INformation Systems

INS

Efficient image retrieval by exploiting vertical fragmentation

A.P. de Vries, N. Mamoulis, N.J. Nes, M.L. Kersten

REPORT INS-R0109 OCTOBER 31, 2001

ISSN 1386-3681

Efficient Image Retrieval by Exploiting Vertical Fragmentation

Arjen P. de Vries

Nikos Mamoulis

Niels J. Nes

Martin L. Kersten
{arjen,nikos,niels,mk}@cwi.nl

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT
In content-based retrieval systems, the goal of similarity search is to identify the k most similar images to a given example. Images are represented and queried by high-dimensional feature vectors encoding dominant characteristics like color and texture. We propose an efficient similarity search method that is robust to dimensionality and has optimal space complexity. Our approach fragments the feature vectors vertically, and computes the similarity of all images dimension by dimension. The innovation lies in gradually removing images that cannot participate in the response set. We show how to apply this scheme for two common similarity metrics, namely histogram intersection and euclidean distance. The implementation of our algorithm in Monet illustrates that core database technology supports image retrieval well, without special extensions. Finally, we report the effectiveness of our approach on real and synthetic data sets, and show significant improvements in response time yielded.

1998 ACM Computing Classification System: H.2.2 [Database Management]: Physical Design – Access Methods; H.2.8 [Database Management]: Database Applications – Image Databases, H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing.
Keywords and Phrases: Nearest Neighbour Search, High-dimensional Search
Note: Work carried out under the MIA project

1. Introduction

Image retrieval is an important application of high-dimensional querying. The dimensions correspond to features such as color bins, texture patterns (like grayness or smoothness), or transformations capturing information about the image structure (e.g., local colors, shapes) [ACDM00]. The image content is then abstracted into a tuple of values, one for each feature. This tuple is called *feature vector*, or, in the case of color bins, *histogram*. The dimensionality of these feature vectors is large, i.e., at least 64.

In contrast to exact retrieval in conventional databases, image retrieval asks for objects that are *similar* to a query vector [Fag98]. Usually, we are interested in the k most similar (or dissimilar) images only. *Similarity* between images is defined by a metric applied on the corresponding feature vectors. A popular metric, called *histogram intersection* [SB91], sums the overlap between the two histograms in each dimension. 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.

A naive similarity search method compares the query vector to all image vectors, while maintaining an array with the best k. Unfortunately, this approach is impractical in large image collections:

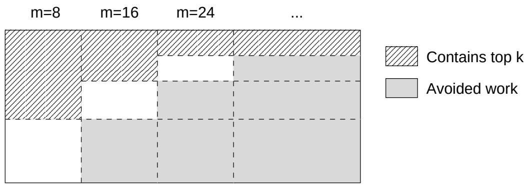


Figure 1: Our approach visualized. The first m columns are scanned, and the best partial scores are computed. Images with a best-case score that is smaller than the worst-case score of the k-th most similar image are pruned. This process is repeated until the remaining set contains exactly k images, or all dimensions have been processed.

computing the distance between two histograms can be expensive, and scanning all histograms accesses a very large data set.

In this paper, we propose a simple, yet efficient solution to the image similarity search problem. Our approach is based on a decomposed storage scheme for the feature vectors, or *vertical fragmentation*: we maintain a separate table for each dimension, containing the coefficients *of all images* in that dimension.[1] The similarity between the query vector and each of the image vectors is accumulated by scanning these tables one-by-one. After processing a few dimensions, we know the *partial* similarity between the query and all images. We then exploit lower and upper bounds on the final similarity of the k-most similar images to accelerate query processing. Images that can never reach the scores of the k best images so far are safely pruned from search, and the corresponding rows are not considered when processing the remaining dimensions. By applying this pruning process iteratively, we reduce the candidate set such that the last stages are performed very cheaply. The resulting search process is visualized in Figure 1.

The advantages of our proposal are summarized as follows:

- It avoids a large number of computations compared to a full sequential scan. In some cases, pruning is so effective that the k most similar images are identified after processing less than 10% of the dimensions.
- 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* high-dimensional search problem, see [BGRS99]), in contrast to most search methods.
- It is a novel technique, orthogonal to previous approaches based on compression of feature vectors (such as [WSB98]).

An interesting property of our approach is that it can be expressed in standard relational algebra; it does not require ADTs (user-defined types) or advanced indexing structures. We implemented our approach in the *Monet* database system [BK99]. Our experimental results demonstrate that a pure relational approach performs better than an object-relational approach with ADTs.

The remainder of this paper is organized as follows. Section 2 summarizes the notation used throughout the paper and provides definitions of histogram intersection and Euclidean distance. Section 3 presents the generic form of our approach and presents pruning rules that are used by our methodology for both studied similarity metrics. Section 4 discusses techniques to further optimize the search process, based on the data and query distribution. Section 5 describes the implementation of this approach in Monet, followed by an experimental evaluation with real and synthetic datasets in Section 6. The next section discusses the efficiency of our approach in the presence of weights on the query dimensions. Section 8 discusses related work, and demonstrates that branch-and-bound is a new technique, orthogonal to existing approaches for high-dimensional search. Finally, Section 9 summarizes our contributions with this paper.

[1]Decomposed storage of 'normal' relations was first proposed by [CK85].

