Distance Join Queries of Multiple Inputs in Spatial Databases

Antonio Corral[1]*, Yannis Manolopoulos[2], Yannis Theodoridis[3], and Michael Vassilakopoulos[4]

[1] Department of Languages and Computation, University of Almeria
04120 Almeria, Spain
acorral@ual.es
[2] Department of Informatics, Aristotle University
GR-54006 Thessaloniki, Greece
manolopo@csd.auth.gr
[3] Department of Informatics, University of Pireaus
GR-18534 Pireaus, Greece
ytheo@unipi.gr
[4] Department of Informatics, Technological Educational Institute of Thessaloniki
GR-54101 Thessaloniki, Greece
vasilako@it.teithe.gr

Abstract. Let a tuple of n objects obeying a query graph (QG) be called the n-tuple. The "$D_{distance}$-value" of this n-tuple is the value of a linear function of distances of the n objects that make up this n-tuple, according to the edges of the QG. This paper addresses the problem of finding the K n-tuples between n spatial datasets that have the smallest $D_{distance}$-values, the so-called K-Multi-Way Distance Join Query (K-MWDJQ), where each set is indexed by an R-tree-based structure. This query can be viewed as an extension of K-Closest-Pairs Query (K-CPQ) [4] for n inputs. In addition, a recursive non-incremental branch-and-bound algorithm following a Depth-First search for processing synchronously all inputs without producing any intermediate result is proposed. Enhanced pruning techniques are also applied to the n R-trees nodes in order to reduce the total response time of the query, and a global LRU buffer is used to reduce the number of disk accesses. Finally, an experimental study of the proposed algorithm using real spatial datasets is presented.

Keywords: Spatial databases, Distance join queries, R-trees, Performance study

1 Introduction

The term *spatial database* refers to a database that stores data from phenomena on, above or below the earth's surface, or in general, various kinds of multidimensional data of modern life [11]. In a computer system these data are represented

* Author supported by the Spanish project with ref. TIC 2002-03968.

L. Kalinichenko et al. (Eds.): ADBIS 2003, LNCS 2798, pp. 323–338, 2003.

by points, line segments, polygons, volumes and other kinds of 2D/3D geometric entities and are usually referred to as *spatial objects*.

Some typical spatial queries appearing in spatial databases are the following. (a) *point* (*range*) query; (b) *nearest neighbor* query (K-NNQ); (c) *pairwise* (*multi-way*) *join* query; and (d) *distance join* or *closest pair* query (K-CPQ). Moreover, processing of novel queries, like multi-way spatial join ones, has recently gained attention [9,10,13,14]. In the majority of those papers, a multi-way spatial join query is modeled by a query graph whose nodes represent spatial datasets and edges represent spatial predicates. One way to process this query is as a sequence of pairwise joins when all join spatial datasets are supported by spatial indexes or not (pipelined or build-and-match strategies, respectively). Another possible way, when all join spatial datasets are indexed (using e.g. R-trees), is to combine the filter and refinement steps in a synchronous tree traversal. Moreover, the research interest on distance-based queries involving two datasets (e.g. distance join queries) has increased in the last years, since they are appropriate for data analysis, decision making, etc. Given two datasets S1 and S2, the K-CPQ finds the K closest pairs of objects $<\text{obj}_{1_i}, \text{obj}_{2_j}>$, such that $\text{obj}_{1_i} \in S1$ and $\text{obj}_{2_j} \in S2$, with the K smallest distances between all possible pairs of objects that can be formed by choosing one object of S1 and one object of S2 [4,5,8,16]. If both S1 and S2 are indexed in R-trees, we can use the synchronous tree traversal with Depth-First or Best-First search for the query processing [4,5].

From the above, it is clear that the extension of distance join queries to n inputs with M predicates or constraints (like the multi-way joins) results in a novel query type, the so-called K-Multi-Way Distance Join Query (K-MWDJQ). To our knowledge, this query type has not been studied in the literature so far and this is the aim of this paper.

Definition: Given n non-empty spatial object datasets $S1, S2, \ldots, S_n$ and a query graph QG, the K-Multi-Way Distance Join Query retrieves the K distinct n-tuples of objects of these datasets with the K smallest $\text{D}_{distance}$-values (i.e. the K $\text{D}_{distance}$-smallest n-tuples).

The general environment for this kind of query can be represented by a *network*, where nodes correspond to spatial datasets and edges to binary metric relationships (distances), assigning positive real number to the edges. This framework is similar to the one defined in [10], where the graph is viewed as a constraint network: the nodes correspond to problem variables (datasets) and edges to binary spatial constraints (spatial predicates). Therefore, our network is a weighted directed graph, which directed edges correspond to binary metric relationships (e.g. distances) between pairs of spatial datasets (nodes) with specific weights (positive real numbers) and directions. We also assume that the weighted directed graph cannot be split into non-connected subgraphs. Of course, if we would have such a graph, it could be processed by solving all subgraphs and computing the appropriate combination.

K-Multi-Way Distance Join Queries are very useful in many applications using spatial data for decision making (e.g. in logistics) and other demanding data

handling operations. For example, suppose we are given four spatial datasets consisting of the locations of *factories, warehouses, stores* and *customers*, connecting as Figure 1.a. A K-MWDJQ will find K different quadruplets (factory, warehouse, store, customer) that minimize a $D_{distance}$ function (the K smallest $D_{distance}$-values of the quadruplets are sorted in ascending order). Such a function would be, for example, the sum of distances between a factory and a warehouse, this warehouse and a store and this store and a customer. Such information could then be exploited by a transport agency or a fleet management system for different purposes (e.g. for minimizing transport cost). Moreover, the way to connect the spatial datasets could be more complex than a simple sequence. For example, in Figure 1.b, we can observe the case when the containers of products must be recycled from customers to factories through stores, and new distances must be considered for computing $D_{distance}$. We have considered directed graphs instead of undirected graphs, because the former allow us the configuration of *itineraries* following a specific order; and the users can assign directions and weights to the arcs (directed edges) of the itineraries.

