# VA-Files vs. R*-Trees in Distance Join Queries*

Antonio Corral[1], Alejandro D'Ermiliis[1], Yannis Manolopoulos[2], and Michael Vassilakopoulos[3]

[1] Department of Languages and Computing, University of Almeria, 04120 Almeria, Spain
{acorral, sandro}@ual.es
[2] Department of Informatics, Aristotle University, GR-54124 Thessaloniki, Greece
manolopo@csd.auth.gr
[3] Department of Informatics, Technological Educational Institute of Thessaloniki, P.O. BOX 141, GR-57400, Thessaloniki, Greece
vasilako@it.teithe.gr

**Abstract.** In modern database applications the similarity of complex objects is examined by performing distance-based queries (e.g. nearest neighbour search) on data of high dimensionality. Most multidimensional indexing methods have failed to efficiently support these queries in arbitrary high-dimensional datasets (due to the dimensionality curse). Similarity join queries and *K* closest pairs queries are the most representative distance join queries, where two high-dimensional datasets are combined. These queries are very expensive in terms of response time and I/O activity in case of high-dimensional spaces. On the other hand, the filtering-based approach, as applied by the VA-file, has turned out to be a very promising alternative for nearest neighbour search. In general, the filtering-based approach represents vectors as compact approximations, whereas by first scanning these approximations, only a small fraction of the real vectors is visited. Here, we elaborate on VA-files and develop VA-file based algorithms for answering similarity join and *K* closest pairs queries on high-dimensional data. Also, performance-wise we compare the use of VA-files and R*-trees (a structure that has been proven to be of robust nature) for answering these queries. The results of the comparison do not lead to a clear winner.

## 1 Introduction

Large sets of complex objects are used in modern applications (e.g. multimedia databases [11], medical images databases [15], etc.). To examine the similarity of these objects, high-dimensional feature vectors (i.e. points in the high-dimensional spaces) are extracted from them and organized in multidimensional indexes. Then, distance-based queries (e.g. nearest neighbour, similarity join, *K* closest pairs, etc.) are applied on the high-dimensional points. The most representative high-dimensional *distance join queries* (DJQ), where two datasets are involved, are the *similarity join*

* Supported by the ARCHIMEDES project 2.2.14, «Management of Moving Objects and the WWW», of the Technological Educational Institute of Thessaloniki (EPEAEK II), co-funded by the Greek Ministry of Education and Religious Affairs and the European Union, INDALOG TIC2002-03968 project «A Database Language Based on Functional Logic Programming» of the Spanish Ministry of Science and Technology under FEDER funds, and the framework of the Greek-Serbian bilateral protocol.

J. Eder et al. (Eds.): ADBIS 2005, LNCS 3631, pp. 153 – 166, 2005.

query (SJ) and the *K closest pairs query* (*K*-CPQ). The SJ query discovers all pairs of points from two different point datasets, where the distance does not exceed a distance threshold $\delta$. The *K*-CPQ discovers $K>0$ distinct pairs of points formed from two different point datasets that have the *K* smallest distances between them. The former does not take into account the cardinality and order of the final result (but only the user-defined distance threshold $\delta$), whereas the latter does not consider any distance bound (but only the user-defined final result cardinality *K*). Note that these queries have been successfully applied in data mining algorithms (e.g. clustering algorithms based on similarity join [3] and closest pairs [16]).

Here, we focus on performing DJQ using a *filtering-based approach* that has proven to outperform a sequential scan for high dimensionalities, when a tree index fails to process a *K* nearest neighbour query (*K*-NNQ) efficiently (dimensionality curse). The VA-file (vector-approximation file) is the most representative access method of this category [20]. Instead of partitioning, the VA-file constructs the *index file* by compressing each feature vector. With respect to query processing, the compact vector approximations are sequentially scanned and filtered in the first stage so that a small fraction of them remains to be visited in the second stage. The improvement for *K*-NNQ arises due to the reduced I/O accesses (as the index file size is small) and due to the smaller response time (because of the fewer distance computations).

The main goal of this paper is to develop VA-file based algorithms for DJQ involving two sets of high-dimensional data. More specifically, we develop algorithms for SJs and *K*-CPQs in high-dimensional spaces, where both point datasets are indexed by VA-files. To achieve this goal, we propose new bounds on the distance between pairs of points and new pruning conditions. Moreover, we present experimental results comparing the performance of these algorithms with analogous algorithms that make use of R*-trees [1], in terms of the I/O activity and the response time. Based on these results, we draw conclusions about the behaviour of the algorithms that use VA-files for DJQ in high-dimensional spaces.

The paper is organized as follows. In Section 2, we review the related literature and motivate the research reported here. In Section 3, a brief description of the VA-file structure, definitions of the most representative DJQ, approximation-based distance functions and pruning conditions are presented. In Section 4, algorithms based on distance bounds and pruning conditions over VA-files for *K*-CPQ and SJ are examined. In Section 5, a comparative performance study of these algorithms is reported. Finally, in Section 6, conclusions on the contribution of this paper and future work are summarized.

## 2 Related Work and Motivation

Numerous algorithms have been proposed for satisfying DJQ in high-dimensional environments. For similarity joins on high-dimensional point datasets, the most representative papers are [18, 14, 10, 4]. In [18] an index structure ($\varepsilon$-kdB tree) and an algorithm for similarity self-join on high-dimensional points was presented. The basic idea is to partition the dataset perpendicularly to a selected dimension into stripes of the width $\varepsilon$ to restrict the join algorithm to pairs of subsequent stripes. In [14] the

problem of computing high-dimensional similarity joins between two high-dimensional point datasets, where neither input is indexed (Multidimensional Spatial Join, MSJ), was investigated. The basic idea of this access method is to partition the dataset into level-files, each of which contains the points of a level in the order of their Hilbert values. In [10] a new algorithm (Generic External Space Sweep, GESS), which introduces a rate of data replication to reduce the number of distance computations as an enhancement of MSJ, was proposed. In [4], a complex and interesting index architecture (Multipage Index, MuX) and join algorithm (MuX-join), which allows a separate optimization CPU time and I/O time, were presented. On the other hand, the *K*-CPQ has not been studied in-depth for high-dimensionality data. In [8], DFS-based approximate algorithms for the *K*-CPQ using R-trees [13] have been proposed (in order to get suboptimal results in reasonable time). One of the main objectives of this work was to examine the influence of the approximate parameters on the trade-off between accuracy and efficiency of such algorithms.

