Processing a Multimedia Join through the Method of Nearest Neighbor Search

Harald Kosch\textsuperscript{a} and Solomon Atnafu\textsuperscript{b}

\textsuperscript{a}Institute of Information Technology, University Klagenfurt, Austria
\textsuperscript{b}Information Systems Engineering Lab, INSA de Lyon, France

Abstract

This paper presents a multimedia join operator that is carried out through the method of the nearest neighbor search. In contrast to related approaches that utilize a similarity function to perform a join between two instances of the input tables, we adopt the more flexible and widely used nearest neighbor method. First, we introduce a simple nearest neighbor search algorithm based on an nested-loop execution strategy, second an optimized version is proposed which takes advantage of query point clustering in a hypersphere. Several experiments are performed to demonstrate the efficiency of the optimized algorithm over the simple one for different datasets, datasizes and dimensions.

Key words: Multimedia Databases, Processing a Multimedia Joins, 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 \cite{1}. However, there are only few attempts \cite{2,3} that consider 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

\textit{Email addresses:} Harald.Kosch@itec.uni-klu.ac.at (Harald Kosch), Solomon.Atnafu@insa-lyon.fr (and Solomon Atnafu).
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 (see Section 4). Finally we will demonstrate experimentally that our implementation decreases considerably the number of index partitions to be accessed (see Section 5).

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.

**Motivation** Let us consider a sample multimedia investigation scenario (housebreaking during the summer holidays in a residential district) to motivate the usefulness of a multimedia join. Let us suppose that we have to manage three sets of images with its content descriptions: images of housebreakers $BL$, images of residents $RES$ and scanned images, $SI$, of individuals who appeared at the entry point of the residential 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 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 as an object- 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. $RES$ contains photos of the residents, the corresponding $fv$, and names, addresses, etc. $SI$ 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 three 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.

Q1) 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.

Imagine now, from the result of this query, we know that one of the sus-
pects was a resident of the residential compound under consideration but had changed his/her address after his/her last record in BL and BL has not been updated. Here we observe that we cannot do a relational join between BL and RES on the attributes name or address, because the suspect is estimated to have changed his identity from that recorded in the police files. Hence, we expect that the following query (query Q2) will discover the required suspects.

Q2) Get the name and address of the residential persons from RES whose photos are among the best matches to the pictures in BL that are selected as the most similar (best 5) to the images in SI shot on August 24 between 4-6 PM.

First, Query Q1 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. Query Q2 requires two multimedia joins. The first is a multimedia join between SI and BL identical to query Q1, the result table of which is then taken as left input to the second join. The right input is RES. In other words, the first join identifies pairs of tuples from BL that are most similar (the best 5) to the images in SI shot on August 24 between 4-6 PM, whereupon a multimedia join with RES is performed.

2 Problem Definition

In an image database, a k-Nearest Neighbor (k-NN) search method is the retrieval of a set of $k$ ($k \leq 1$) objects from a larger set of image data objects that are similar to a given query object, using their feature vector representations [1]. Formally, the k-NN search is defined as follows:

**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:

$$NN_k(S,q) = X \iff X \subseteq S \land Card(X) = k \land \forall x \in X; y \in (X \setminus S) \bullet \|x - q\| \leq \|y - q\|.$$  

Note that $\|x - y\|$ stands for the distance between the object $x$ and the query object $q$. The distance is commonly computed using a certain metric such as
the Euclidean distance metric. For more information on other metrics, please consider [4]. The value of $k$ is assumed to be less than the cardinality of $S$ and at the instance where there more than one image at the $k^{th}$ closest point to $q$ exists, the $k$-NN search algorithm selects one randomly. For $k = 1$, it searches for the most similar object.

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:

$MM_1 \bowtie_{NN}^k MM_2 = X \iff X \in MM_1 \times MM_2 \land \forall ((p, fv_p, a_p), (q, fv_q, a_q)) \in X \bullet q \in NN_k(MM_2, p)$.