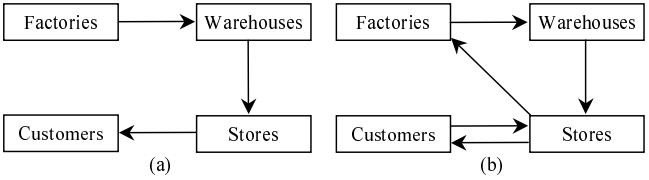

Fig. 1. Directed graphs for factories, warehouses, stores and customers.

The fundamental assumption in this paper is that the n spatial datasets are indexed by R-tree-based structures [7]. R-trees are hierarchical, height balanced multidimensional data structures in secondary storage, and they are used for the dynamic organization of a set of d-dimensional geometric objects represented by their Minimum Bounding d-dimensional hyper-Rectangles (MBRs). These MBRs are characterized by "min" and "max" points of hyper-rectangles with faces parallel to the coordinate axis. Using the MBR instead of the exact geometrical representation of the object, its representational complexity is reduced to two points where the most important features of the object (position and extension) are maintained. R-trees are considered an excellent choice for indexing various kinds of spatial data (points, line segments, polygons, etc.) and have already been adopted in commercial systems, such as Informix [3] and Oracle [12]. Moreover, we must highlight that, in case of non-point objects, an R-tree index can only organize objects' MBRs, together with the pointers to the place where their actual geometry has been stored. Under this framework, K-MWDJQ processing algorithms using R-trees produce only a set of n-tuples of MBRs (hence, candidate objects) in the filter step. For the refinement step, the exact geometry of the spatial objects has to be retrieved and exact distances

have to be computed, according to the $D_{distance}$ function based on the query graph. The algorithm proposed in this paper addresses only the filter step.

The organization of the paper is as follows: In Section 2, we review the literature (K-closest-pairs queries and multi-way join queries) and motivate the research reported here. In Section 3, an expression for a linear distance function based on a given query graph, the definition of K-MWDJQ, an MBR-based distance function and a pruning heuristic are presented. In Section 4, enhanced pruning techniques and a recursive non-incremental branch-and-bound algorithm (called MPSR) for K-MWDJQ are proposed. Section 5 exhibits a detailed experimental study of the algorithm for K-MWDJQ, including the effect of the increase of K and n, in terms of performance (number of disk accesses and response time). Finally, in Section 6, conclusions on the contribution of this paper and related future research plans are summarized.

2 Related Work and Motivation

The K-Multi-Way Distance Join Query can be seen as a "combination" of K-Closest-Pairs query and Multi-way Spatial Join Query; therefore, we review these query types, focusing on the processing techniques of the query algorithms.

***K*-Closest-Pair (or Distance Join) Queries:** [4,5] presented recursive and iterative branch-and-bound algorithms for K-CPQ following a non-incremental approach, which computes the operation when K is known in advance and the K elements, belonging to the result, are reported all together at the end of the algorithm, i.e. the user can not have any result until the algorithm ends. The main advantage of this approach is that the pruning process during the execution of the algorithm is more effective even when K is large enough. On the other hand, the incremental approach for solving the distance join queries [8,16] computes the operation in the sense that the number of desired elements in the result is reported one-by-one in ascending order of distance (pipelined fashion), i.e. the user can have part of the final result before ending the algorithm. The incremental algorithms work in the sense that having obtained K elements in the result, if we want to find the $(K + 1)$-th, it is not necessary to restart the algorithm for $K + 1$, but just to perform an additional step.

Multi-way Spatial Join Queries: [9] proposed a pairwise join method that combines pairwise join algorithms in a processing tree where the leaves are input relations indexed by R-trees and the intermediate nodes are joins operator. [14] proposed a multi-way spatial join by applying systematic search algorithms that exploit R-trees to efficiently guide search, without building temporary indexes or materializing intermediate results. On the other hand, [13] proposed a multi-way R-tree join as a generalization of the original R-tree join [2], taking into account its optimization techniques (the search space restriction and the plane-sweep method). In addition, a recent work [10] reviews pairwise spatial join

algorithms and shows how they can be combined for multiple inputs, explores the applications of synchronous tree traversal for processing synchronously all inputs without producing intermediate results; and then, integrates the two approaches (synchronous tree traversal and pairwise algorithms) in an *engine* using dynamic programming to determine the optimal execution plan.

All the previous efforts have mainly been focused over multi-way spatial join queries, using a sequence of pairwise join algorithms or synchronous tree traversal over R-tree structures on the filter step. Thus, the main objective of this paper is to investigate the behavior of recursive branch-and-bound algorithms in a non-incremental manner for K-MWDJQ as a generalization of K-CPQ between n spatial datasets indexed by R-trees, without producing any intermediate result. To do this, we extend the distance metrics and the pruning heuristic based on the query graph for solving this kind of distance-based query. In addition, we apply techniques for improving the performance with respect to the I/O activity (buffering) and response time (distance-based plane-sweep) in our experiments over real spatial datasets of different nature (points and line segments).

3 *K*-Multi-Way Distance Join Queries Using R-Trees

Let us recall the assumptions we make: (a) n spatial datasets are involved, each supported by an R-tree structure; (b) M ($M \geq n-1$) spatial predicates (metric relationships) between pairs of objects are defined; and (c) a query graph declares the spatial constraints that have to be fulfilled. In the following, we state more formally the details of the problem.

3.1 The Query Graph and the $\mathbf{D}_{distance}$ Function

Query Graph (QG). A query graph QG = (S, E) is a weighted directed graph which consists of a finite non-empty set of nodes $S = \{s_1, s_2, \ldots, s_n\}$ and a finite set of directed edges $E = \{e_{i,j} = (s_i \rightarrow s_j) : s_i, s_j \in S\}$. Each directed edge $e_{i,j}$ connects an ordered pair of nodes $(s_i \rightarrow s_j)$, where s_i and s_j are called start and end nodes of the directed edge, respectively. Associated with each directed edge $e_{i,j}$, there exists a weight $w_{i,j}$, which is a positive real number ($w_{i,j} \in \Re^+$). We assume that the reader is familiar with the related concepts of weighted directed graphs (path, circuit, cycle, self-loop, etc.).