2. Definitions

The notation used throughout the paper is summarized in Table 1. Some symbols are implicitly parametrized by an index m or a collection $\mathcal{X}$.[2] We also define the two image search problems for which we present optimized search algorithms and experiments.

2.1 Notation

Let $\mathcal{X}$ be a collection of sequences $\mathbf{x} = x_1 \ldots x_N$. 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, $\mathbf{x}^- = x_1 \ldots x_m$ and $\mathbf{x}^+ = x_{m+1} \ldots x_N$. 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 $[\mathbf{x}_1, \ldots, \mathbf{x}_{|\mathcal{X}|}]^T$, 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 $\mathbf{X}$, $\mathbf{H}$, $\mathbf{V}$.

We often apply an operation to each element of a collection. $\mathbf{S} = \mathcal{S}(\mathbf{X})$ denotes the result of elementwise 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., $\mathbf{X}^- = [\mathbf{x}_1^-, \ldots, \mathbf{x}_{|\mathcal{X}|}^-]^T$ and $\mathbf{S}^- = \mathcal{S}(\mathbf{X}^-)$.

We denote by $\mathbf{S}^+_{\min}$ and $\mathbf{S}^+_{\max}$ 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: Notation.

$\mathbf{x}$	Sequence $x_1 \ldots x_N$.
$\mathbf{x}^-$	Sequence $x_1 \ldots x_m$.
$\mathbf{x}^+$	Sequence $x_{m+1} \ldots x_N$.
$T(\mathbf{x})$	$\sum_{i=1}^N x_i$
$T(\mathbf{x}^-)$, $T(\mathbf{x}^+)$	$\sum_{i=1}^m x_i$, $\sum_{i=m+1}^N x_i$
$\mathcal{X}$	Collection of sequences $\{\mathbf{x}\}$.
$\mathbf{X}$	Vector of sequences $[\mathbf{x}_1, \ldots, \mathbf{x}_{|\mathcal{X}|}]^T$.
$\mathcal{S}(\mathbf{x})$	Aggregate function $\mathcal{S} : \mathbf{x} \to \mathbb{R}$.
$\mathbf{S}$	Vector $\mathcal{S}(\mathbf{X})$.
$\mathbf{S}^+_{\max}$, $\mathbf{S}^+_{\min}$	Elementwise 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 $\mathbf{S}_{\min}$, $\mathbf{S}_{\max}$.

2.2 Histogram Intersection

Let $\mathcal{H}$ be a collection of image histograms $\mathbf{h}$. These histograms are N-dimensional vectors that have been normalized[3] as follows:

$$\forall \mathbf{h} \in \mathcal{H} : T(\mathbf{h}) = 1 \tag{2.1}$$

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

$$Sim(\mathbf{h}, \mathbf{q}) = \sum_{i=1}^{N} \min(h_i, q_i) \tag{2.2}$$

Using histogram intersection assumes that the different dimensions are uncorrelated. This metric was reported in [SC95] 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

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

[3]The normalized histogram may be considered as a probability distribution function.

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.

2.3 Euclidean distance

Let $\mathcal{V}$ be a collection of N-dimensional feature vectors $\mathbf{v}$ in the unit hyperbox:

$$\forall \mathbf{v} \in \mathcal{V} : 0 \leq v_i \leq 1 \tag{2.3}$$

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

$$\delta(\mathbf{v}, \mathbf{q}) = \sum_{i=1}^{N} (v_i - q_i)^2 \tag{2.4}$$

The actual Euclidean distance is the square root of $\delta(\mathbf{v}, \mathbf{q})$. We use squared distance δ in order to reduce 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:

$$Sim(\mathbf{v}, \mathbf{q}) = 1 - \sqrt{\frac{1}{N}\delta(\mathbf{v}, \mathbf{q})} \tag{2.5}$$

3. Branch-and-Bound

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 the following *branch-and-bound* alternative to improve efficiency:

Algorithm 2 *Branch-and-Bound($\mathcal{X}$,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 $x_{m+1} \ldots x_N$. 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 $\mathbf{S}_{\min}[i] > \kappa_{\max}$.

3.1 Rules and a pruning strategy 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 2.2. We first break the sum into partial sums $\mathcal{S}(\mathbf{h}^-, \mathbf{q}^-)$ and $\mathcal{S}(\mathbf{h}^+, \mathbf{q}^+)$:

$$\mathcal{S}(\mathbf{h}, \mathbf{q}) = \underbrace{\sum_{i=1}^{m} \min\{h_i, q_i\}}_{\mathcal{S}(\mathbf{h}^-,\mathbf{q}^-)} + \underbrace{\sum_{j=m+1}^{N} \min\{h_j, q_j\}}_{\mathcal{S}(\mathbf{h}^+,\mathbf{q}^+)} \tag{3.1}$$

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

$$\mathcal{S}(\mathbf{h}^+, \mathbf{q}^+) \leq \sum_{j=m+1}^{N} q_j = T(\mathbf{q}^+) = 1 - T(\mathbf{q}^-) \tag{3.2}$$

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:

$$\mathcal{S}(\mathbf{h}_i^-, \mathbf{q}^-) + (1 - T(\mathbf{q}^-)) < \kappa_{\min} \tag{3.3}$$

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 $\mathbf{h}$:

$$\begin{aligned} \mathcal{S}(\mathbf{h}^+, \mathbf{q}^+) \leq \min\{T(\mathbf{h}^+), T(\mathbf{q}^+)\} &= 1 - \max\{T(\mathbf{h}^-), T(\mathbf{q}^-)\} && (3.4) \\ \mathcal{S}(\mathbf{h}^+, \mathbf{q}^+) = \sum_{i=m+1}^{N} \min\{q_i, h_i\} &\geq \sum_{i=m+1}^{N} \min\{q_{\min}, h_i\} && \\ &\geq \min\{q_{\min}, T(\mathbf{h}^+)\} = \min\{q_{\min}, 1 - T(\mathbf{h}^-)\} && (3.5) \end{aligned}$$

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 $T(\mathbf{h}^+)$.