**Example:** The algebraic expression for Query Q1 of Section 1 can be expressed as:

$\Pi_{BL.a.name, BL.a.address}(\sigma_{SI.a.time, SI.a.date}(SI) \bowtie_{NN_5} BL)$, where $\sigma_{S.a.time, S.a.date}$ denote 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.

The algebraic expression for Query Q2 may be written as:

$\Pi_{RES.a.name, RES.a.address}(\Pi_{BL.o, BL.fv, BL.a}(\sigma_{SI.a.time, SI.a.date}(SI) \bowtie_{NN_5} BL) \bowtie_{NN_1} RES)$.

### 3 Related works

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 [5], DISIMA [3] and CHITRA [6]. 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 [7] 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) [8] that is an

---

1 This principle is also applied in Oracle’s interMedia (http://technet.oracle.com) and Informix Image DataBlades (http://examples.informix.com).
extension of OQL, however a multimedia join operation cannot be specified.

There are some works done to explore the algebraic space required for a similarity-based query optimization. Adali. et al. [2], for example, proposes a multi-similarity algebra that integrates heterogeneous similarity measures coming from different similarity implementations in one common framework. However, this work deals at a higher abstraction level and does not provide an implementation level. P. Ciaccia et al. [9] extends the work of S. Adali et al. with their development of the $SAME^W$ algebra. This algebra introduces further user preferences such as weights, and also captures imprecision in feature representations. However, the implementation issues of the complex operations introduced are not addressed.

Other relevant works come from the domain of spatial databases. Typical "spatial join queries" involve two or more (see e.g., [10]) 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 [11] 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 et al. [12] 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 the same paper, the authors 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. The works in [12] and [13] deal with spatial datasets.

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. The distance join produces ordered pairs of the input tables, whereas we employ the $k$-Nearest Neighbor ($k$-NN) search for each object of the outer table. Moreover, our multimedia join can guarantee that for all objects of the outer table, its $k$-NN (for $k \geq 1$) are computed. However, the distance semi-join may leave out some objects of the outer table.

Corral et al [13] reconsider the problem (called $K$-Closest Pair Query ($K$-CPQ) in [13]) and improve the implementation proposed in [12].
without a reference to its closest object in the inner table. Furthermore, it only retains the nearest pairs, not the k-nearest as in the multimedia join.

Let us illustrate this difference in a concrete example. Consider that two image tables $A$ and $B$ have three instances each $a_i \in A$ and $b_i \in B$ ($1 \leq i \leq 3$). The distance between the instances of $A$ and $B$ are shown in table to the right side.

<table>
<thead>
<tr>
<th></th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_1$</td>
<td>1</td>
<td>5.1</td>
<td>2</td>
</tr>
<tr>
<td>$b_2$</td>
<td>2</td>
<td>3</td>
<td>2.2</td>
</tr>
<tr>
<td>$b_3$</td>
<td>1</td>
<td>4</td>
<td>1.4</td>
</tr>
</tbody>
</table>

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.

4 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 [14]. DP-based index structure consists of bounding regions (BR) arranged in a spatial containment hierarchy (e.g., R-, X-, SS-, and TV-trees) [15,14], as SP-based index structure consists of a space that is recursively partitioned into mutually disjoint subspaces (e.g., kDB- and hB-trees) [16].

In this paper, the NN-search method proposed for X-trees [15] 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 [17])), 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.
In practice, if the distance computed between the query object and the object to be searched for is too big (this is, of course, a relative notion!), one must say that the two objects are too far apart to be similar. A common solution is to say that two objects are similar enough if the distance between their feature vectors is less than or an upper similarity value $\varepsilon$. If a system knows how to estimate $\varepsilon$, the NN-search in fact only searches in the hypersphere of the $\varepsilon$ radius. The proposed algorithms can easily be adapted to deal with this case. The real problem here is to determine this value beforehand, as we do not know to what extent the data points are congested or dispersed in the image data space. This would be an appropriate subject for further research.

**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.

**Optimized Algorithm for MM-join Implementation 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
Foreach leaf partition $P_i$ of the outer table index Do
Compute the query-sphere $S$ with radius $r$ around the pivot element of the partition $P_i$, 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 $PL \neq \emptyset$ 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 $\exists P' \in PL, mindist_{Sphere}(P,S) \geq minmaxdist_{Sphere}(P',S)$
   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.