Many approaches have been proposed to overcome the *curse of dimensionality* in the context of *K*-NNQ. They are usually classified into five major categories: (1) tree index structures by partitioning the data space or data-partitioning; (2) space-filling curves, (3) dimensionality reduction approaches; (4) approximate algorithms and (5) filtering-based (i.e. approximation) approaches. In this paper, we are going to focus on the last category. The filtering-based approach overcomes the dimensionality curse by filtering the points so that only a small fraction of them must be visited during a search. In this respect, the most representative access method is the VA-file [20], which divides the data space into $2^b$ rectangular cells, where $b$ denotes a user-specified number of bits. The VA-file allocates a unique bit-string of length $b$ to each cell and approximates data points that fall into a cell by that bit-string. In general, the VA-file itself is simply an array on disk of these compact approximations of points.

Following the ideas of the VA-file, many variants have proposed to improve the performance of *K*-NNQ. The VA$^+$-file [12] combines a linear decorrelation using KLT (Karhunen-Loève Transformation) along with a variance specific quantization scheme using the VA-file principles. The LPC-file [6] enhances the VA-file by adding polar coordinate information of the point (vector) to approximation, increasing the discriminatory power. The GC-tree [5] pursues a hybrid strategy which incorporates a quad-tree-like hierarchical space partitioning with bit-encoded clusters and a point approximation based on local polar coordinates on the leaf nodes. In the IQ-tree [2], all points are globally approximated according to one fixed grid (like the VA-file) and it also maintains a flat directory containing the minimum bounding rectangles (MBRs) of the approximate data representations. The A-tree [17] combines hierarchical indexing and local approximation by quantization. The MBRs of point clusters are approximated by quantization in so-called virtual bounding rectangles (VBRs). And recently, the SA-tree [9] was proposed, which combines data clustering and compression (i.e. it employs the characteristics of each cluster to adaptively compress points to bit-string) to speed up processing of high-dimensional *K*-NNQ.

All the previous efforts have been mainly focused on enhancing the VA-file to improve the performance during the *K*-NNQ (a query applied on a single set of high-dimensional data). The main objective of this paper is to investigate the behaviour of VA-files on DJQ involving pairs of high-dimensional data sets (SJs and *K*-CPQs). For

this reason, we propose new bounds of the distance between pairs of points, new pruning conditions and lead to algorithms for these DJQ using VA-files.

## 3 Distance Join Queries for VA-Files

### 3.1 Distance Join Queries

Let us consider points in the *dim*-dimensional data space ($D^{dim} = \Re^{\dim}$) and a distance function for a pair of these points. A general distance function is the $L_t$-distance *($d_t$)* or Minkowski distance between two points $p_i$ and $q_j$ from two different datasets ($P = \{p_i: 0 \le i \le |P|-1\}$ and $Q = \{q_j: 0 \le j \le |Q|-1\}$, respectively) in $D^{dim}$ ($p_i = (p_i[0], p_i[1],\ldots, p_i[\text{dim-1}])$ and $q_j = (q_j[0], q_j[1],\ldots, q_j[\text{dim-1}])$), where $p_i[d]$ ($q_j[d]$) is the coordinate value of $p_i$ ($q_j$) in dimension $d$, that is defined by:

$$d_t(p_i,q_j)=\left(\sum_{d=0}^{\dim-1}\left|p_i[d]-q_j[d]\right|\right)^{\frac{1}{t}},\ if\quad 1\le t<\infty\text{, and } d_\infty(p_i,q_j)=\max_{0\le d\le \dim-1}\left|p_i[d]-q_j[d]\right|$$

For $t = 2$ and $t = 1$ we have the Euclidean and the Manhattan distances. They are the most known $L_t$-distances. Often, the Euclidean distance is used as a distance function, but, depending on the application, other distance functions may be more appropriate. The *dim*-dimensional Euclidean space (metric space), $E^{dim}$, is the pair ($D^{dim}$, $d_2$). In the following, we will use *dist* instead of $d_2$. The most representative DJQ in $E^{dim}$ are the following:

*Definition. Similarity Join (SJ).* Let P and Q be two point datasets ($P \neq \emptyset$ and $Q \neq \emptyset$) in $E^{\dim}$ and $\delta$ a real number $\delta \ge 0$. Then, the result of the Similarity Join is the set $SJ(P,Q,\delta) \subseteq P \times Q$ containing all possible pairs of points of $P \times Q$ that can be formed by choosing one point of P and one point of Q, having a distance smaller than or equal to $\delta$: $\mathbf{SJ(P, Q, \delta)} = \{(p_i,q_j) \in P \times Q: \text{dist}(p_i,q_j) \le \delta\}$.

