Information Processing Letters 82 (2002) 269–276

Information Processing Letters

www.elsevier.com/locate/ipl

Processing a multimedia join through the method of nearest neighbor search

Harald Kosch [a,*], Solomon Atnafu [b]

[a] *Institute of Information Technology, University Klagenfurt, Austria*
[b] *Information Systems Engineering Lab, INSA de Lyon, France*

Received 30 June 2000; received in revised version 1 March 2001
Communicated by A. El Abbadi

Keywords: Multimedia databases; Processing of a multimedia join; Nearest neighbor search

1. Introduction and motivation

Commonly used content-retrieval systems focus on the problem of finding the nearest neighbor (NN-search) for a given single query object out of a database of media objects [1]. However, there are only few attempts [2,3] that realize join operations on two multimedia tables, where the multimedia data components are represented by their respective feature vectors. The necessity of using multimedia joins in a variety of applications is the motivation behind this search for a more efficient and more general purpose method of performing a join on multimedia tables.

In this perspective, the goal of this paper is to introduce an efficient implementation of such a multimedia join using the method of NN-search. The problem is naturally related to the NN-search for a single query object which suggests a straightforward nested-loop implementation. We will show that this implementation can be considerably improved by extending the notion of a query object to a query-sphere (Section 3). Finally we will demonstrate experimentally that our implementation decreases considerably the number of index partitions to be accessed (Section 4).

Our current work is on the management of image data, but the techniques can be extended to other types of media data such as video and audio. As long as the feature vector representations are used for the content of the media data there is a way to extend our work to video and audio databases.

1.1. Motivation

Let us consider a sample multimedia investigation scenario (housebreaking during summer holidays in a residential district) to motivate the usefulness of a multimedia join. Let us suppose that we have to manage two sets of images with its descriptions: images of housebreakers *BL* and scanned images, *SI*, of individuals who appeared at the entry point of the district.

In order to implement the NN-search in a DBMS, a proper image repository model has to be introduced. With respect to the state of the art in image DBMS (see Oracle's *inter*Media, Informix's Image DataBlades, and the DISIMA DBMS [3]) we propose an object-relational (OR) paradigm for storing and manipulating

[*] Corresponding author.
E-mail address: harald.kosch@itec.uni-klu.ac.at (H. Kosch).

PII: S0020-0190(01)00279-4

image objects and their related data. Our OR image table $MM(o, fv, a)$ contains a component o which is the image object itself. fv is a feature vector representation of the object o and a is an attribute component that may be used to describe the object. a may be declared to be an object type or a set of object types.

In the concrete scenario, the image tables are as follows. *BL* contains the images of housebreakers and their corresponding *fv*, as well as the names, addresses, and information on previous crimes. *SI* consists of: the scanned images, their corresponding *fv*, and the date and time at which each of the images were scanned. Suppose now, there is an investigation scenario for a housebreaking incident, say on August 24 between 4–6 PM, in the locality where the surveillance camera is mounted. *SI* alone cannot give a complete set of information on the suspects. It is therefore required to perform some operations on the two tables in order to get fuller information. The following is a possible query to retrieve the list of suspected persons with records for housebreaking crime. *For the pictures in SI, shot or filmed on August* 24 *between* 4–6 *PM, find the corresponding most similar* (*best* 5) *photos that are in the image table of criminals BL, plus their corresponding name and address.*

This query performs a relational selection on the date and time attribute of *SI*. That is, it selects only those instances of *SI* where the date and time of the photo shot is August 24 between 4–6 PM. Then, it does a *multimedia join* operation between the selected instances of *SI* and the image table *BL*. The multimedia join is a binary operation which computes for each image stored in the left input table (the selected *SI*), the k-nearest neighbors in the set of images stored in the right input table, *BL*. Comparison for similarity of two images, actually under consideration in the join, is done by computing the relative distance between the feature vectors *fv*, based on a certain metric.

Let us now give some important definitions for the remainder of the paper.

Definition 1 (*k-nearest neighbor search*). Given a set of images S, a query image q, and a positive integer k; the k-nearest neighbors to the query image q denoted as $NN_k(S, q)$ are the first k images that are a shorter distance from q in the feature space than any of the other images in S. More formally:

$$\begin{aligned} &NN_k(S, q) \\ &\quad = \bigl\{\{o'_1, o'_2, \ldots, o'_k\} \mid o'_i \in S \text{ for } i = 1, \ldots, k \text{ and} \\ &\qquad\qquad \|o'_i - q\| \leqslant \|o - q\| \\ &\qquad\qquad \forall o \in S \setminus \{o'_1, o'_2, \ldots, o'_k\}\bigr\}, \end{aligned}$$

where $\|o - q\|$ stands for the distance between the object o and the query object q. The distance is commonly computed using a certain metric such as the Euclidean distance metric.

