
SS_E	181	190	183	184

As discussed before, related research has shown performance improvements using compressed approximations of the feature vectors, for both sequential search (VA–File) as well as space partitioning methods. A similar improvement can be expected using this approximation technique in combination with BOND. Figure 9 plots the average pruning of H_q on exact and compressed dimensional fragments (an 8-bit approximation of each double coefficient is used per dimension). Pruning on the compressed fragments finishes with 1000 candidates after processing 72 dimensions and follows a similar trend as using the original fragments. Table 4 shows the cost of applying H_q on the compressed fragments, compared to sequentially scanning the equivalent VA–File [22]. Essentially, both approaches return the same sets of candidates, that need to be checked in the additional refinement step. A runtime comparison between BOND applied to approximations (using H_q) and a sequential scan of the equivalent VA–file [22] results in an overall improvement of a factor 3-5 in favour of BOND (on the 166-dimensional dataset). This experiment demonstrates further that the benefit obtained by compressing the feature vectors is orthogonal to our technique: in both cases, the results using approximate feature vectors are roughly two to three times better than the results without compression.

Table 4: Experiments with approximations of the original feature vectors; times in milliseconds.

method	min	max	average	median
filter step H_q	13	96	28	21
filter step SS_H	65	107	81	81
refinement step	1	34	3	2

The improvement of BOND over the VA–file is expected to decrease with dimensionality on the same image set, because the k-NN search becomes less meaningful (as commented in Section 7.3). Experimental results using the A-tree access structure given in [17] show an improvement of factor 4 on a very similar dataset, also decreasing with higher dimensionality. This indicates that BOND offers competitive performance improvements when compared to advanced tree-based indexing structures, while being much simpler to implement. Additional advantages of the proposed technique include the ability to process other variants of k-NN queries efficiently, like weighted and multi-feature queries (as discussed in Section 8).

7.5 Effects of data skew

While BOND seems effective for content-based image retrieval, a natural question arises: is it good for generic k-NN search, and if so, under which conditions? Notice that in the color histogram datasets examined so far the values of a random vector follow a skewed Zipfian distribution (see Figure 2). In such cases, an ordering that considers the most skewed dimensions in the query first, is expected to have nice pruning effect. We also expect our approach to do well when the data are clustered in the high-dimensional space, and k-NN search is meaningful [3]. On the other hand, if there is no skew in the vector values, the best partial solutions after less than half of the dimensions may still turn out to be the worst solutions overall. So, we do not expect the pruning effect to be significant when the average values of a vector follow a uniform distribution, i.e., they have equal 'weight' for the specific vector.

We generated synthetic datasets in order to evaluate the assertion that skew favors pruning. All datasets contain 100,000 128-dimensional vectors, defined in a unit hypercube. In this hypercube, 1000 points define the centers of the clusters; 95% of the generated vectors belong to some random cluster, whereas 5% of them take random values (noise). The distance from each vector to the cluster where it belongs to is defined by a Gaussian distribution around

[6] We also tried a more sophisticated approach, where the partial score of $\mathbf{v}$ was regularly compared to the top-k score found so far and search was abandoned for $\mathbf{v}$ if it was found impossible to reach that score. However, this version of sequential scan was slower on the average (due to the overhead of comparisons, and its incapability to consider first the most promising dimensions).

the cluster's center. This dataset has the nice properties that make NN-search meaningful [3]; the k-nearest neighbor of a point that falls into a cluster is close compared to points from other clusters, and there is a small percentage of outliers that do not fall into a cluster and essentially do not have 'meaningful' nearest neighbors. The coordinates of the clusters' centers follow a Zipfian distribution. If the skew parameter θ is 0 the centers follow a uniform distribution. The larger θ is the more skewed the cluster centers are.

Figure 10 shows the average pruning efficiency of E_v for various values of the skew parameter θ. As expected, the efficiency of BOND depends on the skew in the data set: data skew favors pruning, but the technique is not efficient when the centers of the clusters are uniform. In real-life applications however, where datasets are skewed (like the color histograms), we can expect good results from the proposed method. Also, weighted search puts implicit skew in data and therefore increases the value of our approach, as is discussed in the next section.

8. Weighted Search and Multi-Feature Queries

We have already seen how BOND evaluates efficiently the basic type of k-NN queries: given a collection of high dimensional vectors $\mathcal{X}$ and a query vector $\mathbf{q}$, find the k most similar vectors $\mathbf{x} \in \mathcal{X}$ with respect to a similarity metric $Sim(\mathbf{x}, \mathbf{q})$. This section demonstrates how the proposed storage scheme and query processing techniques are also suitable for two common extensions of this basic query type, often found in practical applications. In the first extension, called *weighted* k-NN query, weights are assigned on the dimensions of the query vector. Queries of second extension, called complex or *multi-feature* queries, have multiple components, whose similarity is tested against various feature sets, possibly using a different metric for each component.