Different configurations of QG depending on the required results by the users are possible. Examples include *sequential* or *"chain"* queries (Figure 1.a), where QG is a *acyclic* weighted directed graph among all datasets, obeying the constraints of a *simple directed path* and it does not contain any directed circuit. Queries with cycles (Figure 1.b) correspond to a QG, with at least one *simple directed circuit* among the nodes existing there (i.e. ordered sequences of nodes with simple directed circuits). Based on this query graph definition, we can define the $\mathrm{D}_{distance}$ function as follows:

$\mathbf{D}_{distance}$ Function. Given n non-empty spatial object datasets $S1, S2, \ldots, Sn$, organized according to a query graph QG and an n-tuple t of spatial objects

of these datasets, $\mathrm{D}_{distance}(t)$ is a linear function of distance values of the pairs of spatial objects belonging in the n-tuple t that result from the directed edges of QG. More formally, we can define $\mathrm{D}_{distance}(t)$ as follows:

$$D_{distance}(t) = \sum_{e_{i,j} \in E_{QG}} w_{i,j}\, distance(obj_i, obj_j)$$

where $t = (obj_1, obj_2, \ldots, obj_n) \in S1 \times S2 \times \ldots \times Sn$, the datasets of the objects of the ordered pair (obj_i, obj_j) are connected in QG by the directed edge $e_{i,j}$, $w_{i,j} \in \Re^+$ is the weight of $e_{i,j}$ and *distance* may represent any Minkowski distance norm (Euclidean, Manhattan, etc.) between pairs of spatial objects.

3.2 Definition of the *K*-Multi-way Distance Join Query

We define the K-Multi-Way Distance Join Query in the spatial database environment as follows:

***K*-Multi-way Distance Join Query.** Let n non-empty spatial object datasets $S1, S2, \ldots, Sn$, organized according to a query graph QG, where a $\mathrm{D}_{distance}$ function is defined. Assume that $obj \subseteq E^d$ (E^d denotes the d-dimensional Euclidean space), for each object obj of any of the above datasets. The result of the K-Multi-Way Distance Join Query, K-MWDJQ($S1, S2, \ldots, Sn, QG, K$), is a set of ordered sequences of K ($1 \leq K \leq | S1 | \cdot | S2 | \cdot \ldots \cdot | Sn |$) different n-tuples of spatial objects of $S1 \times S2 \times \ldots \times Sn$, with the K smallest $\mathrm{D}_{distance}$-values between all possible n-tuples of spatial objects that can be formed by choosing one spatial object from each spatial dataset (i.e. the K $\mathrm{D}_{distance}$-smallest n-tuples):

$$K\text{-MWDJQ}(S1, S2, \ldots, Sn, QG, K) =$$
$$\{(t_1, t_2, \ldots, t_K) : t_i \in S1 \times S2 \times \ldots \times Sn \text{ and } t_i \neq t_j\ \forall i \neq j, 1 \leq i, j \leq K \text{ and}$$
$$\forall t \in S1 \times S2 \times \ldots \times Sn - \{(t_1, t_2, \ldots, t_K)\}, D_{distance}(t) \geq D_{distance}(t_K) \geq$$
$$D_{distance}(t_{(K-1)}) \geq \ldots \geq D_{distance}(t_2) \geq D_{distance}(t_1)\}$$

In other words, the K $\mathrm{D}_{distance}$-smallest n-tuples from the n spatial object datasets obeying the query graph QG are the K n-tuples that have the K smallest $\mathrm{D}_{distance}$-values between all possible n-tuples of spatial objects that can be formed by choosing one spatial object of $S1$, one spatial object of $S2, \ldots$, and one spatial object of Sn. Of course, K must be smaller than or equal to $| S1 | \cdot | S2 | \cdot \ldots \cdot | Sn |$ ($| Si |$ is the cardinality of the dataset Si) i.e. the number of possible n-tuples that can be formed from $S1, S2, \ldots, Sn$. Note that, due to ties of $\mathrm{D}_{distance}$-values, the result of the K-Multi-Way Distance Join Query may not be unique for a specific K and a set of n spatial datasets $S1, S2, \ldots, Sn$. The aim of the presented algorithm is to find one of the possible instances, although it would be straightforward to obtain all of them.

3.3 MBR-Based Distance Function and Pruning Heuristic

The following distance functions between MBRs in E^d have been proposed for the K-CPQ [4,5] as bounds for the non-incremental branch-and-bound algorithms: MINMINDIST (it determines the minimum distance between two MBRs, and it is a generalization of the function that calculates the minimum distance between points and MBRs), MINMAXDIST and MAXMAXDIST.

In the following the definition of the new metric, called $D_{MINMINDIST}$, between n MBRs that depends on the query graph and is based on MINMINDIST distance function between two MBRs in E^d (i.e. $D_{MINMINDIST}$ can be viewed as an instance of $\mathrm{D}_{distance}$ for MINMINDIST function) is presented.