A multimedia join finds the k-nearest images in the set of images stored in the right input (inner) table for each image o_1 stored in the left input (outer) table. The definition is given below.

Definition 2 (*Multimedia join operation*). Let $MM_1(o, fv, a)$ and $MM_2(o, fv, a)$ be two image tables, where the dimensionality and format of *fv* of MM_1 and *fv* of MM_2 are the same. The multimedia join operation (*MM-join*) through the method of k-NN-search is formally defined as:

$$\begin{aligned} &MM_1 \bowtie_{NN_k} MM_2 \\ &\quad = \bigl\{(t_1, t_2) \mid t_1 = (o_1, fv_1, a_1) \in MM_1 \text{ and} \\ &\qquad\qquad t_2 = (o_2, fv_2, a_2) \in MM_2, \text{ and} \\ &\qquad\qquad o_2 \in NN_k(MM_2, o_1)\bigr\}, \end{aligned}$$

where k is a positive integer.

Example. The algebraic expression for the query in the motivation is:

$$\Pi_{BL.a.name, BL.a.address}\bigl(\sigma_{SI.a.time, SI.a.date}(SI) \bowtie_{NN_5} BL\bigr),$$

where $\sigma_{SI.a.time, SI.a.date}$ denotes the relational selection operator on the specified time and date of the crime and $\Pi_{BL.a.name, BL.a.address}$ is the relational projection.

2. Related work

During the last decade, several systems that support content-based query have been proposed (see the review in [1]). Some of the commonly known prototypes are systems such as MARS [2], DISIMA [3] and CHITRA [4]. Though many works exist, there are very little of them that consider a multimedia join operation that associates two sets of data for similarity. For example, the MARS system allows complex query formulation by an intelligent query refinement tool for the user-interaction, but does not support a definition of similarity-based join. The DISIMA system accepts queries in MOQL [3] that extends OQL by adding new predicate expressions. However, the similarity-based join proposed in DISIMA results in pairs of instances from the two input tables for which a user-defined threshold similarity value governing the difference of the respective feature vectors has been exceeded. The CHITRA system uses a fuzzy object query language (FOQL) [4] that is an extension of OQL, however a multimedia join operation cannot be specified.

Other relevant work comes from the domain of spatial databases. Typical "spatial join queries" involve two or more sets of spatial data and combine pairs (or tuples) of spatial objects that satisfy a given predicate. For instance, one might be interested in finding all the pairs of objects that intersect with each other (intersection join). Brinkhoff et al. present in [5] a very detailed analysis of different implementation strategies for an intersection join. They show that a spatial sort-merge-based join based on a plane sweep technique outperforms a nested-loop one. However, the plane sweep technique proposed for an intersection join cannot apply for the purpose of our multimedia join, since they have different definitions (if in our join, projections of bounding boxes intersect, it cannot be guaranteed that a nearest neighbor has been found) and deal with different data sets (dimension of the feature vectors is in general higher than that of spatial data).

Recent papers in spatial databases attempt to combine spatial join queries with an NN-search. Hjaltason and Samet [6] and Corral et al. [7] define a "distance join" between two input sets which computes the K-closest pairs of two input spatial object sets, ordered by the distance of objects in each pair. In addition, Hjaltason and Samet [6] propose a "distance semi-join" which groups the results of the distance join by the objects of the outer table retaining the pairs of objects with closest distance.

Let us now consider these works with reference to our image database (where the points in the index represent feature vectors). It is important to stress that the definitions of distance join and distance semi-join are different from the definition of our multimedia join. We will illustrate the difference in a concrete example. Consider that two image tables A and B have three instances each $a_i \in A$ and $b_i \in B$ $(1 \leqslant i \leqslant 3)$. The distance between the instances of A and B are shown in Table 1.