8.1 Weighted Queries

So far, all dimensions of the query vector have been assumed equally important. In many applications, however, users may assign different *weights* to the query features. This is especially true if the dimensions in the feature space can be interpreted as concepts which are clear in the user's mind. Even when this case does not apply, relevance feedback mechanisms often put weights at dimensions, in order to refine the query results according to what seems to be the user's request [12].

The Appendix describes how to derive the required bounds for weighted Euclidean search. A non-uniform distribution of weights introduces skew in the transformed space, so BOND is expected to perform well. The following experiment validates its efficiency on weighted queries. We used the synthetic dataset described in Section 7.5 with θ=0, which yields the worst-case pruning of E_v (the cluster centers are uniformly distributed). Figure 11 shows the pruning efficiency for various values of the skew on weights. Observe that efficiency improves only if the skew in the weights is large: 10% of the dimensions should get more than 90% of the weights. In real-life situations we believe skewed weighting occurs frequently, a fact that makes our approach especially useful.

An inherent advantage of vertical fragmentation not discussed so far is that having stored *independently* the vector coordinates, we can process seamlessly k-NN queries in *any* dimensional subspace. For example, if the user or the query processor decides to consider any arbitrary set of color bins, BOND avoids accessing information irrelevant to those dimensions. This flexibility cannot be achieved in tree structures that index the space using *all* dimensions. As stated in [11], it is typical for queries in high dimensional spaces to be meaningful only for a subset of dimensions, which may well be different for each query. Notice that k-NN search in a dimensional subspace is a special case of weighted search, where all weights in the search dimensions are positive and equal, and all weights in irrelevant dimensions are zero.

8.2 Multi-feature Queries

In image database systems it is common for a user to ask queries related to more than one visual attribute of the image. For instance, a query may ask for images that are similar to image A in color and to image B in texture. The similarity between two images with respect to multiple attributes is defined as an aggregate of the individual similarities for each attribute.

Previous work in this area [7, 9] has focused on the efficient merging of multiple streams that contain the most similar results with respect to each query component. Stream merging, although generic enough to be used for any search problem, has certain disadvantages. While k defines the number of desired global results and not the output of each stream, it is difficult to determine a priori how many objects are required from each stream in order to identify the k most similar objects. Therefore, the search algorithm requests incrementally nearest neighbors from each stream, until the global stopping condition has been satisfied. The number of candidates selected from each stream affects however the performance of the search algorithm used within that stream; the larger the number of retrieved objects, the higher the retrieval cost (cf. Figure 6). Furthermore, combining different streams requires random accesses in order to compute the global similarity of the best objects in one stream, which are not present in the other(s).

If the dimensional data are fragmented vertically in each feature collection, the separate ranking and merging steps can be integrated. BOND can compute the best global k-matches in one step by simultaneously applying dimension-wise search in all relevant feature collections. This demonstrates another powerful feature of vertical fragmentation; BOND can be applied for numerous types of complex queries including, (i) queries with weights assigned to the various components, (ii) queries having different similarity metrics for each component, provided that the global similarity is well defined from the merging of the individual ones, and (iii) queries with various aggregate functions, including arithmetic ones [9] like average and fuzzy logic [7, 15], like min and max.

As an example, consider a complex query where we are looking for k images with the best weighted average color similarity with A and texture similarity with B. We consider the union of color and texture dimensions as one large set of dimensions. For such dimension, the partial similarity of a candidate object with the query object is defined by the actual partial similarity (using the corresponding metric, depending on which feature set this dimension belongs) multiplied by the corresponding weight. The worst/best case pruning bounds are calculated using the global partial similarities and the query values in the remaining dimensions. The optimization methods for ordering of dimensions and

choice of m are also applicable here: the most skewed query dimensions (after normalization using the weights) are chosen first. Fuzzy logic aggregates can be handled in a similar way. The pruning bounds are calculated using the partial similarities and the query values in the remaining dimensions in each component and the aggregate function. For instance, the worst case global similarity using the min aggregate can be computed by its worst case similarities at each component (with potentially different similarity functions).

Optimizing multi-feature queries using BOND is an interesting research topic. Preliminary experimental results have shown that synchronized search in multiple feature collections is a promising technique for multi-feature queries. For H_q, we compared this method with one that uses H_q in each individual dataset and merges the results using the algorithm of [9]. The experiment was performed on two synthetic datasets, representing different feature collections with 100,000 objects and having dimensionality 64 and 128, respectively. The datasets were generated using the methodology described in Section 7.5, i.e., they contain clusters of points, and the query instances were taken from the datasets. On the average, the synchronized search method was found 20% faster than stream merging when the aggregate function is average and 70% faster when the aggregate function is min. Given the fact that the k used to search each individual stream was the optimal (which is unknown in reality), the performance improvement of synchronized search over stream merging can only be larger in practice.