*Definition. K closest pairs query (K-CPQ).* Let P and Q be two point datasets ($P \neq \emptyset$ and $Q \neq \emptyset$) in $E^{\dim}$ and K an integer number in the range $1 \le K \le |P| \cdot |Q|$. Then, the result of the K closest pairs query is the set $K\text{-CPQ}(P,Q,K) \subseteq P \times Q$ containing all the ordered sequences of K different pairs of points of $P \times Q$ with the K smallest distances between all possible pairs of points that can be formed by choosing one point of P and one point of Q: $\mathbf{K\text{-}CPQ(P,Q,K)} = \{((p_1,q_1), (p_2,q_2), ..., (p_K,q_K)) \in (P \times Q)^K: p_1, p_2, ..., p_K \in P, q_1, q_2, ..., q_K \in Q, (p_i,q_i) \neq (p_j,q_j)\ i \neq j\ 1 \le i,j \le K, \forall (p_i,q_j) \in P \times Q - \{(p_1,q_1), (p_2,q_2), ..., (p_K,q_K)\}$ and $\text{dist}(p_1,q_1) \le \text{dist}(p_2,q_2) \le \ldots \le \text{dist}(p_K,q_K) \le \text{dist}(p_i,q_j)\}$.

For SJ, if the sets $P$ and $Q$ coincide, then the DJQ is called *similarity self-join* (widely studied in [18, 14, 10, 4]). Fig. 1 illustrates these DJQs, where the points of $P$ and $Q$ are represented by starts (*) and crosses (+), respectively. In the left part of Fig. 1, we can observe that $SJ(P,Q,\delta) = \{(p_3,q_1), (p_4,q_6), (p_6,q_6), (p_8,q_8), (p_8,q_9), (p_8,q_{10}), (p_{11},q_9), (p_{11},q_{10})\}$ where $\delta = 0.8$. If we want to obtain the four closest pairs ($K = 4$) of the two data-sets depicted in the right part of Fig. 1, the result is $K\text{-CPQ}(P,Q,K) = \{(p_8,q_8), (p_{11},q_{10}), (p_4,q_6), (p_8,q_9)\}$.

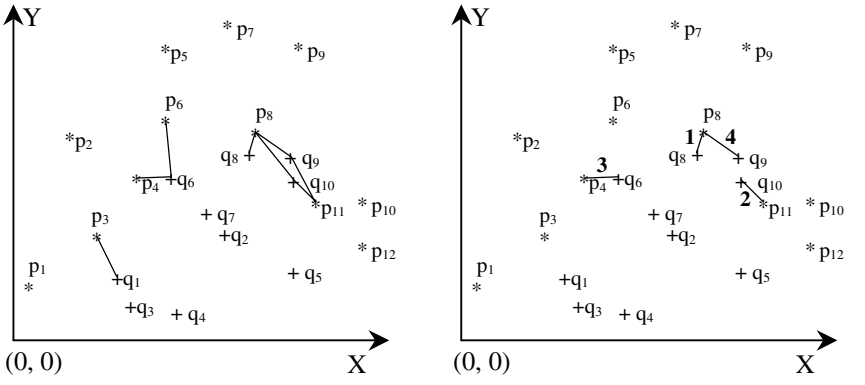


**Fig. 1.** Examples of SJ and *K*-CPQ using 2-dimensional points

### 3.2 The VA-File (Vector-Approximation File)

The VA-file [20] does not partition the data, but the data space is partitioned into rectangular cells which are used to generate bit-encoded approximations of the points. Therefore, the VA-file consists of two files: one contains an approximation of the feature representation of each point *(approximation file)*, whereas the other one the exact representation of each point *(vector file)*. They are connected by indexes, since they are simple arrays on disk. The quantization is obtained by laying a grid over the data space and approximating the points by their surrounding cells (see left part of Fig. 2). The grid has $2^{b_d}$ intervals along dimension $d$ ($0 \le d \le dim-1$), where $b = \sum_d b_d$ is the number of bits per approximation, $b_d$ is the number of bits for dimension $d$ and $dim$ the dimensionality of the data space. In Fig. 2, $b_d = 2$ and $dim = 2$ (a realistic $b_d$ value for nearest neighbour search would be between 6 and 8 according to [20]). The *intervals* of this grid are numbered from 0 to $2^{b_d} - 1$ (see left part of Fig. 2), and the *partition points* $m[d, 0]$, $m[d, 1]$, …, $m[d, 2^{b_d}]$ bound them. That is, $m[d, k]$ represents the $k$-th partition point in dimension $d$; and in total, there are $2^{b_d} + 1$ partition points

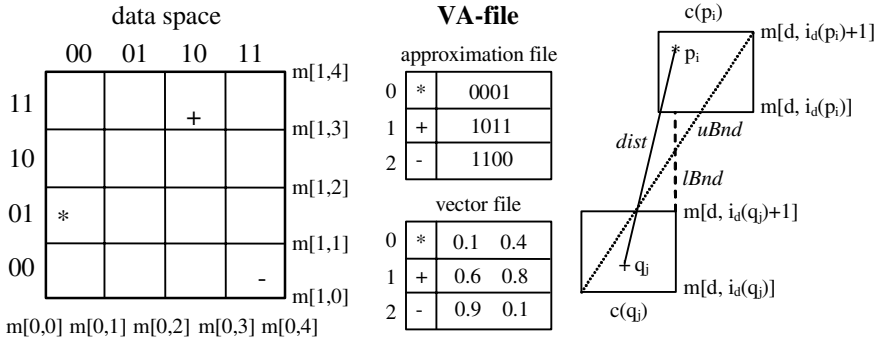


**Fig. 2.** Structure of the VA-file and, distances between points and cells

and $2^{b_d}$ intervals. These partition points are determined so that each interval contains the same number of vectors. Given a point $p_i$, $i_d(p_i)$ denotes the *interval* in dimension $d$ that $p_i$ falls into, i.e. it is the approximation of a point $p_i$ ($P = \{p_i: 0 \le i \le |P|-1\}$) in dimension $d$ and $i_d(p_i) \in \{0,1,\ldots, 2^{b_d}-1\}$. Thus, the following expression holds ($p_i$[d] is the value of $p_i$ in dimension $d$): $m[d, i_d(p_i)] \le p_i[d] < m[d, i_d(p_i)+1]$, $\forall d$: $0 \le d \le dim-1$.