Table 1

	a_1	a_2	a_3
b_1	1	5.1	2
b_2	2	3	2.2
b_3	1	4	1.4

Then, the distance join generates the following pairs for $K = 6$ (i.e., 2/3 of the Cartesian Product is generated): (a_1, b_1), (a_1, b_3), (a_3, b_3), (a_1, b_2), (a_3, b_1), (a_3, b_2). The distance semi-join computes in this example as: (a_1, b_1), (a_3, b_3).

The multimedia join (for $k = 1$), however would produce: (a_1, b_1), (a_2, b_2), (a_3, b_3). For $k = 2$, it would produce: (a_1, b_1), (a_1, b_3), (a_2, b_2), (a_2, b_3), (a_3, b_3), (a_3, b_1). The distance join cannot provide a comparable case for $k > 1$.

We realize here that $a_2 \in A$ and its NN is not in the result of the distance semi-join (i.e., a_2 has been discarded in the distance join), thus information on a_2 got lost. In order to guarantee for the distance semi-join that all objects of the outer table are considered, we would have to generate the Cartesian Product in the distance join, but this is not practical in many cases.

3. Processing a multimedia join through the method of NN-search

Efficient processing of a multimedia join requires multidimensional index structures which make it possible to find images that are similar to a query image, while using only few index page accesses. Existing index structures for high-dimensional feature spaces can be classified into Data Partitioning (DP) based and Space Partitioning (SP) based structures [8]. DP-based index structure consists of bounding regions (BRs) arranged in a spatial containment hierarchy (e.g., R-, X-, SS-, and TV-trees) [9,8], as SP-based index structure consists of a space that is recursively partitioned into mutually disjoint subspaces (e.g., kDB- and hB-trees) [10].

In this paper, the NN-search method proposed for X-trees [9] is used as a reference implementation.

The proposed join algorithms work however with different DP-based index structures that use a hierarchical directory structure. In order to meet the different bounding predicates of DP-based index structures, changes must be made in the definitions of the *mindist*() and *minmaxdist*() functions between a query point and a leaf partition (which can be looked up in the respective papers (e.g., for SS-trees in [11])), but not in the definitions of the functions *mindist*() and *minmaxdist*() between a query-sphere and a leaf partition. Extensions of our algorithms to SP-based index structures are the adaptation of the index traversal such that the minimum number of possible points are visited.

3.1. *Simple nested-loop implementation*

A straightforward method for performing an MM-join is to directly apply the algorithm for the NN-search for each object of the outer table as a query object looking for its closest objects from the inner table. The tuple containing a data object of the outer table and tuples containing its selected nearest neighbor objects from the inner table are concatenated to form the resulting instance.

This algorithm has a nice property that enables us to reuse the NN-search implementation for a single query object. However, this algorithm can be improved if the NN-search is modified to cluster query points in a query hypersphere.

3.2. *Optimized algorithm for the MM-join based on query point clustering*

The main idea of our optimized algorithm lies in the way we consider objects in the leaf partition of the outer table index. The data objects considered in the outer table index are those that are clustered together on a relatively small BR. This spatial proximity can be exploited when searching for the nearest point in the index of the inner table by searching not only for the nearest neighbor of a single query object, but by searching for the nearest neighbor of a whole hypersphere containing all data objects of this leaf (here called the query-sphere). The other advantage of this new approach is that, for each object of the outer table, we need no longer traverse the whole index of the inner table. This reduces significantly the number of accessed leaf partitions of the inner table index (i.e., disk pages accessed). Note that, in this join the number of data objects which can be stored in a page is substantial. For instance, a page of size 4 KB can hold 32 feature vectors of dimensionality 16 where each object coordinate is assumed to be stored in 8 bytes.

Fig. 1 shows the general algorithm for the optimized MM-join implementation. The algorithm traverses the leaf partitions of the outer table index. For each leaf partition *Pi* of the outer table index, a query-sphere S that contains all the data objects of *Pi* is determined. As we are no longer dealing with a single query object, the nearest neighbor search method has to be partially redesigned. The data structure has to be redesigned in such a way that we maintain a list of candidate nearest neighbor objects *Candidate* and their respective leaf partitions sorted by their distance to the farthest away object in S. Furthermore, new distance functions *mindist*() and *minmaxdist*() that compute the respective distances between a query-sphere and the partitions have to be introduced and the pruning policy has to be updated.

