Multi-way Distance Join Queries in Spatial Databases

Antonio Corral
Dept. Languages & Computation
University of Almeria,
04120 Almeria, Spain
acorral@ual.es

Yannis Manolopoulos
Dept. of Informatics
Aristotle University
GR-54006 Thessaloniki, Greece
manolopo@csd.auth.gr

Yannis Theodoridis
Dept. of Informatics
University of Piraeus
GR-18534 Piraeus, Greece
ytheod@unipi.gr

Michael Vassilakopoulos
Dept. of Informatics
TEI 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) [CMT+01] 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 n R-trees nodes in order to reduce the total response time and the number of distance computations of the query. Due to the exponential nature of the problem, we also propose a time-based approximate version of the recursive algorithm that combines approximation techniques to adjust the quality of the result and the global processing time. Finally, we give a detailed experimental study of the proposed algorithms using real spatial datasets, highlighting their performance and the quality of the approximate results.

Keywords: Spatial databases, Branch-and-bound algorithms, Distance join queries, R-trees, Approximate techniques.

Multi-way Distance Join Queries in Spatial Databases

1 Introduction

The term *spatial database* refers to a database that stores data from phenomena on, above or below the earth's surface [LaT92], or in general, various kinds of multidimensional data of modern life [MTT99]. In a computer system these data are represented by points, line segments, polygons, volumes and other kinds of 2D/3D geometric entities and are usually referred to as *spatial objects*. The field of spatial databases can be defined by its accomplishments and by the needs of existing applications [SCR+99]. Spatial databases are included in specialized applications such as Geographical Information Systems (GIS), Computer Aided Design (CAD), Multimedia Information Systems (MMIS), Data Warehousing (DW), etc. The role of spatial databases is continuously increasing in other many applications during the last years. Transportation planning, resource management and geo-marketing are just some of the emerging applications assisted by spatial databases.

The key characteristic that makes a spatial database a powerful tool is its ability to manipulate spatial data, even rather than simply to store and represent them. The most basic form of such a manipulation is answering queries related to the spatial properties of the data. Some typical spatial queries are the following.

- A *point* (*range*) query involves a single dataset and seeks for spatial objects that fall on a given point (overlap with a given region, usually expressed as a hyper-rectangle).
- A *nearest neighbor* query also involves a single dataset and seeks for the spatial objects residing most closely to a given object. In its simplest form, it discovers one such object (the NN) while in its generalization it discovers K closest objects (K-NN), for a given K.
- A *pairwise* (*multi-way*) *join* query involves two ($n > 2$) datasets and discovers pairs (n-tuples, in general) of spatial objects that satisfy one (M) given predicate(s). The predicate(s) are usually the intersection or overlap between pair(s) of objects.
- As a combination of nearest neighbor and pairwise join queries, a *distance join* or *closest pair* query involves two datasets and discovers the pair of spatial objects (one from each dataset) with the smallest possible distance. Like a join query, all pairs of objects are candidates for the final result; like a nearest neighbor query, proximity metrics are the basis for pruning heuristics and the final ordering. In its incremental form, the query finds the 1st pair, the 2nd pair, and so on, until it is stopped by the user or a trigger (e.g. due to a time restriction). On the other hand, in its non-incremental form (K-Closest-Pairs Query, called K-CPQ), it discovers K pairs of objects, for a beforehand known K.

Processing of multi-way spatial join queries has recently gained attention [MaP01, PMT99, MaP99, PCC99]. 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, when all join spatial datasets are supported by spatial indexes or not (pipelined or build-and-match strategies, respectively), is as a sequence of pairwise joins. 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 similarity join [KoS98] finds all pairs of objects $<obj_{1i}, obj_{2j}>$ such that $obj_{1i} \in S1$ and $obj_{2j} \in S2$, whose distance is smaller than a predefined distance threshold ε. Another example of a distance-based query is the closest pairs query [HjS98, CMT+00, SML00, CMT+01, YaL02], which discovers the K closest pairs of objects in the Cartesian product S1×S2. If both S1 and S2 are indexed by R-trees, we can use the synchronous tree traversal with Depth-First or Best-First search for the query processing [CMT+00, CMT+01].