MINMINDIST Function. Let $M(A, B)$ represent an MBR in E^d, where $A = (a_1, a_2, \ldots, a_d)$ and $B = (b_1, b_2, \ldots, b_d)$, such that $a_i \leq b_i$, $1 \leq i \leq d$, be the endpoints of one of its major diagonals. Given two MBRs $M_1(A, B)$ and $M_2(C, D)$ in E^d, MINMINDIST$(M_1(A, B), M_2(C, D))$ is defined as:

$$\text{MINMINDIST}(M_1, M_2) = \sqrt{\sum_{i=1}^{d} {y_i}^2}, \quad y_i = \begin{cases} c_i - b_i, & \text{if } c_i > b_i \\ a_i - d_i, & \text{if } a_i > d_i \\ 0, & \text{otherwise} \end{cases}$$

D*MINMINDIST *Function. Let $M(A, B)$ represent an MBR in E^d, where $A = (a_1, a_2, \ldots, a_d)$ and $B = (b_1, b_2, \ldots, b_d)$, such that $a_i \leq b_i$, $1 \leq i \leq d$, are the endpoints of one of its major diagonals. R_{S_i} is the R-tree associated to the dataset S_i and QG is a query graph obeyed by the n R-trees $R_{S_1}, R_{S_2}, \ldots, R_{S_n}$. Given an n-tuple t of MBRs stored in n R-trees (i.e. t is a tuple of n MBRs from $R_{S_1}, R_{S_2}, \ldots, R_{S_n}$) $D_{MINMINDIST}(t)$ is a linear function of MINMINDIST distance function values of the pairs of t that result from the edges of QG. More formally, we can define $D_{MINMINDIST}$(t) as follows:

$$D_{MINMINDIST}(t) = \sum_{e_{i,j} \in E_{QG}} w_{i,j} MINMINDIST(M_i, M_j)$$

where $t = (M_1, M_2, \ldots, M_n)$ with M_i an MBR of the R-tree R_{S_i} $(1 \leq i \leq n)$, the R-trees of the MBRs of the ordered pair (M_i, M_j) are connected by the directed edge $e_{i,j}$ in QG and $w_{i,j} \in \Re^+$ is the weight of $e_{i,j}$. In other words, $D_{MINMINDIST}$ represents our $\mathrm{D}_{distance}$ function based on MINMINDIST metric for each possible pair of MBRs that belongs in the n-tuple t and satisfies QG.

$D_{MINMINDIST}$ expresses the minimum possible distance of any n-tuple containing n MBRs. For example, in Figure 2, six MBRs (a *6*-tuple of MBRs, t $= (M_{11}, M_{23}, M_{32}, M_{44}, M_{52}, M_{65})$) and their MINMINDIST distances are depicted for a sequential query (QG $= (S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_6)$). $D_{MINMINDIST}$ represents the sum of their MINMINDIST distance values.

We can extend the same properties of MINMINDIST metric between two MBRs to the $D_{MINMINDIST}$ for an n-tuple of MBRs. The most basic properties of $D_{MINMINDIST}$ are the following:

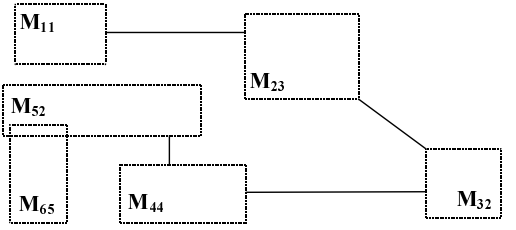

Fig. 2. Example of $D_{MINMINDIST}$ for a sequential query.

(a) Given an n-tuple t of MBRs, the value of $D_{MINMINDIST}$, for a given dimension i, $1 \leq i \leq d$, is always smaller than or equal to the total computation of $D_{MINMINDIST}$: $D_{MINMINDIST}(t, i) \leq D_{MINMINDIST}(t), 1 \leq i \leq d$
(b) Lower-bounding property. For each n-tuple t of spatial objects, enclosed by a n-tuple of MBRs t', it holds that: $D_{MINMINDIST}(t') \leq D_{distance}(t)$
(c) $D_{MINMINDIST}$, like MINMINDIST, is monotonically non-decreasing with the R-tree heights. This means that, for a given n-tuple t of MBRs enclosed by another n-tuple of MBRs t' (where each MBR of t' covers its respective MBR in t), it holds that: $D_{MINMINDIST}(t') \leq D_{MINMINDIST}(t)$

In [5], a pruning heuristic (based on MINMINDIST) was presented in order to minimize the pruning distance (distance value of the K-th closest pair that has been found so far) during the processing of branch-and-bound algorithms for K-CPQ. It declared that, *if* $MINMINDIST(M_1, M_2) > z$*, then the pair of MBRs* (M_1, M_2) *can be discarded*, where z can be obtained from the distance of the K-th closest pair of spatial object found so far. We can extend this pruning heuristic for our new $D_{MINMINDIST}$ function as follows: *if* $D_{\mathrm{MINMINDIST}}(t) > z$*, then the n-tuple of MBRs t can be pruned, where z is the* $D_{distance}$*-value of the K-th n-tuple of spatial objects discovered so far.*

4 An Algorithm for *K*-Multi-way Distance Join Queries

In this section, based on $D_{MINMINDIST}$ function and the pruning heuristic, we are going to propose a recursive non-incremental algorithm for solving the K-Multi-Way Distance Join Query, processing all inputs (n R-trees, indexing n spatial datasets) without producing any intermediate result. This recursive algorithm follows a Depth-First search between n spatial objects indexed in n R-trees. Moreover, enhanced pruning techniques are used in the pruning process for avoiding considering all possible n-tuples of MBRs from n R-tree nodes.

4.1 Enhancing the Pruning Process

An improvement over branch-and-bound algorithms consists in exploiting the spatial structure of the indexes using the plane-sweep technique [15]. We extend the distance-based plane-sweep technique proposed in [5] for K-CPQ in order to

restrict all possible combinations of n-tuples of MBRs from n R-tree nodes in a similar way as in the processing of multi-way join query presented in [10].

Plane-sweep is a common technique for computing intersections [15]. The basic idea is to move a line, called *sweep-line*, perpendicular to one of the dimensions, e.g. X-axis, from left to right. We apply this technique for restricting all possible combinations of n-tuples of MBRs from n R-tree nodes stored in n R-trees. If we do not use this technique, then we must create a list with all possible combinations of n-tuples from n R-tree nodes and process it.