The general principle is that the NN-search for the query-sphere S traverses the inner table index and for every partition P visited stores a list of sub-partitions ordered by their $\mathit{mindist}_{\mathit{Sphere}}()$. $\mathit{mindist}_{\mathit{Sphere}}(P, S)$ is the minimum possible distance from the partition P to the nearest possible point in S, i.e., $\mathit{mindist}_{\mathit{Sphere}}(P, S)$ can be reduced to the $\mathit{mindist}(P, Q)$ between a partition P and a query object Q by the following means (see also Fig. 2). The point C denotes the center of the query-sphere S and r its radius. Assuming an Euclidean distance we have:

$$\mathit{mindist}_{\mathit{Sphere}}(P, S) = \begin{cases} \mathit{mindist}(P, C) - r & \text{if } \mathit{mindist}(P, C) - r \geqslant 0, \\ 0 & \text{otherwise,} \end{cases}$$

$\mathit{mindist}(P, Q)$ and $\mathit{minmaxdist}(P, Q)$ for X-trees are defined in [9], for SS-trees in [11] (see also above).

If the algorithm visits a leaf, it first computes the data object of the leaf which has the smallest distance to the farthest away object in the hypersphere S. This object NN_{ref} is used to prune the list of partitions *PL* and the list of candidate nearest-neighbor objects *Candidate*. $\mathit{Prune}(NN_{ref}, \underline{\mathit{var}}\ \mathit{Candidate}, \underline{\mathit{var}}\ PL)$ is the procedure proposed to process the pruning. It takes as input two variable parameters *Candidate*

```
Foreach leaf partition $Pi$ of the outer table index Do
Compute the query-sphere $S$ with radius $r$ around the pivot element of the par-
tition $Pi$, such that all data elements in the leaf partition are contained in $S$.
//Search the nearest neighbor of $S$ in the inner table index
Let $Candidate$ be a list of tuples of the form (object, leaf),
sorted by $distance_{Sphere}(object, S)$. Initialize $Candidate = \{\}$.
Initialize $PL$ with the sub-partitions of the inner table root-partition.
Sort elements in $PL$ by $mindist_{Sphere}()$.

While $PL \# \{\}$ Do
    If first element partition $P$ of $PL$ is a leaf Then
        Find nearest neighbor object $NN_{ref}$ to $S$ in $P$.
        $Candidate := Candidate \cup (NN_{ref}, P)$.
        // Do the pruning of $PL$ and $Candidate$
        $Prune(NN_{ref}, Candidate, PL)$.
    Elsif $\nexists P' \in PL, mindist_{Sphere}(P, S) \geqslant minmaxdist_{Sphere}(P', S)$
    Then Replace $P$ in $PL$ with its son nodes.
            Resort $Candidate$ by $distance_{Sphere}(object, S)$.
    Else Remove $P$ from $PL$. Endif
    Resort $PL$ by $mindist_{Sphere}()$.
EndWhile
For each data object in $S$ calculate the NN from $Candidate$ and concatenate them.
EndForeach
```

Fig. 1. Optimized algorithm for the MM-join.

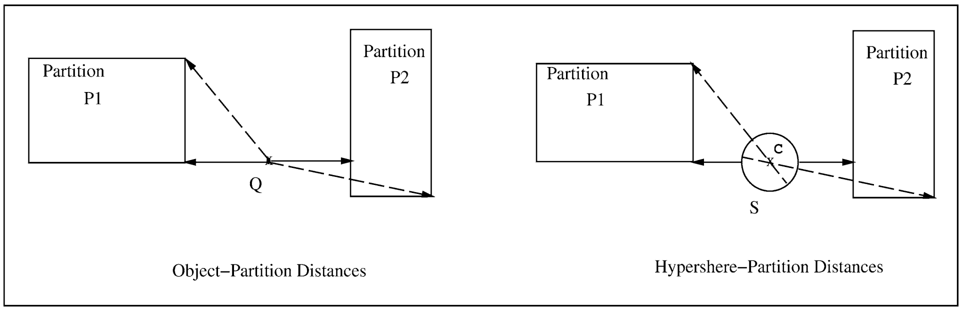


Fig. 2. Distance measures: *mindist*() (solid lines) and *minmaxdist*() (dotted lines). Single query object Q on the left. Query-sphere S on the right.

and PL. First, all partitions $P \in PL$ that have $mindist_{Sphere}(P, S)$ larger than the distance of NN_{ref} to the farthest away object in the hypersphere S ($= distance_{Sphere}(NN_{ref}, S)$) are removed from the list PL. Then, the list of candidate nearest-neighbor objects *Candidate* is pruned, i.e., an $object \in Candidate$ is removed from *Candidate*, if:

$$distance_{Sphere}(NN_{ref}, S) \leqslant distance_{Sphere}(object, S).$$

The function $distance_{Sphere}(object, S)$ can be defined with the help of *distance*() in the context of a single query object as follows:

$$distance_{Sphere}(object, S) = distance(object, C) + r,$$

where C denotes once again the center of the hypersphere S and r its radius.