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 datasets S1, S2, …, Sn 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 $D_{distance}$-values (i.e. the K $D_{distance}$-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 [MaP01], 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, in which the 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 (in the opposite case, the graph could be processed by answering the query for all subgraphs and computing the appropriate combination of results).

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 c*ustomers*, connected as in Figure 1.1.a. A K-MWDJQ will find K different 4-tuples (factory, warehouse, store, customer) that minimize a $D_{distance}$ function (the K smallest $D_{distance}$-values of the 4-tuples 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 based on distances). Moreover, the way to connect the spatial datasets could be more complex than a simple sequence. For example, in Figure 1.1.b, we can observe the case where the containers of products must be recycled from customers to factories through stores, and new distances must be considered for computing $D_{distance}$.

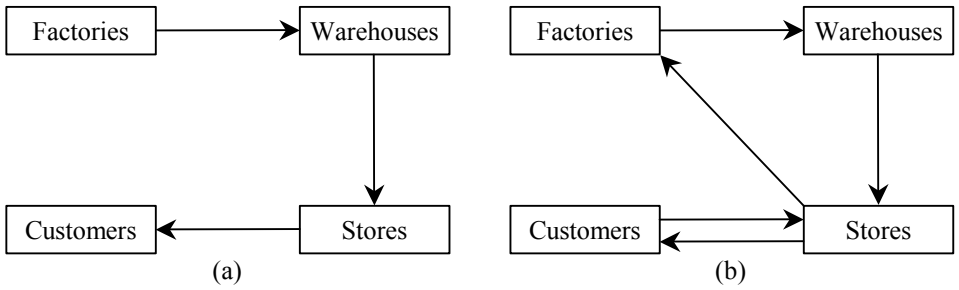

Figure 1.1. Examples of directed graphs for factories, warehouses, stores and customers.

We have considered directed graphs instead of undirected graphs, because the former allow us the expression of *itineraries* (between the spatial datasets) following a specific order. E.g., the users can assign directions and weights to the arcs (directed edges) of the itineraries. Another example of K-MWDJQ application would be the following: a person wants to rent a flat and she/he poses some constraints to the real estate agency. She/he wants to find the itinerary (flat → swimming-pool → gymnasium → flat) with the minimum (Manhattan) distance, since her/his schedule after getting up in the morning at home is to swim, take exercise and return to home. That is, the agency has to report to the client the 3-tuples (flat, swimming-pool, gymnasium) and the rent price of the flat, ordered by the $D_{distance}$ value of itinerary (flat → swimming-pool → gymnasium → flat). Many other similar problems can be found in the real life.

An interesting dimension of the problem is the introduction of weights in distances. For example, covering a distance δ from a factory to a warehouse may be more expensive than covering the same distance δ from a store to a customer (e.g. because of the different means used). Thus, each distance should be multiplied by different weights in order to compute $D_{distance}$. For instance, in our example of (factories, warehouses, stores, customers), if we want to minimize the travel time, we can divide the distance (kilometers) between pairs of points in the itinerary by its known average velocity or speed (kilometers/hours). The quantity 1/velocity is the positive multiplier of each directed edge (the weights of the Query Graph arcs).