The distance-based plane-sweep technique starts by sorting the entries of the n current R-tree nodes $N_i (1 \leq i \leq n)$ from the n R-trees, based on the coordinates of one of the corners of their MBRs (e.g. lower left corner) in increasing or decreasing order (according to the choice of the sweeping direction and the sweeping dimension, based on the sweeping axis criteria [16]). Suppose that this order is increasing and that Sweeping_Dimension = 0, or X-axis. Then, a set of n pointers (one for each R-tree node) is maintained, initially pointing to the first entry of each X-sorted R-tree node. Among all these entries, let $E_{ix} \in N_i$ $(1 \leq x \leq C_{N_i}$, where C_{N_i} is the capacity of the R-tree node $N_i)$ be the one with the smallest X-value of lower left corner of MBR. We fix the current pivot $\mathbf{P} = E_{ix}$. The MBR of the pivot $\mathbf{P}$ must be paired up with all the MBRs of the entries of the other $n-1$ R-tree nodes $N_j (1 \leq j \leq n$ and $j \neq i)$ from left to right that satisfy MINMINDIST($\mathbf{P}$.MBR, E_{jy}.MBR, Sweeping_Dimension) $\leq z$, where E_{jy} $(1 \leq y \leq C_{N_j})$ is an entry of the R-tree node N_j and z is the $\mathrm{D}_{distance}$-value of the K-th n-tuple of spatial objects found so far. A set of n-tuples of MBRs, ENTRIES = $\{t_1, t_2, \ldots\}$ (empty at the beginning), is obtained. After all these n-tuples of MBRs been processed, the pointer currently pointing E_{ix} is advanced to the next entry of N_i (according to X-order), $\mathbf{P}$ is updated with the entry with the next smallest value of lower left corner of MBR pointed by one of the n pointers, and the process is repeated.

Notice that we apply MINMINDIST(M_{ix}, M_{jy}, Sweeping_Dimension) because the distance over one dimension between a pair of MBRs is always smaller than or equal to their MINMINDIST(M_{ix}, M_{jy}) (a direct extension of the property of MINMINDIST distance function [5]). Moreover, the searching is restricted only to the closest MBRs (belonging to the remainder $n-1$ R-tree nodes) from the pivot $\mathbf{P}$ according to the z value, and no duplicated n-tuples are obtained because the rectangles are always checked over sorted R-tree nodes. The application of this technique can be viewed as a *sliding window* on the sweeping dimension with a width equal to z plus the length of the MBR of the pivot $\mathbf{P}$ on the sweeping dimension, where we only choose all possible n-tuples of MBRs that can be formed using the MBR of the pivot $\mathbf{P}$ and the other MBRs from the remainder $n-1$ R-tree nodes that fall into the current sliding window.

For example, Figure 3 illustrates three sets of MBRs of $(n =)$ 3 R-tree nodes: $\{M_{P1}, M_{P2}, M_{P3}, M_{P4}, M_{P5}, M_{P6}\}$, $\{M_{Q1}, M_{Q2}, M_{Q3}, M_{Q4}, M_{Q5}, M_{Q6}, M_{Q7}\}$, and $\{M_{R1}, M_{R2}, M_{R3}, M_{R4}, M_{R5}, M_{R6}\}$. Without applying this technique we should generate 6*7*6 = 252 triples of MBRs and process them. If we apply the previous method over the X-axis (sweeping dimension), this number of possible triples will be considerably reduced. First of all, we fix the MBR of

the pivot $\mathbf{P} = M_{P1}$ and it must be paired up with $\{M_{Q1}, M_{Q2}, M_{Q3}, M_{Q4}\}$ and $\{M_{R1}, M_{R2}, M_{R3}\}$ because all triples that can be formed from them have MINMINDIST(M_{P1}, M_{Ry}, Sweeping_Dimension) $\leq z$ and the other MBRs can be discarded: $\{M_{Q5}, M_{Q6}, M_{Q7}\}$ and $\{M_{R4}, M_{R5}, M_{R6}\}$. In this case, we will obtain a set of 12 triples of MBRs: $\{(M_{P1}, M_{Q1}, M_{R1}), (M_{P1}, M_{Q1}, M_{R2}), (M_{P1}, M_{Q1}, M_{R3}), (M_{P1}, M_{Q2}, M_{R1}), \ldots, (M_{P1}, M_{Q4}, M_{R3})\}$. When processing is finished with $\mathbf{P} = M_{P1}$, the algorithm must establish the pivot $\mathbf{P} = M_{Q1}$ that is the next smallest value of lower left corner and the process is repeated. At the end, the number of triples of MBRs is 193 = |ENTRIES| (we save 59 *3*-tuples).

After obtaining a reduced set of candidate n-tuples of MBRs from n R-tree nodes (ENTRIES), applying the distance-based plane-sweep technique, we can consider the $D_{MINMINDIST}$ function based on the query graph (QG) over the Sweeping_Dimension as another improvement of the pruning process. Thus, we will only choose for processing those n-tuples of MBRs for which it holds that $D_{MINMINDIST}$(t, Sweeping_Dimension) $\leq z$. This is called $D_{MINMINDIST}$-Sweeping_Dimension filter (i.e. apply the pruning heuristic over the Sweeping_Dimension, preserving the order of entries in this dimension). In the previous example of Figure 3, we can reduce the number of *3*-tuples of MBRs (ENTRIES), depending on the organization of the query graph. If it is a sequential query ($R_P \rightarrow R_Q \rightarrow R_R$) and $\mathbf{P} = M_{P1}$, then the *3*-tuples of MBRs $\{(M_{P1}, M_{Q4}, M_{R1}), (M_{P1}, M_{Q4}, M_{R2})\}$ can be discarded. At the end of the processing of this second filter |ENTRIES| = 164 (we save 29 *3*-tuples). On the other hand, if the query graph is a cycle ($R_P \rightarrow R_Q \rightarrow R_R \rightarrow R_P$) and $\mathbf{P} = M_{P1}$, then the *3*-tuples of MBRs $\{(M_{P1}, M_{Q2}, M_{R3}), (M_{P1}, M_{Q3}, M_{R2}), (M_{P1}, M_{Q3}, M_{R3}), (M_{P1}, M_{Q4}, M_{R1}), (M_{P1}, M_{Q4}, M_{R2}), (M_{P1}, M_{Q4}, M_{R3})\}$ can be discarded, considering only a set of 6 *3*-tuples of MBRs. At the end of the processing of this second filter |ENTRIES| = 107 (we save 86 *3*-tuples). In summary, the pruning process over n R-tree nodes consists of two consecutive filters:

(1) Apply the distance-based plane-sweep technique: select all possible n-tuples of MBRs that can be formed using an MBR as pivot and the others MBRs from the remainder $n-1$ R-tree nodes that fall into a *sliding window* with width equal to z plus the length of the pivot MBR on the Sweeping_Dimension (ENTRIES), since MINMINDIST(M_{ix}, M_{jy}, Sweeping_Dimension) $\leq$ MINMINDIST(M_{ix}, M_{jy}).
(2) Apply the $D_{MINMINDIST}$-Sweeping_Dimension filter: consider from ENTRIES, only those n-tuples of MBRs that satisfy $D_{MINMINDIST}$(t, Sweeping_Dimension) $\leq z$, since $D_{MINMINDIST}$(t, Sweeping_Dimension) $\leq$ $D_{MINMINDIST}$(t).

4.2 A Recursive Branch-and-Bound Algorithm for *K*-MWDJQ

The recursive non-incremental branch-and-bound algorithm follows a Depth-First searching strategy making use of recursion and the previous pruning heuristic based on the $D_{MINMINDIST}$ function. In addition, we employ the distance-based plane-sweep technique and $D_{MINMINDIST}$-Sweeping_Dimension filter for

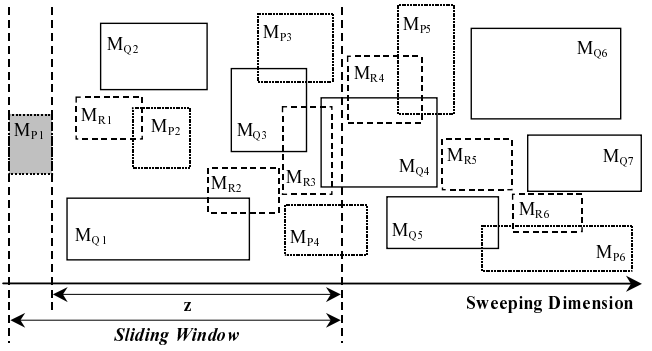

Fig. 3. Using the plane-sweep technique over the MBRs from three R-tree nodes.

obtaining a reduced set of candidate n-tuples of entries from n R-tree nodes (ENTRIES). Then, it iterates over the ENTRIES set and propagates downwards only for the n-tuples of entries with $D_{MINMINDIST}$-value smaller than or equal to z ($D_{distance}$-value of the K-th n-tuple of spatial objects found so far). Also, we need an additional data structure, organized as a maximum binary heap (called K-heap) that holds n-tuples of spatial objects according to their $D_{distance}$-values and stores the K $D_{distance}$-smallest n-tuples and helps us to update z (pruning distance). The **MPSR** algorithm (extension of the PSR algorithm [5] for K-Multi-Way Distance Join Query) for n R-trees storing spatial objects (points or line-segments) on the leaf nodes, with the same height can be described by the following steps.

MPSR1. Start from the roots of the n R-trees and set z to ∞.

MPSR2. If you access to a set of n internal nodes, apply the distance-based plane-sweep technique and the $D_{MINMINDIST}$-Sweeping_Dimension filter in order to obtain the set of n-tuples of candidate MBRs, ENTRIES. Propagate downwards recursively only for those n-tuples of MBRs from ENTRIES that have $D_{MINMINDIST}$-value $\leq z$.

MPSR3. If you access a set of n leaf nodes, apply the distance-based plane-sweep technique and the $D_{MINMINDIST}$-Sweeping_Dimension filter to obtain the set of candidate n-tuples of entries, ENTRIES. Then calculate the $D_{distance}$-value of each n-tuple of spatial objects stored in ENTRIES. If this distance is smaller than or equal to z, remove the n-tuple of spatial objects in the root of K-heap and insert the new one, updating z and K-heap.

In general, the algorithm synchronously processes the n R-tree indexes of all spatial datasets involved in the query (following a Depth-First traversal pattern), using the combinations of R-tree nodes reported by the application of the distance-based plane-sweep technique and $D_{MINMINDIST}$-Sweeping_Dimension filter that satisfy the query graph and pruning the n-tuples with $D_{MINMINDIST}$-value (n internal nodes) or $D_{distance}$-value (n leaf nodes) larger than z.

The advantage of the algorithm that synchronously traverses, with a Depth-First search strategy, all R-trees is that it transforms the problem in smaller local subprolems at each tree level and it does not produce any intermediate result. The downward propagation in step **MPSR2** is done in the order produced by the distance-based plane-sweep technique; and this order is quite good, since it leads to very accurate results quickly. In addition, the algorithm consumes an amount of space that is only a linear function of the heights of the trees and n (number of inputs), and its implementation is relatively easy, because we can use of the recursion. A disadvantage of this algorithm (Depth-First search) is that it tends to consume time to exit, once it deviates from the branches where no optimal solutions of the initial problem are located and the recursion is more expensive with the increase of n.

5 Experimental Results

This section provides the results of an experimentation study aiming at measuring and evaluating the efficiency of the MPSR algorithm. In our experiments, we have used the R*-tree [1] as the underlying disk-resident access method and a global LRU buffer over the n R*-trees with 512 pages. R*-trees nodes, disk pages and buffer pages will have the same size. If the R*-trees have different heights, we will use the fix-at-leaves technique [4]. All experiments were run on an Intel/Linux workstation at 450 MHz with 256 Mbytes RAM and several Gbytes of secondary storage.

In order to evaluate the behavior of the K-MWDJQ algorithm, we have used four real spatial datasets of North America in the same workspace from [6], representing (a) populated places (NApp) consisting of 24,493 2d-points, (b) cultural landmarks (NAcl) consisting of 9,203 2d-points, (c) roads (NArd) consisting of 569,120 line-segments, and (d) railroads (NArr) consisting of 191,637 line-segments. Besides, we have generated (e) a 'pseudo-real' dataset from the 'populated places', simulating archeological places (NAap) of North America and consisting of 61,012 2d-points. With all these datasets, we have designed the following configurations for our experiments, where SQ represents *sequential query* and CY means *query with cycles* in the query graph (the weights of the directed edges: $w_{i,j} = 1.0$).