A *bit-string* of length $b = \sum_d b_d$ ($0 \le d \le dim-1$) represents each *cell*. Such a bit-string is the concatenation of the bit-strings of the interval numbers of the cell (for example, the point (+) falls into the cell with the bit-string 1011). Thus, the approximation of $p_i$ is the bit-string of the cell (represented by $c(p_i)$) that contains $p_i$ and it is denoted by $a(p_i)$ (i.e. elements of approximation file). Thus, the approximation file is simply an array of these approximations. Intuitively, $a(p_i)$ contains sufficient information to determine the cell $c(p_i)$ in which $p_i$ lies. Notice that for large *dim* values, the volume of a cell is so small that it is highly unlikely the two points lie in the same cell.

### 3.3 Distance Bounds Between Cells and Pruning Conditions

Next, we are going to show how pairs of cells can be used to derive (lower and upper) bounds between pairs of points. Given two points from two different points datasets $p_i \in P$ and $q_j \in Q$, the minimum (maximum) distance between their cells ($c(p_i)$ and $c(q_j)$, respectively) is a lower (upper) bound of its distance. Thus, given the cells of two points from two different datasets, we can bound from below and above their distance ($dist(p_i,q_j)$) as follows (according to the terminology of [20]): $lBnd(c(p_i), c(q_j)) \le dist(p_i, q_j) \le uBnd(c(p_i), c(q_j))$.

The lower bound, $lBnd(c(p_i), c(q_j))$, is the smallest distance between the cells of $p_i$ and $q_j$. Obviously, $lBnd(c(p_i),c(q_j),d) \le lBnd(c(p_i),c(q_j))$, $\forall d$: $0 \le d \le dim-1$ [7]. Analogously, we can obtain the upper bound, $uBnd(c(p_i), c(q_j))$. The right part of the Fig. 2 shows these distance bounds and its relation with $dist(p_i, q_j)$.

$$lBnd\left(c(p_i),c(q_j)\right) = \sqrt{\sum_{d=0}^{\dim-1} \begin{cases} \left(m[d,i_d(p_i)] - m[d,i_d(q_j)+1]\right)^2, & m[d,i_d(p_i)] > m[d,i_d(q_j)+1] \\ \left(m[d,i_d(q_j)] - m[d,i_d(p_i)+1]\right)^2, & m[d,i_d(q_j)] > m[d,i_d(p_i)+1] \\ 0, & otherwise \end{cases}}$$

$$uBnd\left(c(p_i),c(q_j)\right) = \sqrt{\sum_{d=0}^{\dim-1} \begin{cases} \left(m[d,i_d(p_i)+1] - m[d,i_d(q_j)]\right)^2, & m[d,i_d(p_i)] > m[d,i_d(q_j)+1] \\ \left(m[d,i_d(q_j)+1] - m[d,i_d(p_i)]\right)^2, & m[d,i_d(q_j)] > m[d,i_d(p_i)+1] \\ \max\left\{\begin{matrix}\left(m[d,i_d(p_i)+1] - m[d,i_d(q_j)]\right)^2, \\ \left(m[d,i_d(q_j)+1] - m[d,i_d(p_i)]\right)^2\end{matrix}\right\}, & otherwise \end{cases}}$$

In order to design efficient algorithms for DJQ using the VA-file structure, pruning conditions need to be defined.

*Pruning Condition 1.* If $lBnd(c(p_i),c(q_j)) > z$, then the pair of points $(p_i,q_j)$ will be discarded from the final result, where z is the $\delta$ distance threshold for SJ, or the

distance value of the K-th closest pair that has been found so far (K-$cp^{dist}$(p,q)) for K-CPQ. lBnd(c($p_i$), c($q_j$)) ≤ δ ⇒ ($p_i$, $q_j$) ∈ SJ(P, Q, δ) and lBnd(c($p_i$), c($q_j$)) ≤ K-$cp^{dist}$(p, q) ⇒ ($p_i$, $q_j$) ∈ KCPQ(P, Q, K)

*Pruning Condition 2*. If lBnd(c($p_i$),c($q_j$)) > y, then the pair of points ($p_i$, $q_j$) will be discarded from the final result, where y is the δ distance threshold for SJ, or the distance value of the K-th largest upper bound encountered so far (K-$cp^{uBnd}$(c(p), c(q))) for K-CPQ. lBnd(c($p_i$), c($q_j$)) > δ ⇒ ($p_i$, $q_j$) ∉ SJ(P, Q, δ) and lBnd(c($p_i$), c($q_j$)) > K-$cp^{uBnd}$(c(p), c(q)) ⇒ ($p_i$, $q_j$) ∉ KCPQ(P, Q, K). Note that in the case of SJ the two pruning conditions are the same.

## 4 Algorithms for Distance Join Queries Using VA-Files

The previous distance bounds between cells and pruning conditions can be embedded into search algorithms for VA-files and obtain the result of DJQ. In this section we describe additional data structures needed for DJQ, a distance-based sweeping technique for fast pruning, and two search algorithms using VA-files as in [20].

### 4.1 Data Structures for the Result and Distance-Based Sweep Technique

In order to design algorithms for processing *K*-CPQ in a non-incremental way (*K* must be fixed in advance) [7], an extra data structure that holds the *K* closest pairs (result of *K*-CPQ) is needed. This data structure is organized as a maximum binary heap, called *Kheap* [8]. The closest pair with the largest distance ($K\text{-}cp^{dist}(p,q)$) resides on top of the Kheap (the root), and it will be used in *pruning condition 1*. Notice that this data structure will also be used to calculate $K\text{-}cp^{uBnd}(c(p),c(q))$, used in *pruning condition 2*. On the other hand, the result of the SJ must not be ordered, and the Kheap is not needed. Therefore, the data structure that holds the result set is (instead of Kheap) a file of records (*resultFile*) of three fields, where the first field will be the distance, whereas the second and the third ones will be the pair of points ($p_i$,$q_j$). To accelerate the performance of SJs, a page buffer is used in main memory to hold the records as they are computed and as soon as it gets full, we add a new page to the result file.