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 3.4 and 3.5 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:

$$\underbrace{\mathcal{S}(\mathbf{h}_i^-, \mathbf{q}^-) + 1 - \max\{T(\mathbf{h}_i^-), T(\mathbf{q}^-)\}}_{\mathbf{S}_{\max}[i]} < \kappa_{\min} \tag{3.6}$$

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 images.

3.2 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 best matches ($k = 3$). The three best matches are $\{\mathbf{h}_3, \mathbf{h}_5, \mathbf{h}_7\}$, which could

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

$\mathbf{H}$	$h \in \mathcal{H}$	$\mathbf{S}^-$	$\mathbf{S}_{\min}$	$\mathbf{S}_{\max}$	$\mathcal{S}$
$\mathbf{h}_1$	$< 0, 0.1, 0, 0.9 >$	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

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 $\mathbf{S}^-$). Because 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.

Using rule H_h, we take advantage of the information in $\mathbf{h}^-$ as well. We compute columns $\mathbf{S}_{\min}$ and $\mathbf{S}_{\max}$ as shown in the table, determine a (higher) $\kappa_{\min} = 0.75$ from $\mathbf{S}_{\min}$, and select the histograms 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 we should expect good pruning. Our experiments in Section 6 demonstrate that this branch-and-bound strategy works on real data sets indeed.

3.3 Rules and a pruning strategy for squared 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 2.4, i.e., it is the squared Euclidean distance[4]. Unlike histogram intersection, we are interested in the objects with the smallest values in $\mathbf{S}$. Like before, we start with pruning rule E_q that depends on 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:

$$\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+) \leq \sum_{i=m+1}^{N} \max\{q_i, 1 - q_i\}^2 \tag{3.7}$$

We now define stricter bounds for $\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+)$, assuming that we know $T(\mathbf{v}^+)$.

Lemma 1 *Assume that the values of $\mathbf{q}^+$ are in decreasing order, i.e., $q_i > q_{i+1}, \forall i > m$. Let $f = \lceil T(\mathbf{v}^+) \rceil$. Let $l = m + 1 + f$. The following inequality defines the upper bound of $\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+)$:*

$$\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+) \leq \sum_{i=m+1}^{l-1} q_i^2 + (T(\mathbf{v}^+) - f - q_l)^2 + \sum_{i=l+1}^{N} (1 - q_i)^2 \tag{3.8}$$

[4]The reason we do not use the metric defined by equation 2.5 is that it is far more complex and gives essentially the same result.

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, we can prove inequality 3.8 for every $m < N - 2$. □

Lemma 2 *The following inequality defines the lower bound of* $\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+)$*:*

$$\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+) \geq \frac{(T(\mathbf{v}^+) - T(\mathbf{q}^+))^2}{N - m} \tag{3.9}$$

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 10 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.[5] Notice that we need $T(\mathbf{v}^+)$ 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 $T(\mathbf{v})$.

4. Optimization Issues

4.1 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 images 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 $\mathcal{H}$ can guide the definition of a good order.

Without additional knowledge about the distribution in $\mathcal{H}$, we expect condition H_q in histogram intersection to prune the candidate set most succesfully, if the right-hand side of the inequality has the highest value, i.e., we process the dimensions in decreasing order of scalars in $\mathbf{q}$. In other words, had $\mathbf{q}$ in the example of Section 3.1 been $< 0.15, 0.1, 0.7, 0.05 >$, we should consider dimensions 3 and 1 to compute the partial scores $\mathbf{S}^-$. Notice that this ordering is not necessarily the optimal.

For rules H_h and E_v, we need to consider also the distribution of values in $\mathcal{H}$. Figure 2 shows statistics from a real dataset containing 166-dimensional color histograms of 59619 images from the Corel collection [Cor]. The left diagram plots the mean value of each bin and the right the distribution of the values in a histogram. 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 upper 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, we would consider a different processing order taking the most skewed dimensions first.

[5]Let $\mathit{diff} = (T(\mathbf{v}^+) - T(\mathbf{q}^+))/(N-m)$ and assume that the values in $\mathbf{q}^-$ are in decreasing order (like in the definition of lemma 1). In the special cases (*i*) $\mathit{diff} < 0, |\mathit{diff}| < q_N$ and (*ii*) $\mathit{diff} > 0, q_{m+1} + |\mathit{diff}| > 1$, we use a stricter lower bound than inequality 3.9. Details are omitted for sake of readability.