Upon visiting an internal partition P its successor nodes replace P in the list of partitions PL, only when there exists no other partition $P' \in PL$ such that its $minmaxdist_{Sphere}(P', S)$ is smaller than $mindist_{Sphere}(P, S)$. $minmaxdist_{Sphere}(P, Q)$ of a partition P with regard to the query-sphere S is the maximum possible distance from the farthest away object in S with respect to P to the nearest data object inside a partition P. It can be defined by using *minmaxdist*() of a query object Q and a partition P, as follows (see also Fig. 2). The point C is the center of the hypersphere S and r its radius:

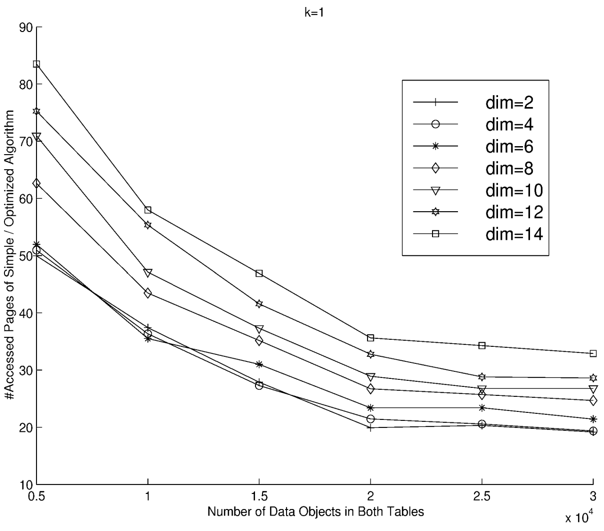
$$minmaxdist_{Sphere}(P, S) = minmaxdist(P, C) + r.$$

The branch-and-bound part of the algorithm terminates when *PL* becomes empty, thus no more internal nodes are to be exploited. The final operation is to search for each object in S, the corresponding nearest objects from the remaining candidate leaf partitions in *Candidates*.

4. Experimental results

We implemented in C++ the nested-loop and optimized algorithms of the NN-search based on the X-tree as reference implementation [9]. The effective capacity of an X-tree page was $512/dim$ data objects, supposing the example settings of the last section. All experiments were carried out on a Pentium III, 450 MHz with 128 MB main memory and 21 GB of storage device. The program simulates the *buffer management* of a DBMS. We employed the settings of Berchtold et al. [9] and assumed that the buffer size reserved for each X-tree involved in the multimedia join depends on the number of index nodes *num_nodes*, the dimension *dim* and the page size *page_size*, thus computes as: $\lceil (12.5 * dim * num\_nodes)/page\_size \rceil$. The first internal levels of the index tree are initially kept in the buffer, whereas a minimum the root node of each X-tree has to be held in the buffer. The sum of the cached pages of the two input X-trees is limited by 256 pages. If it is exceeded, the number of cached pages for each index is reduced proportionally.

The experimental analysis was carried out on two distinct datasets, one *synthetic* (*uniform dataset*) and one *real* (results of the real datasets can be found in the extended version [12]). The *performance metric* is firstly the number of page accesses to the disk that are performed during the MM-join (i.e., page accesses of both tables) and secondly the running time of the algorithms (CPU-time). Each experiment has been performed 100 times (starting from the initial buffer each time) and the mean value has been considered. The confidence interval tests of the results indicate that, 98% of the results are within 7% of the mean in almost all cases.