Variations of the problem:

The K-MWDJQ can be extended to K-Self-MWDJ, Semi-MWDJ and Multi-way Similarity Join Queries. In particular,

- The K-Self-MWDJQ corresponds to a special case of query graph, where there is a single dataset S, which is connected to itself *n* times (in other words, the query graph contains *n* self-loops). This kind of query can be viewed as a K-Multi-way Distance Join Query in the sense that we can use the same input dataset (S) *n* times. In the previous example, we wish to obtain the 4-tuples which are the K $D_{distance}$-smallest 4-tuples of warehouses among all possible such 4-tuples.
- The Semi-MWDJQ corresponds to the general case of query graph, with the constraint that one of the datasets is chosen as a "reference". For example, if we fix "factories", we wish to obtain for each factory, the smallest $D_{distance}$-value of the triplets (warehouse, shop, customer).
- The Multi-way Similarity Join Query reports all possible *n*-tuples of objects that have a distance smaller than a given distance threshold ε. In this case, the K value is unknown in advance and the result of the query need not be sorted in ascending order of $D_{distance}$-values. For example, we

may wish to obtain those 4-tuples (factory, warehouse, store, customer) that can be visited within a distance smaller than 2 kilometers.

The application of K-Multi-way Distance Join Queries is not restricted to point datasets only. As another example, we can consider three spatial datasets representing populated places, roads and cultural landmarks, respectively. Clearly, the K-MWDJQ can involve non-point datasets as well, assuming that a well-defined distance norm between those spatial objects exists.

The fundamental assumption in this paper is that the n spatial datasets are indexed by R-tree-based structures [Gut84]. R-trees are hierarchical, height balanced multidimensional data structures of 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 [Bro01] and Oracle [Ora01]. 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 only produce 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 algorithms proposed in this paper only address the filter step.

Due to exponential nature of the K-MWDJQ, depending mainly on the cardinalities of the datasets and the number of inputs, the exact processing of the K-MWDJQ algorithms can be prohibitively expensive. However, the performance of these algorithms can be improved if the search space is restricted somehow. Besides, in many situations, for practical purposes, approximate solutions are usually as valuable as exact ones, because such solutions can provide good upper-bounds of the optimum result and can be achieved much faster than the precise ones. In this case, the recursive branch-and-bound algorithm can be used as an approximate algorithm [CCV02] modified with the "time" constraint. The recursive K-MWDJQ algorithm explores the search space in a Depth-First order, finds many solutions quickly (although, it may take very long time to obtain the best solution if it does not traverse the search space in the right direction), improves their qualities continuously and can be stopped at any time during its execution. Here, we also present the recursive K-MWDJQ algorithm as a *time-based approximate algorithm*, and study it in terms of several performance measurements and its performance profile along time. Based on these results, we draw conclusions about the performance of approximate algorithms and examine the influence of the approximate parameters on the trade-off between cost of the algorithm and accuracy of the result.

The organization of the paper is as follows: In Section 2, we review the literature (distance join queries and multi-way join queries) and motivate the research reported. 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 presented. Due to the exponential nature of the K-MWDJQ problem, in Section 5 the recursive algorithm is adapted to a time-based approximate algorithm (called, AMPSR). Section 6 exhibits a detailed experimental study of the algorithms for K-MWDJQ, including the effect of the increase of K

and *n*, the influence of the approximate parameters in terms of performance and quality of the results, and the behavior of the approximate algorithm over the time. In Section 7, 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 that are employed by the query algorithms. Related work includes the following two well-known research areas:

- **K-Closest-Pair (or distance join) Queries**: the problem consists of two given point sets, $P = \{p_1, p_2, \ldots, p_{NP}\}$ and $Q = \{q_1, q_2, \ldots, q_{NQ}\}$ in the multidimensional Euclidean space, stored in a spatial database and a constant K (maximum cardinality in the final result). Then, the result of the K closest pairs query (K-CPQ) is a set of ordered sequences of K ($1 \leq K \leq |P| \cdot |Q|$) different pairs of points of $P \times Q$, with the K smallest distances between all possible pairs of points that can be formed by choosing one point of P and one point of Q. [CMT+00, CMT+01] presented recursive (Depth-First) and iterative (Best-First) branch-and-bound algorithms for K-CPQ following a non-incremental approach, which compute this 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 issue of the non-incremental variant is to separate the treatment of the terminal candidate (the elements of the final result) from the rest of the candidates (intermediate elements). One important advantage of this approach is that the pruning process during the execution of the algorithm is more effective even when K is large enough, making use of various distance functions (MINMINDIST, MINMAXDIST and MAXMAXDIST). Moreover, two well-known optimization techniques are included in the algorithms to reduce the execution time (distance-based plane-sweep technique) and I/O activity (buffering). Recently, a new index structure (the bichromatic Rddn-Tree, which uses information about nearest neighbors to help pruning of the search path more effectively) for improving closest pairs and related distance join queries in spatial databases by implementing several algorithms in a non-incremental manner was proposed [YaL02].
  On the other hand, the incremental (Best-First) approach for solving the distance join queries [HjS98, SML00] computes the desired elements in the result 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, to obtain the (K+1)-th element, it is not necessary to restart the algorithm, but just to perform an additional step. The kernel of the incremental algorithms is a priority queue (distance queue) built on a distance function associated to the specific kind of the distance-based query. The strong point of this approach is that, when K is unknown in advance, the user stops the algorithm when he/she is satisfied by the result. On the other hand, when the number of elements in the result grows, the amount of the required resources to perform the query increases too. Thus, incremental algorithms are competitive when a small quantity of elements of the result is needed. For large K values, the distance queue may not work well as an effective pruning tool, because the cutoff value (pruning distance) stored in the distance queue may remain too high for a long duration. Finally, we must highlight that the work proposed in [HjS98] was enriched in [SML00] by including the plane-sweep technique during the expansion

of node pairs, using the estimation of the k-closest pair distance (eDmax) to suspend unnecessary computations of MBR distances and insertions into the distance queue for the incremental processing.

Besides, several closely related studies for distance-based queries have been recently reported in the literature. For example, the *similarity join* [KoS98] involves two spatial datasets and a given distance threshold δ; the answer is a set of pairs of spatial objects within distance δ from each other. The *iceberg distance join* [SMC+03] also involves two spatial datasets, a given distance threshold δ and a cardinality threshold K; it reports a set of pairs of spatial objects within distance δ from each other, provided that the first object appears at least K times in the join result. Another example is the *K nearest neighbors join* [BoK02], which involves two spatial datasets and a cardinality threshold K; the answer is the smallest subset from the Cartesian Product of the two input datasets that contains for each point of the first dataset at least K points of the second one (i.e. this query combines each of the points of the first dataset with its K nearest neighbors in the second dataset). Moreover, there are nearest neighbor search algorithms based on Voronoi cells [BEK+98] and branch and bound techniques [RKV95], a nearest neighbor search algorithm for ranking requirements [HjS99] and multi-step k-nearest neighbor search algorithms [SeK98].

- **Multi-way Spatial Join Queries**: The problem consists of *n* given datasets D_1, D_2, …, D_n (each indexed by an R-tree) and a query Q. The multi-way join query finds all tuples $\{(r_{1,w}, \ldots, r_{i,x}, \ldots, r_{j,y}, \ldots, r_{n,z})$ such that $\forall i,j : r_{i,x} \in D_i, r_{j,y} \in D_j, r_{i,x}\ Q_{ij}\ r_{j,y}\}$, where Q_{ij} represents the spatial predicate that should hold between D_i and D_j, [MaP01]. In general, multi-way join queries can be considered as a generalization of pairwise spatial joins. [MaP99] 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 join operators. Processing multi-way joins by integration of a sequence of pairwise join algorithms is the standard approach in relational databases, and the order of pairwise joins is determined by the minimization of expected I/O cost (in terms of page accesses). [PMT99] 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, [PCC99] proposed a multi-way R-tree join (M-way join algorithm) as a generalization of the original R-tree join [BKS93], taking into account its optimization techniques (the ordering of the search space restriction and the plane-sweep method). In addition, a recent and extensive work [MaP01] 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; the integration of the two approaches (synchronous tree traversal and pairwise algorithms) in an engine using dynamic programming to determine the optimal execution plan is also presented in [MaP01]. Moreover, two optimizations for synchronous tree traversal algorithm which exploit the spatial structure of the multi-way join problem were proposed in the same paper: (1) Static Variable Ordering (it pre-orders the problem variables by placing the most constrained one first; hence, variables are sorted in decreasing order of their degree), and (2) Plane-Sweep combined with Forward Checking (this is an improved implementation of procedure *find-combinations*, which decomposes a local problem into a series of smaller problems, one for each event of the sweep line). In this paper, experimental results showed that the improvement due to the first improvement is significant when the few first variables are more constrained, whereas this does not apply for complete query graphs. Moreover, the combination of both optimizations showed