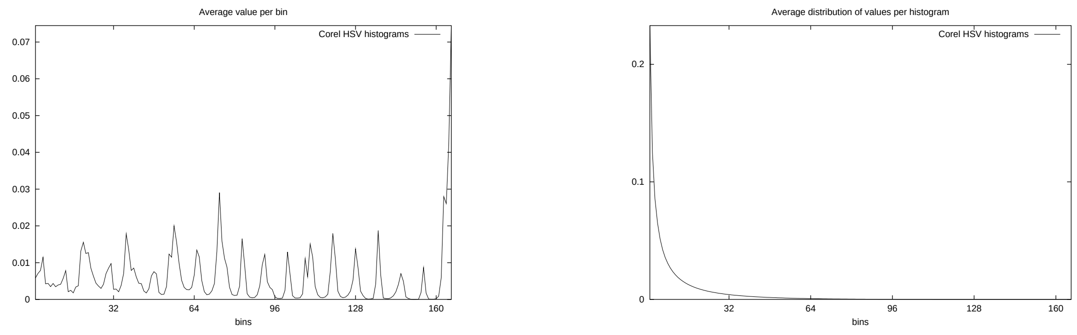


Figure 2: Statistics from a real dataset

4.2 How many dimensions count?

Taking 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 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.

We can optimize the choice of m by taking data statistics and the query vector into account. For instance, H_q will not prune any image until the right-hand side of equation 3.3 is positive. This can happen only when $T(\mathbf{q}^-) > 0.5$, since $\kappa_{\min} \leq T(\mathbf{q}^-)$. Thus, we should not attempt pruning 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.

5. Implementation

We implemented the proposed branch-and-bound strategy in the Monet database system [BK99]. Its data model is based on Binary Association Tables (BATs); the first column of a BAT is called its head, the other column its tail. Monet supports a flexible query language on BATs, called the Monet Interface Language (MIL). It provides a variety of highly efficient implementations for common algebraic operators (join, select, etc.), using strategies such as positional lookup, hash lookup, or binary search. Administration of so-called 'properties' propagates fragmentation information about BATs through operators, avoiding unnecessary joins as much as possible.

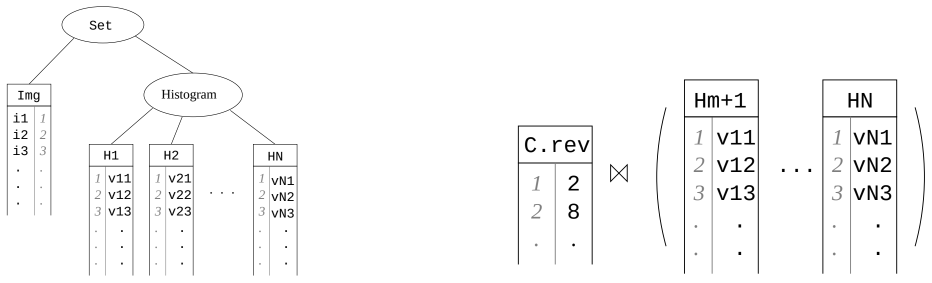
5.1 Basic algorithm

We vertically fragment the collection of histograms $\mathcal{H}$ into N BATs `Hi` of length $|\mathcal{H}|$, storing tuples with an 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, Monet's support for so-called 'virtual oids' 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[6].

Algorithm 2 with rule H_q is expressed in (pseudo-)MIL as follows:

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



(a) vertically fragmented histograms (b) step 3 of the algorithm visualized

Figure 3: Implementation in Monet

```
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 is automatically available for each MIL operator `f`. A multi-join map performs an implicit equi-join on the head columns of multiple BATs, and executes its operator (`f`) on the tail arguments of the join result. The `const` keyword tells Monet to use 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 Monet knows that these tables are aligned, the join-algorithm chosen is the positional join (with negligible cost).

Step 2 computes the maximum bound derived in rule H_q, and selects the histogram identifiers of the candidate set: the 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`; its implementation is discussed in Section 5.2. The `uselect` operator is the 'unary range select', which returns the head values of tuples from `Smin` with tail values in the specified range, and a tail value set to a densely ascending range of (virtual) oids. Finally, step 3 reduces the remaining tables of $\mathbf{H}^+$ to the candidate set (Figure 3_b, 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 an alternative, we use a `uselect` that creates a bitmap index on the histogram identifiers to representing the pruned candidate set. After some iterations, when the candidate set starts to reduce significantly, we switch to the 'standard' positional joins approach, resulting in much smaller base tables for the subsequent iterations.

The actual MIL-code executed in Monet matches this pseudo-code closely. The real code is parametrized by variable m (using an imperative loop construct), and represents $\mathbf{q}$ as a table and $\mathbf{H}$ as a nested table, allowing to choose the order in which dimensions are processed more flexibly.

5.2 Fast detection of the k-th element

Each time a dimension that is a multiple of m has been processed, the k-th element with respect to the similarity metric needs to be identified (the `kfetch` operator in the pseudo-MIL code). A naive way to locate the k-th element is to sort the current n candidates, with associated cost $O(n \log n)$. To improve this situation, we use a priority queue implemented as a heap. A *min-heap* is a structure that

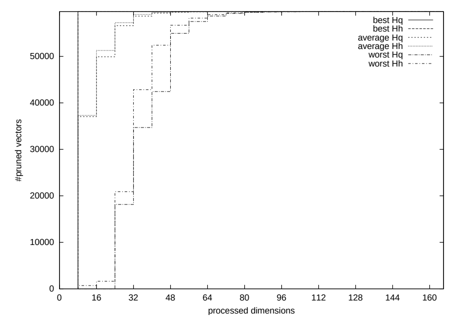


Figure 4: Pruning effects of H_q and H_h

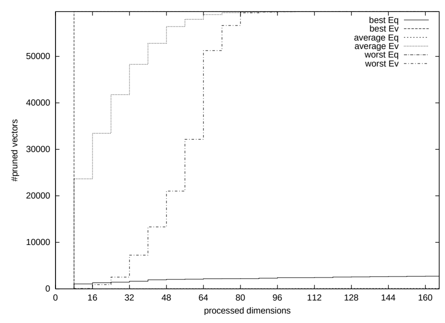


Figure 5: Pruning effects of E_q and E_v

answers min queries in $O(1)$ time, and has $O(\log k)$ update cost. Each of the n candidates is compared with the minimum element of the heap (current k-th largest element) and replaces it if found larger. The heap is then updated. At the end of this procedure, with worst-case cost $O(n \log k)$, the heap contains the query results. In the case of Euclidean distance, a max-heap is used.

5.3 Updates

By their nature, large image databases are 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 by Copeland and Koshafian in [CK85], update performance will approximate the efficiency of updates in a normal relational storage scheme, especially when using differential files and performing mainly batch updates. This observation is consistent with our experience with data management in Monet. As a final comment, the bitmap used in step 3 marks deleted image histograms, until periodic reorganization of the collection.

6. Experiments

In this section, we evaluate the pruning efficiency and run-time cost for the various adaptations of Algorithm 2. First, we use a real dataset to verify the pruning effects of the four criteria. We then experiment on the effects of k and ordering of dimensions on performance. The next experiment validates the robustness of branch-and-bound to dimensionality. A run-time cost comparison between pruned search and sequential scan follows. Finally, we check the pruning efficiency of our method on synthetic datasets with varying data distributions. Unless otherwise stated, in all experimental instances k was set to 10. All experiments were run on a normal personal computer, with a 400 MHz Celeron and 512 MB main memory.

6.1 Pruning effects of the criteria

We applied the histogram intersection algorithm for 100 random samples on a 166-dimensional dataset created from the Corel image database (59,619 images). The histograms were extracted using the methodology and parameters described in [SC94]. 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.

In the experiments we set $m = 8$, and ordered the dimensions by decreasing value in $\mathbf{q}$. 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 pruned on the average after 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

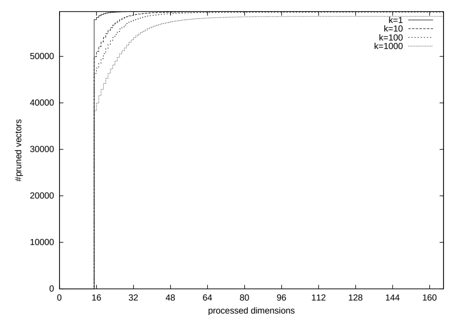


Figure 6: Effects of k in search $(m = 1)$

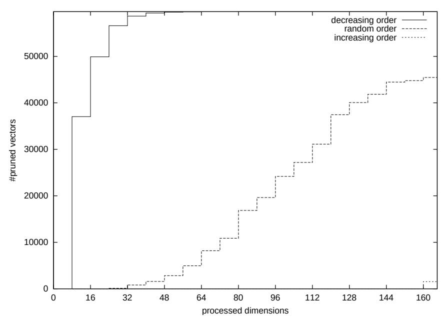


Figure 7: Effects of dimensional orderings

accessed at all. This shows another advantage of our approach; even if the worst memory settings apply, dimension-wise pruned search would do much better than sequential scan.

We applied search on the same dataset, using Euclidean distance as metric, and pruning criteria E_q and E_v. 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 3.7) 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. Figure 5 plots the pruning efficiency of the heuristics. 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. Although we performed the Euclidean 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. In the rest of the paper, we will not consider criterion E_q again.