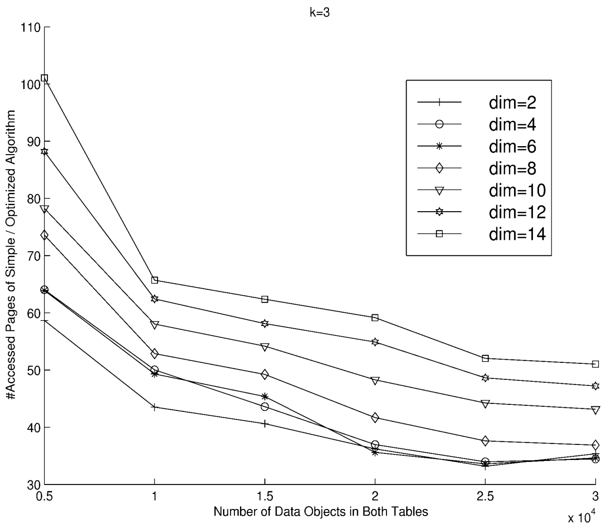


Fig. 3. Page access improvement factor optimized over the simple algorithm.

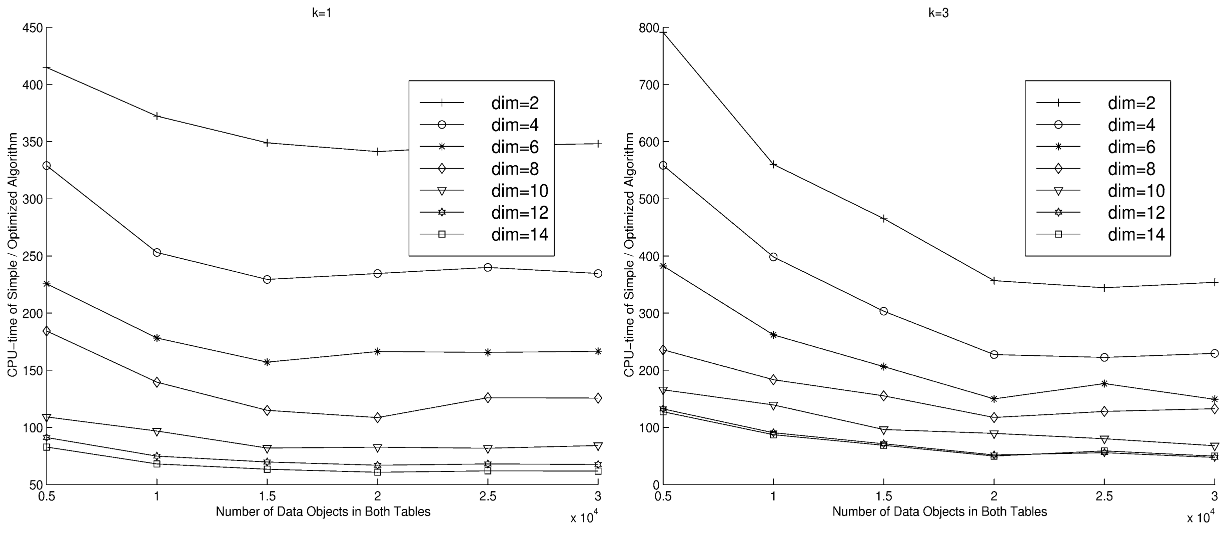


Fig. 4. CPU-time improvement factor optimized over the simple algorithm.

Fig. 3 (for page accesses) and Fig. 4 (for CPU-time) show the improvement factor of the optimized over the simple algorithm with respect to the variation of the dimension, the number of data objects, and the number k of computed NN. On the left side of each figures, the number of accessed disk pages is shown for $k = 1$ and on the right side for $k = 3$. The case of $k = 5$ is not shown due to the lack of space, but shows similar characteristics.

Fig. 3 shows that the highest page access improvement factor is achieved independently from the dimension for $N = 5000$. The factor decreases towards 10,000 data objects. However, for a number of data objects above 20,000 (for $k = 1$) and 25,000 (for $k = 3$) the improvement factor remains nearly constant. For example, for $dim = 14$ the factors hovers at 52 for $k = 3$ and at 35 for $k = 1$. At the same time, for whatever number of objects, the improvement factor increases with the dimension. The improvement factor for $k = 3$ is on average 23% higher than that for $k = 1$, and the improvement factor for $k = 5$ is on average 21% higher than that for $k = 3$. The higher search effort in the inner table shows a greater advantage for the optimized algorithm over the simple one.