significant reduction in both I/O and CPU cost, compared to the version of synchronous tree traversal algorithm that does not use them.

Since a multi-way spatial join is the combination of sequence of pairwise joins and in this sequence a pair does not necessarily correspond to a join between two R-trees [BKS93, HJR97], we also have to consider as related work the cases where only one of the inputs is indexed [LoR94, PRS99, MaP03] or when both inputs are non-indexed [LoR96, PaD96, KoS97].

All the previous efforts have been mainly focused on multi-way spatial join queries, using a sequence of pairwise join algorithms or synchronous tree traversal over R-tree structures on the filter step and on the design of efficient (incremental or non-incremental) K-CPQ algorithms between two R-trees. The main objective of this paper is to investigate the behavior of recursive branch-and-bound algorithms that work 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 (global 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:

1. *n* spatial datasets are involved, each supported by an R-tree structure
2. M ($M \geq n - 1$) spatial predicates (metric relationships) between pairs of objects are defined
3. 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 $D_{distance}$ Function

Query Graph (QG). A query graph QG = (S, E) is a weighted directed graph which consists of a finite nonempty set of nodes $S = \{s_1, s_2, \ldots, s_n\}$ and a finite set of directed edges $E = \{e_{i,j} = (s_i \rightarrow s_j) : s_i, s_j \in S \text{ and } 1 \leq i, j \leq n\}$; 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 ($w_{i,j} \in \Re^+$). A directed edge is called *self-loop*, if both start-end nodes are identical; this is the case of $e_{i,i}$ $(s_i \rightarrow s_i)$.

A *directed path* is a sequence of directed edges connecting pairs of nodes $\{e_{1,2}, e_{2,3}, \ldots, e_{h-1,h}\}$ such that: $e_{i,i+1}$ and $e_{i+1,i+2}$ have only a common node (s_{i+1}) and there does not exist any self-loop. In this case, s_1 and s_h are called the start and end nodes of the directed path. A directed path is called *simple* if no node appears on it more than once (simple directed path). A *directed circuit* is a directed path whose start and end nodes are the same. A directed circuit is called *simple* if no node, apart from start-end node, appears more than once, and the start-end node does not appear elsewhere in the directed circuit.

Different configurations of QG depending on the required results by the users are possible. Examples include *sequential* or "*chain*" queries (Figure 3.1.a), where the QG is an *acyclic* weighted directed graph among all datasets, obeying the constraints of a *simple directed path* that does not

Figure 3.1. Types of query graphs from four spatial datasets.

contain any directed circuit. Queries with *cycles* (Figure 3.1.b) correspond to a QG, with at least one *simple directed circuit* among its nodes (i.e. ordered sequences of nodes with simple directed circuits).