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.

Fig. 2. Distance Measures: $mindist()$ (Solid Lines) and $minmaxdist()$ (Dotted Lines) for the Case of a Single Query Object $Q$ on the Left and For the Case of a Query-Sphere $S$ on the Right.

The general principle of the whole process is that the nearest neighbor 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 $mindist_{Sphere}()$. $mindist_{Sphere}(P,S)$ is the minimum possible distance from the partition $P$ to the nearest possible point in $S$, i.e., $mindist_{Sphere}(P,S)$ can be reduced to
the \(\text{mindist}(P,Q)\) between a partition \(P\) and a query object \(Q\) by the following means (see also Fig. 2). \(\text{minmaxdist}_S(P,S)\) is the minimal distance from the partition \(P\) to the farthest away point in \(S\), where at least one point in \(P\) has a smaller distance to the farthest away point in \(S\).

The point \(C\) denotes the center of the query-sphere \(S\) and \(r\) its radius. Assuming an Euclidean distance we have:

\[
\text{mindist}_S(P,S) = \begin{cases} 
\text{mindist}(P,C) - r & \text{if } \text{mindist}(P,C) - r \geq 0, \\
0 & \text{otherwise}
\end{cases}
\]

\(\text{mindist}(P,Q)\) and \(\text{minmaxdist}(P,Q)\) for X-trees are defined in [15], for SS-trees in [17] (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 \(\text{Candidate}\). \(\text{Prune}(NN_{ref}, \text{var Candidate}, \text{var PL})\) is the procedure proposed to process the pruning (see Fig. 3). It takes as input two variable parameters \(\text{Candidate}\) and \(PL\). First, all partitions \(P \in PL\) that have \(\text{mindist}_S(P,S)\) larger than the distance of \(NN_{ref}\) to the farthest away object in the hypersphere \(S\) (\(=\text{distancesphere}(NN_{ref}, S)\)) are removed from the list \(PL\). Then, the list of candidate nearest-neighbor objects \(\text{Candidate}\) is pruned, i.e., an object \(\in \text{Candidate}\) is removed from \(\text{Candidate}\), if:

\[
\text{distancesphere}(NN_{ref}, S) \leq \text{distancesphere}(\text{object}, S).
\]

The function \(\text{distancesphere}(\text{object}, S)\) can be defined with the help of \(\text{distance()}\) in the context of a single query object as follows: \(\text{distancesphere}(\text{object}, S) = \text{distance}(\text{object}, C) + r\), where \(C\) denotes once again the center of the hypersphere \(S\) and \(r\) its radius.

**Procedure** \(\text{Prune}(NN_{ref}, \text{var Candidate}, \text{var PL})\)

**Foreach** \(P \in PL\) Do

\(\text{If } \text{mindist}_S(P,S) \geq \text{distancesphere}(NN_{ref}, S) \text{ Then} \)

\(\text{Remove } P \text{ from } PL. \text{ EndIf}\)

**EndForeach**

**Foreach** \(\text{object } \in \text{Candidate}\) Do

\(\text{If } \text{distancesphere}(NN_{ref}, S) \leq \text{distancesphere}(\text{object}, S) \text{ Then} \)

\(\text{Remove } \text{object} \text{ from } \text{Candidate}. \text{ EndIf}\)

**EndForeach**

Fig. 3. *The Pruning Procedure.*
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 $\text{minmaxdist()}$ 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:  

$\text{minmaxdist}_{\text{Sphere}}(P, S) = \text{minmaxdist}(P, C) + r$.

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 $\text{Candidates}$.