Fig. 4 shows that the CPU-time improvement factor decreases with higher dimensions, however the decrease becomes less important, e.g., in the case of $k = 1$, for $d = 8$ the factor is on average 133.2, for $d = 10$ it is 89.5, for $d = 12$ it is 73.1, and $d = 14$ it is 66.5. This is the inverse result of the page access metric, where the higher improvement factors are achieved for higher dimensions. The reason is that with a higher dimension less query objects are contained in a page and the sum of CPU-time spent for all query objects of one outer table leaf partitions becomes less important compared to the CPU-time spent in the optimized algorithms. Contrarily, for the disk access metric, the impact of the dimension (we observe that for dimensions above 6 a high increase in the number of page accesses for the simple algorithm) dominates the results. Similar to the access page metric, for a number of data objects above 15,000 (for $k = 1$) and 20,000 (for $k = 3$) the CPU-time improvement factor remains nearly constant. Once again we note that the improvement factor for $k = 3$ (right-hand figure) is on average 24.5% higher than that for $k = 1$ (left-hand figure), and the improvement factor for $k = 5$ is on average 21% higher than that for $k = 3$. Finally, one notices that the CPU-time improvement factor is higher than the page access one.

In general, the access page and CPU-time improvement factors are significant with respect to the dimension, the number of data objects in both input tables and the number of nearest neighbors computed. This argues strongly for the use of the optimized MM-join implementation.

5. Conclusion and future work

This paper focused on processing a multimedia join through the method of the NN-search. We first introduced a simple nested-loop solution which applies for all data objects of the outer table an NN-search in the set of objects stored in the inner table. We then proposed a novel optimized strategy which takes advantage of query point clustering in a hypersphere. Several experiments have been performed to demonstrate the efficiency of the optimized algorithm over the simple one for different data sizes, dimensions and number of nearest neighbors to be searched. Future research will focus on formalizing the similarity-based operators on multimedia databases and on developing a similarity-based algebra.

References

[1] Y. Rui, T.S. Huang, S.-F. Chang, Image retrieval: Past present and future, J. Visual Comm. Image Representation 10 (1999) 1–23.

[2] K. Porkaew, M. Ortega, S. Mehrotra, Query reformulation for content based multimedia retrieval in MARS, in: Proc. IEEE International Conference on Multimedia Computing and Systems, Florence, Italy, Vol. 2, June 1999.

[3] V. Oria, M.T. Özsu, B. Xu, L.I. Cheng, P. Iglinsk, DISIMA: An object-oriented approach to developing an image database system, in: Proc. Internat. Conference on Data Engineering (ICDE), San Diego, CA, February–March 2000, pp. 672–673.

[4] S. Nepal, M.V. Ramakrishna, Query processing issues in image (multimedia) databases, in: Proc. Internat. Conference on Data Engineering (ICDE), Sydney, Australia, March 1999, pp. 22–29.

[5] T. Brinkhoff, H.-P. Kriegel, B. Seeger, Efficient processing of spatial joins using R-trees, in: Proc. ACM SIGMOD Internat. Conf. on Management of Data, Washington, DC, June 1993, pp. 10–15.

[6] G.R. Hjaltason, H. Samet, Incremental distance join algorithms for spatial databases, in: Proc. ACM SIGMOD Conf. on Management of Data, Seattle, WA, June 1998, pp. 237–248.

[7] A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos, Closest pair queries in spatial databases, in: Proc. ACM SIGMOD Conf. on Management of Data, Dallas, TX, May 2000, pp. 189–200.

[8] K. Chakrabarti, S. Mehrotra, The hybrid tree: An index structure for high dimensional feature spaces, in: Proc. Internat. Conf. on Data Engineering (ICDE), Sydney, Australia, March 1999, pp. 440–447.

[9] S. Berchtold, D.A. Keim, H.-P. Kriegel, The X-tree: An indexing structure for high-dimensional data, in: Proc. VLDB Conference, Bombay, India, September 1996, pp. 28–39.

[10] G. Evangelidis, D. Lomet, B. Salzberg, The hB$^{\pi}$ tree: A multi-attribute index supporting concurrency, recovery and node consolidation, VLDB J. 6 (1) (1997) 1–25.

[11] R. Kurniawati, J.S. Jin, J.A. Shepherd, The SS+-tree: An improved index structure for similarity searches in a high-dimensional feature space, in: Proc. SPIE Storage and Retrieval of Image and Video Databases, San Jose, CA, February 1997, pp. 110–120.

[12] H. Kosch, Processing a multimedia join through the method of nearest neighbor search (extended version), Technical Report TR/ITEC/01/2.02, ITEC, Univ. Klagenfurt, 2001, http://www-itec.uni-klu.ac.at/∼harald/ipllong.pdf.