9. Conclusions

This paper provides insight into the potential benefits of a different storage scheme for high-dimensional data, that supports efficient k-NN search. By fragmenting a collection of 166-dimensional image histograms vertically into 166 distinct tables, significant performance gains have been demonstrated using the proposed query processing techniques. Even more performance gains are achieved if some dimensions are more important than others, making the approach particularly suited for interactive image retrieval applications.

BOND is simple and introduces negligible storage overhead. It is a novel application of the well-known 'push-select-down' heuristic, turning a selection predicate captured by a complex mathematical formula (the metric) into selection predicates on the individual dimensions. The approach is easily integrated in a relational database system, without interfering with the database optimizer. Like most high-dimensional k-NN search methods, BOND is efficient on skewed data where search is meaningful. Unlike tree-based indexing methods, its efficiency is versatile to various types of k-NN queries, including weighted and multi-feature queries, and to different similarity metrics, while it can be applied on a single, non-redundant data representation.

Indeed, the vertical fragmentation of high dimensional data opens the road to novel processing techniques of queries on high dimensional data. A promising direction of future work is to develop new techniques for other search problems in high dimensional spaces (e.g., clustering), when applied to dimension-wise decomposed data. Another extension of this work is the application of the same method to searching large audio and video databases, especially for the combination of various search strategies. Using branch-and-bound over different feature spaces simultaneously allows for advanced query optimization techniques.

The quality of a k-NN query is defined in [11] to be parametric to subsets of the full dimensional space. In our framework, queries with significant skew seem to correspond to *useful* queries, a property that may be exploited as a measure of the quality of a query vector to describe information needs. In other words, the search quality may not be simply a parameter of a dimensional subset, but depend on a distribution of weights on all dimensions. As already pointed out in Section 8, subspace search is a special case of weighted search, and the definition of [11] becomes a special case of our concept of usefulness. We intend to investigate the theoretical soundness of this generic definition, as well as its practical value in problems like clustering.

Acknowledgements

We would like thank to Jan-Mark Geusebroek, for inspiring this research. Many thanks to our colleagues Menzo Windhouwer, Peter Bosch for their help with the datasets used in this paper, and to Peter Boncz for believing in this work and keeping us going in spite of several set-backs.

References

[1] M. Annamalai, R. Chopra, S. DeFazio, and S. Mavris. Indexing images in oracle8i. In *Proc. of ACM SIGMOD Int'l Conference*, 2000.

[2] S. Berchtold, C. Böhm, H. Jagadish, H.-P. Kriegel, and J. Sander. Independent quantization: An index compression technique for high-dimensional spaces. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, 2000.

[3] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proc. of Int'l Conference on Database Theory (ICDT)*, 1999.

[4] P. Boncz and M. Kersten. MIL primitives for querying a fragmented world. *The VLDB Journal*, 8(2):101–119, 1999.

[5] G. Copeland and S. Koshafian. A decomposition storage model. In *Proc. of ACM SIGMOD Int'l Conference*, 1985.

[6] http://www.corel.com.

[7] R. Fagin. Fuzzy queries in multimedia database systems. In J. Paredaens, editor, *Proceedings of the Seventeenth ACM Symposium on Principles of Database Systems (PODS '98)*, 1998.

[8] C. Faloutsos and K.-I. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc. of ACM SIGMOD Int'l Conference*, 1995.

[9] U. Güntzer, W.-T. Balke, and W. Kiessling. Optimizing multi-feature queries for image databases. In *Proc. of VLDB Conference*, 2000.

[10] A. Guttman. R-trees: a dynamical index structure for spatial searching. In *Proc. of ACM SIGMOD Int'l Conference*, 1984.

[11] A. Hinneburg, C. Aggarwal, and D. Keim. What is the nearest neighbor in high dimensional spaces? In *Proc. of VLDB Conference*, 2000.

[12] Y. Ishikawa, R. Subramanya, and C. Faloutsos. Mindreader: Querying databases through multiple examples. In *Proc. of VLDB Conference*, 1998.

[13] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and

Z. Protopapas. Fast nearest neighbor search in medical image databases. In *Proc. of VLDB Conference*, 1996.

[14] P. E. O'Neil and D. Quass. Improved query performance with variant indexes. In *Proc. of ACM SIGMOD Int'l Conference*, 1997.