5 Experimental Results

We implemented in C++ the nested-loop and optimized algorithms of the NN-search based on the X-tree as reference implementation [15]. The X-tree page size was fixed at 4096 Bytes, which implies an effective capacity of $512/\text{dim}$ data objects per data page, supposing that the value in each dimension is stored in 8 Bytes. All experiments were carried out on a Pentium III, 450 MHz with 128MB main memory and 21GB of storage device. The program simulates the buffer management of a DBMS. We employed the settings of Berchtold et al. [15] 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: $\left\lceil(12.5*\text{dim}*\text{num nodes})/\text{page size}\right\rceil$. 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 should not exceed the maximum size which is set to 256 pages. If the computed sum of cached pages exceeds 256, the number of cached pages for each index is reduced proportionally (for a concrete example see Section 5.2).

The experimental analysis was carried out on two distinct datasets, one synthetic (uniform dataset) (Section 5.1) and one real (Section 5.2). The performance metric is firstly the number of page accesses to the disk that are performed during the multimedia join (i.e., page accesses of both tables) and secondly the running time of the algorithms (CPU-time). We added the CPU-time (running time without waiting for I/O) as a second metric, since the MM-join requires the nodes to be sorted according to the $\text{mindist()}$ and $\text{distance()}$ functions. 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.
5.1 Uniform dataset

**Separate Evaluation of the Optimized Algorithm** We studied first the sensitivity of the optimized algorithms (disk pages accessed and CPU-time) with respect to the variation of the dimension \( (d = 2, d = 4, \ldots, 14) \), the number of data objects in both input tables \((N = 5000, 10000, \ldots, 30000)\) and the number of computed nearest neighbors \(k\). On the left side of Fig. 4 the analysis of the disk pages accessed and on the right sight that of the CPU-time is shown for \(k = 1\). The cases of \(k = 3, 5\) are not shown due to lack of space, but show similar characteristics.

![Graph showing number of pages accessed and CPU-time](image)

**Fig. 4. Optimized Algorithm. Left: Number of Page Accesses. Right: CPU-time. Both based on the Number of Data Objects.**

Left hand of Fig. 4 shows that the number of pages accessed increases with the number of data objects and with an increase in dimension. For each dimension, the gradient of the metric increases for almost all cases with a higher number of data objects. If we see the impact of the dimension, we observe that for dimensions above 8, a large increase in the number of pages accessed is encountered. Searching for 3-NN instead of searching for 1-NN increases the number of pages accessed by an average of 21%, as from 3-NN to 5-NN the increase is on average 19% (both not shown). The shapes of the curves for the optimized and the simple algorithms reveal similar characteristics.

Right hand of Fig. 4 shows that the CPU-time increases, as does the number of pages accessed, with the number of data objects and with an increase in dimension. Contrarily to the page access metric one notices no increase of the gradient when the number of data objects is increased. Again one notices an increase (on average by 13%) of the CPU-time when searching for 3-NN instead of searching for 1-NN, as well as the increase from 3-NN to 5-NN is on average 11%.
Comparison of the Simple and the Optimized Algorithm

Here, the two proposed algorithms are compared for the page access and CPU-time. We computed respective improvement factors of the optimized algorithm over the simple one by means of the quotient: number of pages accessed (CPU-time) by the simple algorithm over that of the optimized algorithm. Fig. 5 (for page accesses) and Fig. 6 (for CPU-time), show the improvement factor with respect to the variation of the dimension, the number of data objects, and the number $k$ of computed nearest neighbors. Once again, 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. 5. Page Access Improvement Factor of the Optimized Algorithm over the Simple Algorithm.

Fig. 5 shows that the highest page access improvement factor (i.e., the highest advantage of the optimized algorithm over the simple algorithm) is achieved independently from the dimension for $N = 5000$. The factor decreases towards 10000 data objects. However, for a number of data objects above 20000 (for $k = 1$) and 25000 (for $k = 3$) the improvement factor remains nearly constant. For example, for $\text{dim} = 14$ 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 algorithm.

Fig. 6 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
Fig. 6. CPU-time Improvement Factor of the Optimized Algorithm over the Simple Algorithm.

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 15000 (for $k = 1$) and 20000 (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 result is based on the fact that the effort of searching the nearest-neighbors (in the inner table) for all objects of a leaf partition (of the outer table) in the simple algorithm is significantly higher than in the optimized algorithm. This result together with the almost constant access page and CPU-time improvement factor for a larger number of data objects argues strongly for the use of the optimized MM-join implementation.