Since the approximation file itself is simply a *flat array on disk* of all the approximations of points (approximation file), we can adapt the *distance-based plane-sweep technique* [7] for the high-dimensional space to avoid processing all possible combinations of pairs from two approximations files. In general, this technique consists of choosing a sweeping dimension and sorting the approximations on this dimension in increasing order (if both files are sorted already on a common dimension, no sorting is necessary). First, the sweeping dimension ($0 \le sd \le dim-1$) is established (e.g. $sd = 0$ or X-axis). After that, two pointers are maintained initially pointing to the first entry of each sorted approximation file. Let *pivot* be the entry of the smallest value of the approximation over the sweeping dimension pointed by one of these two pointers, e.g. $pivot = a(p_0)$ $\{a(p_i): 0 \le i \le |P|-1\}$. The cell of the pivot must be paired up with the cells determined by the approximations stored in the other approximation file $\{a(q_j): 0 \le j \le |Q|-1\}$ from left to right that satisfy *lBnd*(*c*(*pivot*),

$c(q_j),sd) \leq z$ (where $z$ is a pruning distance, e.g. $z = \delta$ for SJs), obtaining a set of candidate pairs of approximations where the element *pivot* is fixed. After all possible pairs of approximations that contain *pivot* have been found, the pointer of the pivot is increased to the next entry, *pivot* is updated with the approximation of the next smallest value of the approximation over the sweeping dimension pointed by one of the two pointers and the process is repeated until one of the approximation file is completely scanned.

Notice that we apply $lBnd(c(p_i),c(q_j),sd)$ because in this technique, the sweeping takes place only over one dimension. Moreover, the search is only restricted to the closest cells (obtained from approximations of points) with respect to the cell of the *pivot* entry according to the current $z$ value. No duplicated pairs are obtained, since the cells are always scanned over sorted approximation files.

### 4.2 Distance-Based Sweep Algorithm (VA-DBSA)

The general schema for search algorithms using the VA-file structure has two phases. In the first phase *(filtering step)*, the approximations of points (approximation file) are scanned to determine lower bounds on the distance of cells pairs, and pairs of points are pruned according to the distance-based sweep technique and the pruning conditions. In the second phase *(refinement step)*, the filtered points (vector file) are visited and the pairs of points that satisfy the distance condition (SJ or $K$-CPQ) are chosen for the final result. Notice that the performance of this algorithm depends upon the ordering of the approximations and points. The algorithm for processing the $K$-CPQ is described by the following steps ($z = K\text{-}cp^{dist}(p,q)$; at the beginning $z = \infty$):

- *Filtering step*: Apply the *distance-based sweep technique* over the two approximation files, according to $lBnd(c(p_i),c(q_j),sd)$. Then, from these filtered pairs of approximations $(a(p_i),a(q_j))$ select only those that satisfy the *pruning condition 1*, i.e. $lBnd(c(p_i),c(q_j)) \leq z$.
- *Refinement step*: From the final candidates of the filtering step, select only those pairs of points from vector files having $dist(p_i,q_j) \leq z$. Insert all of them into Kheap until it gets full. Then remove the root of the Kheap and insert the new pair of points $(p_i,q_j)$, updating this data structure and $z = K\text{-}cp^{dist}(p,q)$.

The adaptation of this algorithm (VA-DBSA) from $K$-CPQ to the SJ is very simple. In the filtering and refinement steps, replace $z$ with $\delta$. Notice that Kheap is now unnecessary and the final result is stored in *resultFile*.

### 4.3 Near Optimal Distance-Based Sweep Algorithm (VA-NODBSA)

In [20] a near optimal algorithm for $K$-NNQ which minimizes the number of vectors visited was proposed. Here, we present a version of near optimal algorithm for DJQ, although it is *more complex*, *time-consuming* and has *memory-overhead*. It has also two phases. (1) During the *filtering step* the approximations are scanned, the distance-based sweep technique is applied and, the *lBnd* and *uBnd* are computed for each pair of approximations. Assuming that $K\text{-}cp^{uBnd}(c(p),c(q))$ is also calculated using a Kheap, if a pair of approximations is encountered such that $lBnd(c(p_i),c(q_j)) > K\text{-}cp^{uBnd}(c(p),c(q))$, then the pair of points $(p_i,q_j)$ can be discarded. The selected pairs of

approximations and their *lBnd* are organized as a minimum binary heap, called *Nheap* [7]. The size of Nheap could be very large with the increase of *dim* and the cardinality of the datasets, and a hybrid memory/disk scheme and techniques based on range partitioning could be needed [8]. (2) During the *refinement step* the pairs stored in Nheap are visited in increasing order of *lBnd* to determine the final answer set. Not all these candidate pairs of points are visited, but this phase ends when $lBnd(c(p_i),c(q_j)) > K\text{-}cp^{dist}(p,q)$, (recall that $K\text{-}cp^{dist}(p,q)$ is also calculated using a *Kheap*). The algorithm for *K*-CPQ is described by the following steps ($z = K\text{-}cp^{dist}(p,q)$ and $y = K\text{-}cp^{uBnd}(c(p),c(q))$, at the beginning $z = \infty$ and $y = -\infty$):

- *Filtering step*: Create Nheap, and a Kheap structure based on *uBnd*, called *KheapU*. Apply the *distance-based sweep technique* over the two approximation files, according to $lBnd(c(p_i),c(q_j),sd)$. Then, from these pairs of approximations $(a(p_i), a(q_j))$ select only those that satisfy the *pruning condition 2*, i.e. $lBnd(c(p_i),c(q_j)) \leq y$, and store them in Nheap. $y = K\text{-}cp^{uBnd}(c(p)$, c(q)) is computed using KheapU.
- *Refinement step*: Process Nheap from these pairs of approximations $(a(p_i), a(q_j))$ while $lBnd(c(p_i),c(q_j)) \leq z$, i.e. using the *pruning condition 1*. $z = K\text{-}cp^{dist}(p,q)$ is computed using a Kheap structure based on *dist*, called *KheapD*. Moreover, select only those pairs of points from vector files having $dist(p_i, q_j) \leq z$, and insert all of them into KheapD until it gets full. Then remove the root of the KheapD and insert the new pair of points $(p_i, q_j)$, updating this data structure and $z = K\text{-}cp^{dist}(p,q)$.