[15] M. Ortega, Y. Rui, K. Chakrabarti, S. Mehrotra, and T. Huang. Supporting similarity queries in MARS. In *Proc. of ACM Multimedia Conference*, 1997.

[16] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. of ACM SIGMOD Int'l Conference*, 1995.

[17] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: an index structure for high-dimensional spaces using relative approximation. In *Proc. of VLDB Conference*, 2000.

[18] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proc. of ACM SIGMOD Int'l Conference*, 1998.

[19] J. Smith and S.-F. Chang. Tools and techniques for color image retrieval. In *Storage & Retrieval for Image and Video Databases IV*, volume 2670 of *IS & T/SPIE Proceedings*, 1994.

[20] J. Smith and S.-F. Chang. Automated image retrieval using color and texture. Technical Report 414-95-20, Columbia University CTR, New York, 1995.

[21] M. J. Swain and D. H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[22] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. of VLDB Conference*, 1998.

APPENDIX

A. Pruning Rules for Weighted Search

The simple and adaptive nature of branch-and-bound makes the definition of pruning rules for weighted search straightforward. Definition 2 becomes:

DEFINITION 3. *The **weighted squared Euclidean distance** between two vectors* $\mathbf{v}$ *and* $\mathbf{q}$ *of dimensionality* N *is defined as follows:*

$$\delta_w(\mathbf{v},\mathbf{q}) = \sum_{i=1}^{N} w_i (v_i - q_i)^2 \tag{13}$$

Each dimension is assigned a weight w_i which reflects its importance in the query. If $\sum_{i=1}^{N} w_i = N$, equation 3 defines the similarity of the two vectors. Figure 13 visualizes the special case where $m = N-2$; basically, each dimension is stretched or shrinked with a different factor. In the presence of weights, the upper bound of $\mathcal{S}(\mathbf{v}^+,\mathbf{q}^+)$ becomes:

$$\mathcal{S}(\mathbf{v}^+,\mathbf{q}^+) \le \sum_{i=m+1}^{l-1} w_i q_i^2 + w_l(u_l - q_l)^2 + \sum_{i=l+1}^{N} w_i (1-q_i)^2 \tag{14}$$

Equation 14 assumes that the values of $\mathbf{q}^+$ are ordered such that $w_i q_i^2 \ge w_{i+1} q_{i+1}^2, \forall i > m$. Values l and u_l are defined as in Lemma 1. The lower bound is similarly defined

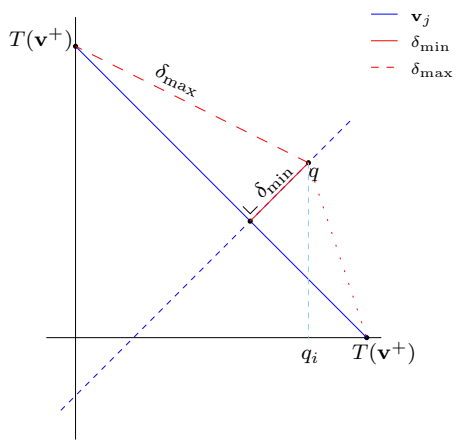


Figure 12: The lower- and upper bound on the Euclidean distance between q and any possible v visualized for $m = N-2$ and $T(\mathbf{v}^+) \le 1$.

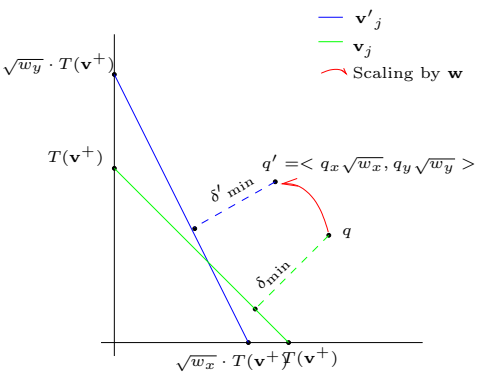


Figure 13: The lower- and upper bound on the Euclidean distance with weighting vector w, visualized for $m = N-2$.

by extending equation 12 as follows:

$$\mathcal{S}(\mathbf{v}^+,\mathbf{q}^+) \ge \frac{\prod_{i=m+1}^{N} w_i}{\sum_{j=m+1}^{N} \prod_{i=m+1, i\ne j}^{N} w_i} (T(\mathbf{v}^+) - T(\mathbf{q}^+))^2 \tag{15}$$

The proofs of equations 14 and 15 can be derived after squaring the maximum and minimum distance of the transformed vector $\mathbf{q'}^+, q'^+_i = \sqrt{w_i} q_i^+, \forall m < i \le N$ from the hyperplane defined by all possible distributions of $T(\mathbf{v}^+)$ to the dimensions after m.