6.2 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 our technique 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 one with large pruning efficiency. No images are pruned until the 15-th dimension, where $T(\mathbf{q}^-)$ becomes larger than 0.5.

The nature of skew in our 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 $\mathbf{q}$ and it is not static favors the application of branch-and-bound in comparison to sequential scan and other methods, because of its flexibility to consider the dimensions in any order without penalty in access cost.

6.3 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 dimensionality, although not dramatically. With the increase of dimensionality the search space becomes more inappropriate for nearest neighbor search: the distance ratio between the nearest and the furthest vector from a random point in space drops [BGRS99]. 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 have shown that the pruning efficiency of branch-and-bound does not degrade with dimensionality.

6.4 Search performance improvement

We evaluated the practical use of our approach, by measuring the response time of our implementation for 100 sample queries from the dataset with 166-dimensional HSV histograms, using both similarity metrics with the most efficient pruning criteria, i.e., H_q, H_h, and E_v. We also included the cost of 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.[7] Table 3 presents statistics for the response time of our implementation. The two versions of this method for histogram intersection and Euclidean distance are denoted as SS_H and SS_E, respectively. All reported times are in milliseconds.

Table 3: Pruned search vs. sequential scan.

method	min	max	average	median
H_q	95	464	162	131
H_h	135	597	225	179
SS_H	878	989	916	915
E_v	147	1044	418	326
SS_E	386	440	394	390

As expected, 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.

The E_v criterion is not that efficient in terms of CPU-cost, mainly due to the fact that it prunes less space on the average. Furthermore, the relatively complex bounds of $\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+)$ add some computational overhead. The median response of this method is faster than sequential scan. The differences in timings do not agree with the pruning effectiveness. We believe that these differences can be attributed to caching and branch-miss prediction effects, and are currently investigating the plausibility of this explanation. The large difference between SS_H and SS_E on all measurements hints into this direction.

Notice the relatively large difference between the median and average responses for each pruning criterion. This shows that fast responses are more frequent than significantly slower ones, such that the median reflects best the expected performance of the pruned search heuristics.

6.5 Effects of data skew

While branch-and-bound is very efficient for content-based image retrieval, a natural question arises: is it good for generic k-NN search, and if so, under which conditions? 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. Notice that in the color histogram datasets examined so far the values of a random vector follow a skewed Zipfian distribution. In such cases, an ordering that considers the most skewed dimensions in the query first, is expected to have nice pruning effect. 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 end up to be the worst overall solutions. We also expect our approach to do well when the data are clustered in the high-dimensional space, and k-NN search is meaningful [BGRS99].

[7]We also tried a more sophisticated approach, where the partial score of $\mathbf{v}$ was regularly compared to the top-k scores 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 much slower on the average (due to the overhead of comparisons and its incapability to consider first the most promising dimensions).

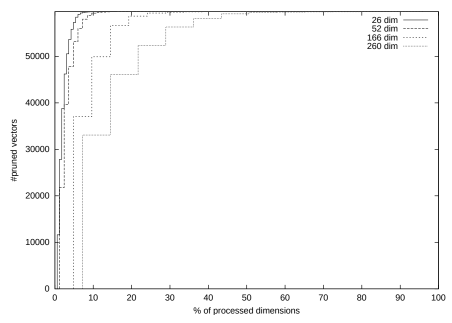


Figure 8: Impact of dimensionality on pruning efficiency

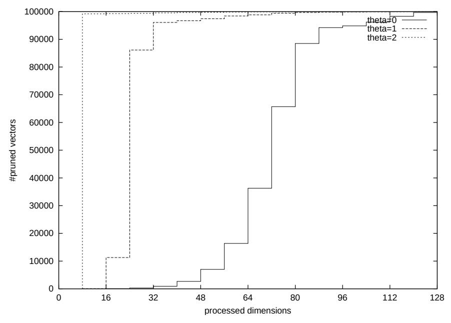


Figure 9: Effect of skew in the performance of E_v

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. 5% of the vectors take random values (noise), whereas 95% of them belong to some random cluster. The distance from each vector to the cluster where it belongs to is defined by a Gaussian distribution around the cluster's center. This dataset has the nice properties that make NN-search meaningful [BGRS99]; 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 9 shows the average pruning efficiency of E_v for various values of the skew parameter θ. As expected, our approach is not efficient when the centers of the clusters are uniform; data skew favors pruning. In real-life applications where datasets are skewed (like color histograms), we can expect good results from the proposed menthod. Also, as explained in the next section, weighted search puts implicit skew in data and increases the practical applicability of our approach.

7. Weighted Search

So far, we have assumed that all dimensions have equal importance in the query. 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 [ISF98].

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:*

$$\delta_w(\mathbf{v}, \mathbf{q}) = \sum_{i=1}^{N} w_i (v_i - q_i)^2 \tag{7.1}$$

Each dimension is assigned a weight w_i which reflects its importance in the query. If $\sum_{i=1}^{N} w_i = N$ then equation 2.5 defines the similarity of the two vectors. Figure 11 visualizes the special case where $m = N - 2$; basically, each dimension is stretched or shrinked with a different factor. In the presence

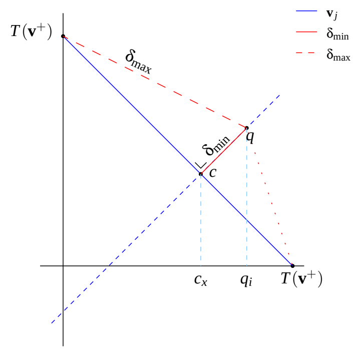