The adaptation of this algorithm (VA-NODBSA) from *K*-CPQ to the SJ is analogous to the adaptation of VA-DBSA for both phases (filtering and refinement).

## 5 Experimental Results

In this section, we have evaluated the performance of our algorithms over real high-dimensional datasets of image features (unlike [20] where uniform data have been used) extracted from a Corel image collection (http://corel.digitalriver.com/), available from [21]. We have chosen two datasets of features based on the colour histogram (CH) and colour histogram layout (HL). Each real dataset contains 68,040 feature vectors of *dim* = 32. From each 32-dimensional vector, we have chosen the first 4, 8, 12, 16 and 32 dimensions, giving rise to pairs of points datasets with different dimensionalities and the same cardinality (68,040). These pairs of datasets are used in *K*-CPQ and SJ.

All experiments were performed on an Intel/Linux workstation with a Pentium IV 2.5 GHz processor, 1 GByte of main memory, and several GBytes of secondary storage, using the gcc compiler. The index page size was 8 Kb, and the number of items sharing the same disk page decreased as the dimensionality increased. All the elements were fetched directly from the disk without caching. The performance measurements are mainly: (a) the elapsed time (wall-clock time) reported in seconds and (b) the number of page accesses. For comparison purposes, we have also implemented distance join algorithms using nested loops over the vector files and R-tree-based distance join algorithms [8], applying in both cases the *distance-based*

*sweep technique* described previously. Besides, the index construction was not taken into account for the total elapsed time.

Our first experiment seeks the most appropriate number of bits per dimension *($b_d$)* for VA-files that will be used in the next experiments. The suggested value in [20] for $b_d$ was 8, although here we have obtained (after many experiments) that $b_d$ = 10 reports better results for DJQ. For higher values of $b_d$ the size of the approximation file can be larger than the size of the vector file, and the filtering power is seriously affected, since the vectors themselves are used without being approximated. We have also observed that VA-NODBSA minimizes the number of vectors visited, although it is time-consuming (slower than VA-DBSA), because in the filtering step it is necessary to maintain two auxiliary structures Nheap and KheapU (variable sizes).

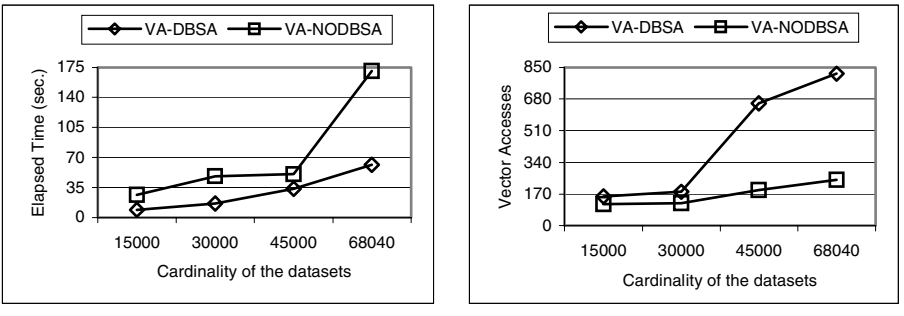


**Fig. 3.** Performance of VA-files algorithms for *K*-CPQ with respect to the dataset sizes

In the second experiment, we have studied the behaviour of the VA-file-based algorithms for *K*-CPQ when the cardinality of the datasets varies. We have the following configuration: *dim* = 16, $|P| = |Q|$ = 15,000, 30,000, 45,000 and 68,040, $K$ = 100 and $b_d$ = 10. Fig. 3 shows that VA-DBSA is faster than VA-NODBSA, although it requires a smaller number of vector accesses (in the refinement step). In addition, we can also observe the effect of the increase of the size of the datasets for VA-NODBSA. This results to the increase of the consumed time and the increase of the memory-overhead, since more items have to be combined in the filtering step.

In the third experiment, we compare the performance of the VA-file-based algorithms (VA-DBSA = DB and VA-NODBSA = NO) with a nested loops algorithm only using vector files (NL) and with an R*-tree distance join algorithm (Rtree), varying the dimensionality (*dim* = 4, 8, 12, 16 and 32). Fig. 4 shows the performance measurements for the following configuration: $|P| = |Q|$ = 68,040, $K$=100, $b_d$ = 10 and, the maximum branching factors for R*-trees were 227, 120, 81, 62 and 31 for each *dim* value, using a node size of 8 Kb. When comparing the results of the *K*-CPQ algorithms with respect to the I/O activity and the elapsed time, we observe that this query becomes more expensive as the dimensionality grows, in particular for values larger than 16. Notice, also, that the huge number of pages accesses (the sum of the number of approximation and vector accesses in the VA-file structure) in all algorithms is due to the absence of global buffering. The R-tree version was the fastest in all cases (e.g. 5 times faster than NL for *dim* = 2), although for low and

medium dimensions it needed many page accesses. NL is also an interesting alternative with respect to the total elapsed time because the expensive filtering step is avoided, but for *dim* = 32 it obtained the largest value of page accesses. DB is better than NO for these two performance metrics, but the latter gets the minimum number of vector accesses after an expensive filtering phase over the two approximation files. For example, for *dim* = 32 the total number of vector accesses was 277 for NO and 2,811 for DB, whereas the number of approximation accesses was 17,227,051 and 8,255,818, respectively.