- $n = 2$: K-MWDJQ(NApp, NAcl, QG, K): QG = (NApp $\rightarrow$ NAcl).
- $n = 3$: K-MWDJQ(NApp, NArd, NAcl, QG, K): QG_{SQ} = (NApp $\rightarrow$ NArd $\rightarrow$ NAcl); QG_{CY} = (NApp $\rightarrow$ NArd $\rightarrow$ NAcl $\rightarrow$ NApp).
- $n = 4$: K-MWDJQ(NApp, NArd, NArr, NAcl, QG, K): QG_{SQ} = (NApp $\rightarrow$ NArd $\rightarrow$ NArr $\rightarrow$ NAcl); QG_{CY} = (NApp $\rightarrow$ NArd $\rightarrow$ NArr $\rightarrow$ NAcl $\rightarrow$ NArr $\rightarrow$ NApp).
- $n = 5$: K-MWDJQ(NApp, NArd, NAap, NArr, NAcl, QG, K): QG_{SQ} = (NApp $\rightarrow$ NArd $\rightarrow$ NAap $\rightarrow$ NArr $\rightarrow$ NAcl); QG_{CY} = (NApp $\rightarrow$ NArd $\rightarrow$ NAap $\rightarrow$ NArr $\rightarrow$ NAcl $\rightarrow$ NArr $\rightarrow$ NApp).

We have measured the performance of our algorithm based on the following two metrics: (1) number of Disk Accesses (DA), which represents the number

Table 1. Comparison of the MPSR algorithm, varying the R*-tree node size.

	1/2 Kbyte		**1 Kbyte**		**2 Kbytes**		**4 Kbytes**		**8 Kbytes**	
n	*DA*	*RT*	*DA*	*RT*	*DA*	*RT*	*DA*	*RT*	*DA*	*RT*
2	2029	0.35	996	0.41	492	0.43	237	0.46	122	0.51
3	32158	16.95	17884	19.39	9436	27.99	4477	34.28	2250	49.75
4	39720	423.38	24551	932.36	13292	2765.1	6384	16603.1	3041	17723.2

of R*-tree nodes fetched from disk, and may not exactly correspond to actual disk I/O, since R*-tree nodes can be found in system buffers, and (2) Response Time (RT), which is reported in seconds and represents the overall CPU time consumed, as well as the total I/O performed by the algorithm (i.e. the total query time). Moreover, due to the different nature of the spatial objects (points and line-segments) involved in the query, we have implemented the minimum distances between points and line-segments.

The first experiment studies the page size for the K-MWDJQ algorithm (MPSR), since the smaller the R*-tree node size is, the smaller the number of n-tuples of R*-tree items have to be considered in the algorithm. We have adopted the following query configurations for MPSR: $n = 2$, 3 and 4; QG_{SQ} (sequential query graphs) and $K = 100$. Table 1 compares the performance measurements for different R*-tree node sizes, where Cmax is the maximum R*-tree node capacity: 1/2 Kbyte (Cmax = 25), 1 Kbyte (Cmax = 50), 2 Kbytes (Cmax = 102), 4 Kbytes (Cmax = 204) and 8 Kbytes (Cmax = 409). We use as minimum R*-tree node capacity Cmin = ⌊Cmax*0.4⌋, according to [1], for obtaining the best query performance.

We can observe from the previous table that the smaller the R*-tree node size is, the faster the MPSR algorithm is, although it obviously needs more disk accesses (using a global LRU buffer, which minimizes the extra I/O cost). As expected, there is a balance between I/O activity (DA) and CPU cost (RT). Since deriving the optimum page size is an unmanageable task due to the number of parameters, we rather focus on the algorithmic issues and not on the previous question. On the other hand, hardware developments are rapid and manufacturers provide disks with larger page sizes year-after-year. Thus, we provide results for the case of page size (R*-tree node size) equal to 1 Kbyte (resulting in the following values of the branching factors for the R*-trees: Cmax = 50 and Cmin = 20, and the reader can extrapolate the method performance for other page sizes. For example, if we compare the size 1 Kbyte and 4 Kbytes for $n = 4$, 1 Kbyte becomes faster than 4 Kbytes by a factor of 17.8, although the increase of disk accesses is only a factor 3.8 using a global LRU buffer with 512 pages.

The second experiment studies the behavior of MPSR algorithm as a function of the number of the spatial datasets involved in the K-MWDJQ. In particular, we use $n = 2, 3, 4$ and 5, $K = 100$, SQ and CY (configurations of the query graph) as the algorithmic parameters for the query. Table 2 shows the performance measurements (DA and RT) of the experiment. We can observe that the increase of the response time is almost exponential with respect to the number of inputs,

Table 2. Comparison of the MPSR algorithm, as a function of the number of inputs.

	$n = 2$		$n = 3$		$n = 4$		$n = 5$	
	DA	*RT*	*DA*	*RT*	*DA*	*RT*	*DA*	*RT*
SQ	996	0.45	17884	19.43	24551	932.36	42124	93942.51
CY			17773	26.47	24088	1169.20	38947	120550.21

whereas the increase of the number of disk accesses is almost linear. This is due to the fact that the number of distance computations depends on the number of considered items in the combination of n R-tree nodes; and it is an exponential function of the R-tree structures (fan-outs: Cmin and Cmax, heights, etc.) and n (Cmin^n). For example, if we compare $n = 4$ and $n = 5$ with QG_{SQ}, the increases of DA and RT are by a factor of 1.7 and 100.7, respectively. Therefore, we can conclude that the response time is more affected than the disk accesses with the increase of the number of inputs in this kind of distance-based query.