Figure 10: The lower- and upper bound on the Euclidean distance between $\mathbf{q}$ and any possible $\mathbf{v}$ visualized for $m = N - 2$ and $T(\mathbf{v}^+) \leq 1$.

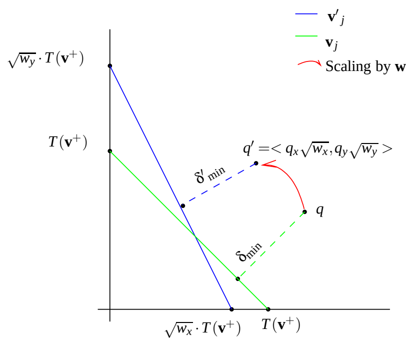


Figure 11: The lower- and upper bound on the Euclidean distance with weighting vector $\mathbf{w}$, visualized for $m = N - 2$.

of weights, the upper bound of $\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+)$ becomes:

$$\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+) \leq \sum_{i=m+1}^{l-1} w_i q_i^2 + w_l(T(\mathbf{v}^+) - f - q_l)^2 + \sum_{i=l+1}^{N} w_i(1 - q_i)^2 \tag{7.2}$$

Equation 7.2 assumes that the values of $\mathbf{q}^+$ are ordered such that $w_i q_i^2 \geq w_{i+1} q_{i+1}^2, \forall i > m$. Values f and l are defined as in Lemma 1. The lower bound is similarly defined by extending equation 3.9 as follows:

$$\mathcal{S}(\mathbf{v}^+, \mathbf{q}^+) \geq \frac{\prod_{i=m+1}^{N} w_i}{\sum_{j=m+1}^{N} \prod_{i=m+1, i \neq j}^{N} w_i} (T(\mathbf{v}^+) - T(\mathbf{q}^+))^2 \tag{7.3}$$

The proofs of equations 7.2 and 7.3 can be derived after squaring the maximum and minimum distance of the transformed vector $\mathbf{q}'^+, q_i'^+ = \sqrt{w_i} q_i^+, \forall m < i \leq N$ from the hyperplane defined by all possible distributions of $T(\mathbf{v}^+)$ to the dimensions after m.

The following experiments demonstrate the efficiency of our technique at weighted queries. A small proportion of high weights introduces skew in the transformed space, so we expect our approach to work well. We used the synthetic dataset from section 6.5 with θ=0, which yields the worst-case pruning of E_v (the cluster centers are uniformly distributed). Figure 12 shows the pruning efficiency for various values of the skew on weights. Observe that, in contrast to Figure 9, the skew on the weights has to be large for the pruning to be efficient. In practice, 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.

8. Related Work

A variety of techniques have been proposed previously to improve upon naive k-NN search. A rather ad-hoc solution precomputes for each image the k most similar ones [Fag98], and generates a *similarity network*. This method avoids the expensive run-time distance computations and ranking, and may be practical in some cases. Yet, updates cannot be done incrementally and the hardwired constants (like the number of neighbors per image and the similarity metric used) do not allow for queries with arbitrary k, nor weighted queries. Also, it is impossible to query for images that are not selected from the indexed collection.

Many researchers have tried to speed up the search process using a variety of indexing techniques. If the number of dimensions is low, a *spatial access method* (e.g., R-trees [Gut84], X-trees [BKK96]) is used to store the feature vectors and a k-nearest neighbor search algorithm (e.g., [RKV95]) facilitates efficient search. In practice, however, the number of dimensions in image databases is quite large, and scanning the complete database is faster than searching with a high-dimensional indexing method [WSB98, BGRS99].

The two-step query process applied in [FL95, KSF$^+$96, SK98] 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. The resulting (low-dimensional) vectors are organized in a spatial access method (SAM). A query is then processed in two steps. First, the SAM is used to locate the k-nearest neighbors in the low-dimensional space. The actual distances for them are computed, and the largest one is used for a range query on the SAM. For these results, the actual distances are also computed, and the k-nearest neighbors are returned.

A major disadvantage of this methodology is the problem of finding a *proper* mapping. The number of dimensions should be sufficiently small for efficient k-NN and range searching, and the distance in the low-dimensional space should lower-bound the distance in the high-dimensional space to guarantee correctness of the results. Moreover, the mapping should preserve enough information from the original space, to retrieve as few images as possible in the range query. 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) [WSB98] uses a smaller, approximate representation of the feature vectors (e.g., 8 bits instead of a double per dimension) 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 performed faster because it requires less bandwidth, and since the refinement step processes much less data, computing the top-k answers is cheaper than a full sequential scan on the original vectors. Recently, this approximation technique has also been applied to improve high-dimensional indexing trees, by compressing the leaf nodes (the IQ-tree [BBJKS00]) or storing a compressed representation of bounding boxes of child nodes in the inner nodes (the A-tree [SYUK00]). But, as mentioned in [BBJKS00], the performance of searching tree structures reduces to that of VA–File for high dimensionality.

Table 4: Experiments with approximations of the original feature vectors.

method	min	max	average	median
filter step H_q	39	278	79	64
filter step SS_H	341	394	369	370
refinement step	4	24	10	10

We expect a similar improvement using this approximation technique in combination with branch-and-bound. Figure 13 plots the obtained pruning of H_q with and without an 8-bit approximation per dimension. On the average, pruning finishes with 1000 candidates after processing 72 dimensions. Pruning on the VA–File follows a similar trend as using the original vectors. Table 4 shows timings of the experiment of Section 6.4, compared to the results of sequentially scanning the equivalent VA–File. These results show clearly 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.