Based on the previous definition of the query graph, we can now define the $D_{distance}$ function as follows:

***$D_{distance}$* Function**. Consider *n* non-empty spatial object datasets S1, S2, …, Sn, organized according to a query graph QG. The $D_{distance}$ is a function from the *n*-tuples of objects from S1 × S2 × … × Sn to $\Re^+$. Let t represent such an *n*-tuple. The $D_{distance}(t)$ is defined as a linear function of distances of the pairs of objects of t that result from the directed edges of QG. More formally, we can define $D_{distance}$(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, …, Sn, organized according to a query graph QG, where a $D_{distance}$ function is defined. Assume that each object of any of the above datasets is a member of the d-dimensional Euclidean space E^d. The result of the K-Multi-way Distance Join Query, **K-MWDJQ(S1, S2, …, Sn, QG, K)**, is a set of ordered sequences of K ($1 \le K \le |S1| \cdot |S2| \cdot \ldots \cdot |Sn|$) different *n*-tuples of spatial objects of S1 × S2 × … × Sn, with the K smallest $D_{distance}$-values between all possible *n*-tuples of spatial objects that can be formed by choosing one spatial object for each spatial dataset (i.e. the K $D_{distance}$-smallest *n*-tuples):

$$\textbf{K-MWDJQ(S1, S2, …, Sn, QG, K)} =$$

$$\{(t_1, t_2, \ldots, t_K): \forall i\ t_i \in (S1 \times S2 \times \ldots \times Sn)^K \text{ and } \forall i \ne j\ t_i \ne t_j,\ 1 \le i, j \le K \textit{ and}$$

$$\forall t \in S1 \times S2 \times \ldots \times Sn - \{(t_1, t_2, \ldots, t_K)\}$$

$$D_{distance}(t) \ge D_{distance}(t_K) \ge D_{distance}(t_{(K-1)}) \ge \ldots \ge D_{distance}(t_2) \ge D_{distance}(t_1)\}$$

In other words, the K $D_{distance}$-smallest *n*-tuples from the *n* spatial object datasets obeying the query graph QG are the K *n*-tuples that have the K smallest $D_{distance}$-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, …,

and one spatial object of Sn. Of course, K must be smaller than or equal to |S1|·|S2|·…·|Sn|, where |Si| is the cardinality of the dataset Si, i.e. the number of possible *n*-tuples that can be formed from S1, S2, …, Sn.

Note that, due to ties of $D_{distance}$-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, …, Sn. The aim of the presented algorithms 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 [CMT+00, CMT+01] 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 (it expresses un upper bound for the distance of the closest pair of spatial objects) and MAXMAXDIST (it obtains the maximum distance between two MBRs).

In the following we present 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 $D_{distance}$ for MINMINDIST function).

MINMINDIST Function. Let M(A, B) represent an MBR in E^d, where A = (a_1, a_2, ..., a_d) and B = (b_1, b_2, ..., b_d), such that $a_i \le b_i$, for $1 \le i \le d$, are 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:

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

***$D_{MINMINDIST}$* Function**. Let M(A, B) represent an MBR in E^d, where A = (a_1, a_2, ..., a_d) and B = (b_1, b_2, ..., b_d), such that $a_i \le b_i$, for $1 \le i \le d$, are the endpoints of one of its major diagonals. R_{Si} is the R-tree associated to the dataset S_i and QG is a query graph obeyed by the *n* R-trees R_{S1}, R_{S2}, …, R_{Sn}. Given an *n*-tuple t of MBRs stored in the *n* R-trees (i.e. t is a tuple of *n* MBRs from R_{S1}, R_{S2}, …, R_{Sn}), $D_{MINMINDIST}(t)$ 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:

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

where t = (M_1, M_2, ..., M_n) with M_i an MBR of the R-tree R_{Si} ($1 \le i \le n$), the R-trees of the MBRs if 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 $D_{distance}$ 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 3.2, seven MBRs (a *7*-tuple of MBRs, t = (M_{11}, M_{23}, M_{32}, M_{41}, M_{54}, M_{62}, M_{75})) and their MINMINDIST distances are depicted for a *sequential query* (QG = ($S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_6 \rightarrow S_7$)). $D_{MINMINDIST}$ represents the sum of their MINMINDIST distance values.