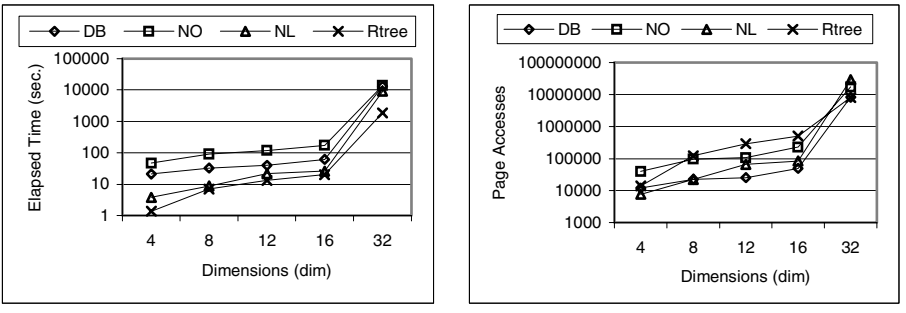


**Fig. 4.** Performance of distance-join algorithms where the dimensionality is increased

The forth experiment compares the performance of the VA-file-based algorithms (DB and NO) with NL and Rtree, varying $K$ from 1 to 100,000. Fig. 5 illustrates the performance measurements for the configuration: *dim* = 16, $|P| = |Q| = 68{,}040$, $b_d = 10$ and the maximum branching factor for R*-trees was 62. In the left chart, we see that the slowest was NO, due to its time and memory consumption, although it needs the minimum number of vector accesses (e.g. $K$ = 100,000, it was 83,033). The DB obtains interesting results for the total number of page accesses, when we have large $K$ values. NL reports very good results since it avoids the filtering step and only works over the vector files using the distance-based sweep technique. For example, it was the fastest and the cheapest in terms of I/O activity for small $K$ values (1 and 10). Finally, the results of the $K$-CPQ algorithm over R*-trees are very interesting as well, since it is the fastest for large $K$ values and it obtains a small number of page accesses, mainly due to the high pruning in the internal nodes on the R*-trees, the use of distance-based sweep technique and the use of large fan-outs of the R-tree nodes.

The last experiment studies the performance of the best VA-file-based algorithm (VA-DBSA), NL and the Rtree variant, for similarity join (SJ) using different $\delta$ values (0.001, 0.003, 0.005, 0.008, 0.01, 0.03 and 0.05). Fig. 6 illustrates the performance measurements for the configuration: *dim* = 16, $|P| = |Q| = 68{,}040$, $b_d = 10$ and the maximum branching factor for R*-trees was 62. We can deduce that the R-tree distance join algorithm using distance-based sweep technique is the best alternative. For example, it was 10.7 times faster than NL for $\delta = 0.001$ (in the result, each point has an average of 7.1 join mates) and 8.1 times for $\delta = 0.05$ (141.8 join mates per point). NL (the filtering step is not performed) is slightly faster than

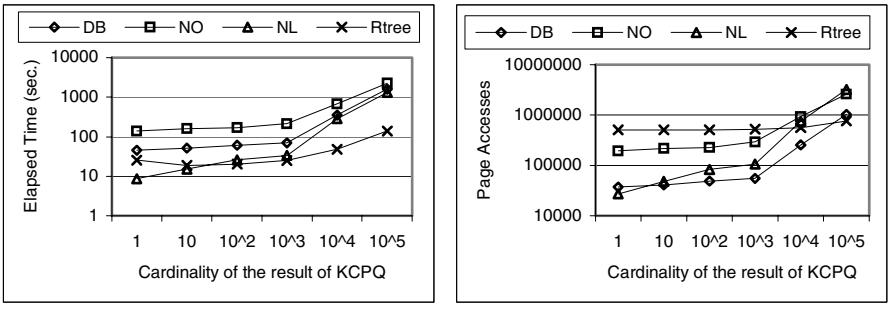


**Fig. 5.** Performance of *K*-CPQ when *K* is varied from 1 to 100,000

VA-DBSA, but it needs more page accesses. An interesting behaviour of the R-tree variant is that from $\delta = 0.01$ to $\delta = 0.05$, it needed 3.3 times more page accesses than for $\delta = 0.001$, whereas for VA-DBSA this was of 17.5 times.

From the previous performance comparison for real high-dimensional datasets, the most important conclusions are the following: (1) the filtering power of VA-file-based algorithms for DJQ is reduced when the dimensionality, cardinality of the datasets, *K* and $\delta$ are increased. (2) VA-NODBSA minimizes the number of vector accesses at the expense of time consumption and memory-overhead. (3) Including the distance-based sweep technique in the R-tree distance join algorithm improves notably its performance mainly with respect to the CPU cost. (4) And finally, the most important conclusion is that for DJQ where two real high-dimensional datasets are involved, the use of hierarchical multidimensional access methods (as R*-trees) with optimization techniques (like distance-based sweep) to the processing of index nodes (controlling the trade-off between I/O and CPU cost with respect to the page size [KoS01]) is the best alternative since its filtering power is increased, when *K* and $\delta$ are not very large (in this case the nested loops is the best alternative because it has no additional index overhead).