9. Conclusions

To the best of our knowledge, we are the first to experiment with k-NN search using vertical fragmentation of feature vectors. A common practice would be to store 166-dimensional image histograms as ADTs; conventional 'wisdom' votes against dividing a collection of histograms over 166 binary tables. But, as we demonstrate in this paper, using the ADT approach may not *always* be the most efficient solution. Our approach allows us to apply database optimization techniques, which would not be possible otherwise: we process the most promising dimensions first, and choose between a bitmap and

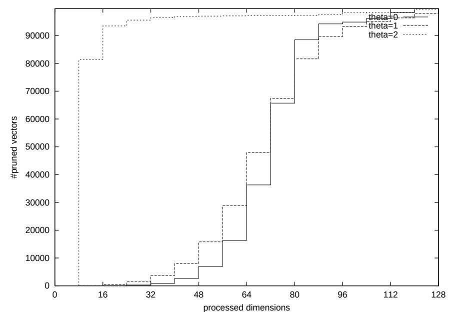


Figure 12: Effects of skew on the weights

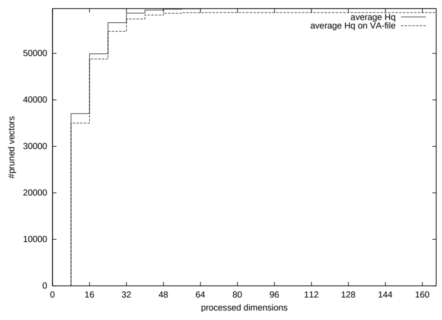


Figure 13: Pruning effects with approximated feature vectors.

lists of identifiers depending on the selectivity of the pruning process. Characterizing our branch-and-bound algorithm in conventional database terms, we adapted the well-known 'push-select-down' heuristic to a not-so-standard fashion that applies to image retrieval, and possibly high-dimensional search in other domains as well.

We demonstrated how to take advantage of vertical fragmentation for k-NN search using two popular similarity metrics in a high-dimensional space. We show how even more advantage is gained if some dimensions are more important than others, a very realistic scenario in interactive search applications. Also, our approach is simple and introduces negligible storage overhead. We showed experimentally that the efficiency gained with branch-and-bound combines trivially with the common approach to compress feature vectors.

Although we have applied our algorithm to image retrieval only, the method is generic and we expect performance gains in other domains as well, e.g., data mining and clustering. In our future work, we plan to investigate a parallel version of the algorithm, as well as optimization strategies for multiple users searching the same database simultaneously. Another promising direction is the combination of various search strategies. For example, we could combine search criteria on text and images in the same web page, or, search only the photographs of some particular publishing agency from a large photo archive.

10. Acknowledgements

Thanks to Jan-Mark Geusebroek, University of Amsterdam, for inspiring this research. Thanks to our colleagues Menzo Windhouwer and Peter Bosch for their help with the datasets used in this paper.

References

[ACDM00] M. Annamalai, R. Chopra, S. DeFazio, and S. Mavris, *Indexing Images in Oracle8i*, Proc. of ACM SIGMOD Int'l Conference, 2000.

[BGRS99] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, *When Is "Nearest Neighbor" Meaningful?* Proc. of Int'l Conference on Database Theory (ICDT), 1999.

[BK99] P. A. Boncz and M. L. Kersten, *MIL Primitives for Querying a Fragmented World*, VLDB Journal 8(2): 101–119, 1999.

[BKK96] S. Berchtold, D. A. Keim and H.-P. Kriegel, *The X-tree : An Index Structure for High-Dimensional Data*, Proc. of VLDB Conference, 1996.

[CK85] G.P. Copeland and S.N. Koshafian, *A Decomposition Storage Model*, Proc. of ACM SIGMOD Int'l Conference, 1985.

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

[Fag98] R. Fagin, *Fuzzy Queries in Multimedia Database Systems*, Proc. of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 1998.

[FL95] C. Faloutsos and K.-I. Lin, *FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets*, Proc. of ACM SIGMOD Int'l Conference, 1995.

[GBK00] U. Güntzer, W.-T. Balke, and W. Kiessling, *Optimizing Multi-Feature Queries for Image Databases*, Proc. of VLDB Conference, 2000.

[Gut84] A. Gutmann, *R-Trees: A Dynamic Index Structure for Spatial Searching*, Proc. of ACM SIGMOD Int'l Conference, 1984.

[ISF98] I. Ishikawa, R. Subramanya, and C. Faloutsos, *MindReader: Querying databases through multiple examples*, Proc. of VLDB Conference, 1998.

[KSF$^+$96] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas, *Fast Nearest Neighbor Search in Medical Image Databases*, Proc. of VLDB Conference, 1996.

[ORC$^+$97] M. Ortega, Y. Rui, K. Chakrabarti, S. Mehrotra, and T. S. Huang, *Supporting Similarity Queries in MARS*, Proc. of ACM Multimedia Conference, 1997.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent, *Nearest Neighbor Queries*, Proc. of ACM SIGMOD Int'l Conference, 1995.

[SB91] M. Swain and D. Ballard, *Color Indexing*, International Journal of Computer Vision, 7(1):11–32, 1991.

[SC94] J.R. Smith and S.-F. Chang, *Tools and Techniques for Color Image Retrieval*, In IS & T/SPIE Proceedings, volume 2670, 1994. Storage & Retrieval for Image and Video Databases IV.

[SC95] J.R. Smith and S.-F. Chang, *Automated Image Retrieval Using Color and Texture*, Columbia University CTR, TechREPORT, 414-95-20, New York, 1995.

[SK98] T. Seidl and H.-P. Kriegel, *Optimal Multi-Step k-Nearest Neighbor Search.* Proc. of ACM SIGMOD Int'l Conference, 1998.

[WSB98] R. Weber, H.-J. Schek, and S. Blott, *A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces.* Proc. of VLDB Conference, 1998.

[BBJKS00] S. Berchtold, Ch. Böhm, H.V. Jagadish, H.-P. Kriegel, and J. Sander, *Independent Quantization: An Index Compression Technique for High-Dimensional Spaces.* Proc. of International Conference on Data Engineering (ICDE), 2000.

[SYUK00] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima, *The A-tree: an index structure for high-dimensional spaces using relative approximation.* Proc. of VLDB Conference, 2000.