5.2 Real dataset

The real data tested consisted of 2000 icon images in the outer table (from ftp: //ftp.sunsite.org.uk) and 10000 images in the inner table (from http: //www.bdeleeuw.demon.nl). From the icon images, we generated a 4 and then
an 8 dimensional feature vector by extracting respective color histograms. For the inner table, the number of data objects in each leaf partition is therefore 128 for $\text{dim} = 4$ and 64 for $\text{dim} = 8$. The respective index has 3 internal and 79 leaf nodes for $\text{dim} = 4$ and 6 internal and 157 leaf nodes for $\text{dim} = 8$, and the buffer comprises for $\text{dim} = 4$ the root node and one more page of the internal level and the root and three more pages of the internal level for $\text{dim} = 8$.

<table>
<thead>
<tr>
<th>Algo</th>
<th>$k=1$</th>
<th>$k=3$</th>
<th>Algo</th>
<th>$k=1$</th>
<th>$k=3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td>dim=4</td>
<td>1280</td>
<td>dim=4</td>
<td>1536</td>
<td>opti.</td>
</tr>
<tr>
<td>simple</td>
<td>dim=8</td>
<td>1856</td>
<td>dim=8</td>
<td>2432</td>
<td>opti.</td>
</tr>
</tbody>
</table>

The table above shows the number of mean pages accessed of the inner table for a single leaf partition of the outer table using the simple and then the optimized algorithm. The total pages accesses are 16 times the mean values for $\text{dim} = 4$ and 32 times the mean values for $\text{dim} = 8$. It can be seen that (as in the general results above), the influence of the dimension to the number of pages accessed is much higher than the influence of the number of computed nearest neighbors $k$. The improvement factor with the dimension increases by an average of 43.2 for $\text{dim} = 4$ and 52.8 for $\text{dim} = 8$, whereas the improvement factor for $k$ increases by an average of 47.1 for $k = 1$ and 48.9 for $k = 3$.

The implementation keeps most of the internal nodes of the inner table in the buffer. Therefore, the simple algorithm benefits from the buffer by retrieving only few internal pages from disk. The major difference in the number of disk pages accessed of the optimized over the simple algorithm lies in the number of leaf nodes accessed. The optimized algorithm can prune, due to the query-sphere search, fewer pages in the inner table than a single object search by the simple algorithm. However, the pruning is still efficient since less than 50% of the total number of leaf pages are considered (e.g., for $\text{dim} = 4$ and $k = 3$, 44.3% and for $\text{dim} = 8$ and $k = 3$ 28.7%). Clearly, the sum of the pages accessed in order to search for the nearest-neighbors (in the inner table) for query objects (of an outer table leaf node) is considerably higher than the page accesses for the query-sphere search, as one has to repeat the search of candidate leaf nodes for each query object. By implementing a buffer replacement strategy, the buffer changes its content dynamically during the join processing, thus the I/O-performance of the simple algorithm can improve.

The replacement strategy depends to a high degree on the access patterns of the leaf pages for different data objects of an outer table index node. Replacement policies which do not take the access patterns into account do not directly lead to a better buffer hit-rate of the simple algorithm. Suppose for instance that the buffer implements a simple page LRU replacement strategy. If many different path accesses (different for subsequent query objects) are used, the simple algorithm can not profit from significant higher buffer hit-
rates. An examination of access patterns with the aim of finding an adequate buffer implementation strategy is the subject of further research.

The CPU-time improvement factor was about the medium value we measured for the uniform dataset and is significant with respect to the dimension and the number of nearest-neighbors. Concretely, for \( k = 1 \), it was once 329 for \( dim = 4 \) and 184 for \( dim = 8 \), while for \( k = 3 \), it was once 558 for \( dim = 4 \) and 235 for \( dim = 8 \).

6 Conclusion and Future Works

This paper focuses on processing a multimedia join through the method of the nearest neighbor search. We first evaluated 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. This work will identify the necessary operators of multimedia selection and multimedia join, and investigate their use in conjunction with the traditional database operators.

References