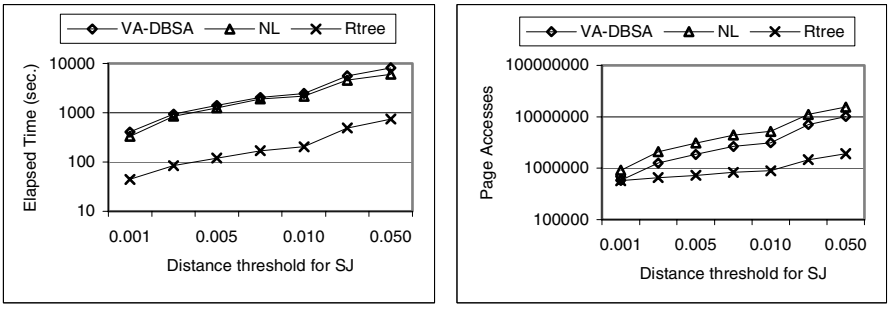


**Fig. 6.** Performance of SJ when δ is varied

## 6 Conclusions and Future Work

The contribution of this paper is twofold. (1) It reports the first development of algorithms for DJQ on pairs of high-dimensional data sets using VA-files. For this purpose, special bounds and pruning conditions have been proposed and employed. (2) It reports a detailed performance comparison of VA-files vs. R*-trees with respect to DJQ using real data. More specifically, for *K*-NNQs and distance range queries, where one real high-dimensional dataset and one query point are involved, one of the best alternatives to overcome the *dimensionality curse* is the use of VA-files (a filtering-based approach). For *K*-CPQs and SJs, where two real high-dimensional datasets are combined, this is not the best alternative with respect to the CPU cost, because the filtering step is overloaded, while it is competitive with respect to the I/O cost. The use of efficient hierarchical multi-dimensional access methods with optimization techniques in the processing of index nodes is a very interesting choice (since the filtering power can be improved notably). Future research may include the use of approximation techniques on VA-files [19, 8], the cost estimation of VA-file-based DJQ [19] and the study of the buffering impact over these DJQs, as in [4].

## References

1. Beckmann, N., Kriegel, H. P., Schneider, R., Seeger, B.: "The R*-tree: an Efficient and Robust Access Method for Points and Rectangles", Proc. SIGMOD Conf. (1990) 322-331
2. Berchtold, S., Böhm, C., Jagadish, H., Kriegel, H. P., Sander, J.: "Independent Quantization: an Index Compression Technique for High-Dimensional Data Spaces", Proc. ICDE Conf. (2000) 577-588
3. Böhm, C., Braunmuller, B., Breuning, M. M., Kriegel, H. P.: "High Performance Clustering based on Similarity Join", Proc. CIKM Conf. (2000) 298-305
4. Böhm, C., Kriegel, H. P.: "A Cost Model and Index Architecture for the Similarity Join", Proc. ICDE Conf. (2001) 411-420
5. Cha, G. H., Chung, C. W.: "The GC-tree: a High-Dimensional Index Structure for Similarity Search in Image Databases", Transactions on Multimedia, Vol. 4, No. 2 (2002) 235-247
6. Cha, G. H., Zhu, X., Petkovic, D, Chung, C.W.: "An Efficient Indexing Method for Nearest Neighbor Searches in High-Dimensional Image Databases", Transactions on Multimedia, Vol. 4, No. 1 (2002) 76-87
7. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: "Algorithms for Processing K-Closest-Pair Queries in Spatial Databases", Data and Knowledge Engineering Journal, Vol. 49, No. 1 (2004) 67-104
8. Corral, A., Vassilakopoulos, M.: "On Approximate Algorithms for Distance-Based Queries using R-trees", The Computer Journal, Vol. 48, No. 2 (2005) 220-238
9. Cui, B., Hu, J., Shen, H., Yu, C.: "Adaptive Quantization of the High-Dimensional Data for Efficient KNN Processing", Proc. DASFAA Conf. (2004) 302-313
10. Dittrich, J. P., Seeger, B.: "GESS: a Scalable Similarity-Join Algorithm for Mining Large Data Sets in High Dimensional Spaces", Proc. SIGKDD Conf. (2001) 47-56
11. Faloutsos, C., Barber, R., Flickner, M., Hafner, J., Niblack, W., Petkovic, D., Equitz, W.: "Efficient and Effective Querying by Image Content", Journal of Intelligent Information System, Vol.3, No.3-4 (1994) 231-262

12. Ferhatosmanoglu, H., Tuncel, E., Agrawal, D., Abbadi, A. E.: "Vector Approximation Based Indexing for Non-Uniform High Dimensional Data Sets", Proc. CIKM Conf. (2000) 202-209
13. Guttman, A.: "R-trees: a Dynamic Index Structure for Spatial Searching", Proc. SIGMOD Conf. (1984) 47-57
14. Koudas, N., Sevcik, K. C.: "High Dimensional Similarity Joins: Algorithms and Performance Evaluation", Transactions on Knowledge and Data Engineering, Vol. 12, No. 1 (2000) 3-18
15. Korn, F., Sidiropoulos, N., Faloutsos, C., Siegel, C., Protopapas, Z.: "Fast Nearest Neighbor Search in Medical Images Databases", Proc. VLDB Conf. (1996) 215-226
16. Nanopoulos, A., Theodoridis, Y., Manolopoulos, Y.: "$C^2P$: Clustering based on Closest Pairs", Proc. VLDB Conf. (2001) 331-340
17. Sakurai, Y., Yoshikawa, M., Uemura, S., Kojima, H.: "The A-tree: an Index Structure for High-Dimensional Spaces using Relative Approximation", Proc. VLDB Conf. (2000) 516-526
18. Shim, K., Srikant, R., Agrawal, R.: "High-Dimensional Similarity Joins", Proc. of ICDE Conf. (1997) 301-311
19. Weber, R., Böhm, K.: "Trading Quality for Time with Nearest Neighbor Search", Proc. EDBT Conf. (2000) 21-35
20. Weber, R., Schek, H. J., Blott, S.: "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces", Proc. VLDB Conf. (1998) 194-205
21. Web site: http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html