The last experiment studies the performance of the MPSR algorithm with respect to the increase of K (number of n-tuples in the result) values, varying from 1 to 100000. Figure 4 illustrates the performance measurements for the following configuration: $n = 4$ (for $n = 3$, the trends were similar), SQ and CY. We can notice from the left chart of the figure that the I/O activity of the algorithm gets higher as K increases and both query graph configurations have similar I/O trends. Moreover, the deterioration is not smooth, although the increase of DA from $K = 1$ to $K = 100000$ is only around a 20%. In the right diagram, we can see that the larger the K values are, the slower the MPSR algorithm becomes, mainly for large K values. For instance, when $K = 1$ and $K = 10000$, the algorithm becomes slower by a factor of 6, and from $K = 1$ to $K = 100000$ the algorithm is 23 times slower for SQ and 27 for CY. From these results, we must highlight the huge response time necessary to report the exact result for large K values and the very small number of required disk accesses.

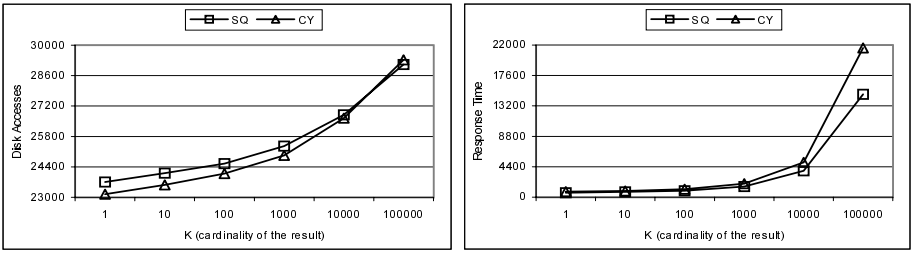

Fig. 4. Performance comparison of the MPSR algorithm for K-MWDJQ varying K for disk accesses (left) and response time (right).

6 Conclusions and Future Work

In this paper, we have presented the problem of finding the K n-tuples between n spatial datasets that have the smallest $D_{distance}$-values, K-Multi-Way Distance Join Query (K-MWDJQ), where each dataset is indexed by an R-tree-based structure. In addition, we have proposed a recursive non-incremental branch-and-bound algorithm following a Depth-First search for processing synchronously all inputs without producing any intermediate result (MPSR). To the best of our knowledge, this is the first algorithm that solves this new and complex distance-based query. The most important conclusions drawn from our experimental study using real spatial datasets are the following: (1) The smaller the R*-tree node size is, the faster the MPSR algorithm becomes. (2) The response time of the query is more affected than the number of disk accesses with the increase of n for a given K, mainly due to the necessary number of distance computations; a similar behavior is obtained from the increase of K for a given n.

Future work may include: (1) The extension of our recursive non-incremental branch-and-bound algorithm to K-Self-MWD Join Query, Semi-MWD Join Query, as well as, the development of algorithms for finding the K $D_{distance}$-largest n-tuples. (2) The use of approximation techniques in our MPSR algorithm that reduce the search space and produce approximate results faster than the precise ones. Further, the study of the trade-off between cost and accuracy of the results obtained by such techniques.

References

1. N. Beckmann, H.P. Kriegel, R. Schneider and B. Seeger; "The R*-tree: and Efficient and Robust Access Method for Points and Rectangles", *Proc. ACM SIGMOD Conf.*, pp. 322–331, 1990.
2. T. Brinkhoff, H.P. Kriegel and B. Seeger; "Efficient Processing of Spatial Joins Using R-trees", *Proc. ACM SIGMOD Conf.*, pp. 237–246, 1993.
3. P. Brown; *Object-Relational Database Development: A Plumber's Guide*, Prentice Hall, 2001.
4. A. Corral, Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos; "Closest Pair Queries in Spatial Databases", *Proc. ACM SIGMOD Conf.*, pp. 189–200, 2000.
5. A. Corral, Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos; "Algorithms for Processing K Closest Pair Queries in Spatial Databases", *Technical Report*, Department of Informatics, Aristotle University of Thessaloniki (Greece), 2001.
6. Digital Chart of the World: Real spatial datasets of the world at 1:1,000,000 scale. 1997 (http://www.maproom.psu.edu/dcw).
7. A. Guttman; "R-trees: A Dynamic Index Structure for Spatial Searching", *Proc. ACM SIGMOD Conf.*, pp. 47–57, 1984.
8. G.R. Hjaltason and H. Samet; "Incremental Distance Join Algorithms for Spatial Databases", *Proc. ACM SIGMOD Conf.*, pp. 237–248, 1998.
9. N. Mamoulis and D. Papadias; "Integration of Spatial Join Algorithms for Processing Multiple Inputs", *Proc. ACM SIGMOD Conf.*, pp. 1–12, 1999.
10. N. Mamoulis and D. Papadias; "Multiway Spatial Joins", *ACM Transactions on Database Systems (TODS)*, Vol. 26, No. 4, pp. 424–475, 2001.

11. Y. Manolopoulos, Y. Theodoridis and V. Tsotras; *Advanced Database Indexing*, Kluwer Academic Publishers, 1999.
12. Oracle Technology Network; *Oracle Spatial*, an Oracle Technical White Paper, 2001 (http://otn.oracle.com/products/oracle9i/pdf/OracleSpatial.pdf).
13. H.H. Park, G.H. Cha and C.W. Chung; "Multi-way Spatial Joins Using R-Trees: Methodology and Performance Evaluation", *Proc. 6th Int. Symp. of Spatial Databases (SSD)*, pp. 229–250, 1999.
14. D. Papadias, N. Mamoulis and Y. Theodoridis; "Processing and Optimization of Multiway Spatial Joins Using R-Trees", *Proc. ACM Symp. on Principles of Database Systems (PODS)*, pp. 44–55, 1999.
15. F.P. Preparata and M.I. Shamos; *Computational Geometry: An Introduction*, Springer-Verlag, 1985.
16. H. Shin, B. Moon and S. Lee; "Adaptive Multi-Stage Distance Join Processing", *Proc. ACM SIGMOD Conf.*, pp. 343–354, 2000.