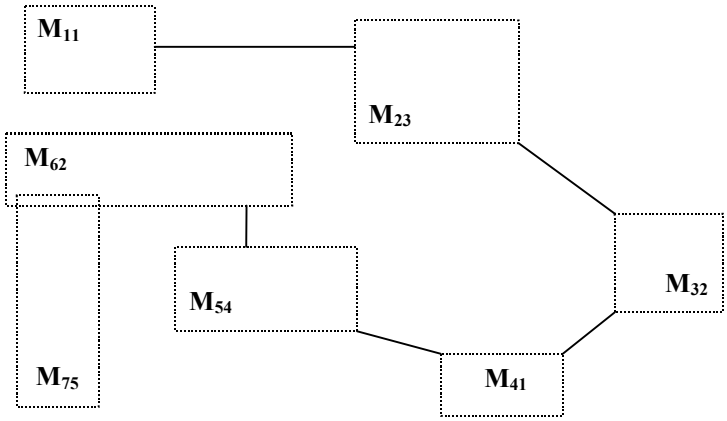

Figure 3.2: Example of $D_{MINMINDIST}$ for a sequential query.

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

- 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 $D_{MINMINDIST}$:

$$D_{MINMINDIST}(t, i) \leq D_{MINMINDIST}(t), \quad 1 \leq i \leq d$$

- Lower-bounding property. For each *n*-tuple t of spatial objects, enclosed by a *n*-tuple of MBRs t', it holds that:

$$D_{MINMINDIST}(t') \leq D_{distance}(t)$$

- $D_{MINMINDIST}$, 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 [CMT+01], a pruning heuristic (based on MINMINDIST) and two updating strategies (based on MINMAXDIST and MAXMAXDIST, respectively) were presented in order to minimize the pruning distance during the processing of branch-and-bound algorithms for K-CPQ. Since using the two updating strategies is optional under given conditions (their computational cost is greater than the gain of updating the pruning distance), we will consider only the pruning heuristic. 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 $D_{MINMINDIST}(t)$ > 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 Exact Algorithm for K-Multi-way Distance Join Queries

At first thought, it would seem easy to find the solution to the MWDJ query by making use of a sequence of 2-way computations, like 2-way joins. For example, in the case of Figure 3.1.a, to compute the K closest pairs between s1 and s2 creating an intermediate result, then to compute the K closest pairs between this intermediate result and s3 result creating another intermediate result and

finally, to compute the K closest pairs between the latter intermediate result and s4). Although, this approach would work in the case of overlap joins, it cannot be used in the general case of distance joins. The tuples that are made up of pairs with minimum distances do not coincide with the tuples that have an overall minimum distance. It is not difficult to find counterexamples where a tuple produced by the above strategy does not belong in the result of the MWDJ query, as well as the opposite. We are going to illustrate this in Figure 4.1 with an example for three datasets (P, Q and R) and a QG configuration corresponding to a sequential query (P → Q → R) for the K-MWDJQ(P, Q, R, 1, QG).

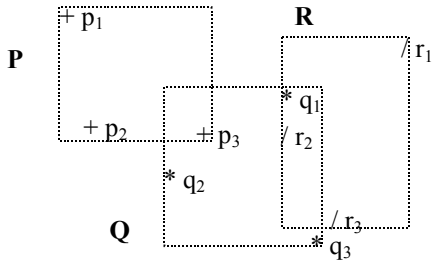

Figure 4.1 Example of three spatial datasets.

It is easy to see that the result for K-CPQ(P, Q, 1) is <p_3, q_2> (q_1 is discarded). Thus the first intermediate result I will contain <p_3, q_2>. Then, we execute the K-CPQ(I, R, 1) and the result is <q_2, r_2>. Thus, the final result consists of the 3-tuple <p_3, q_2, r_2>. This is a wrong result for the K-MWDJQ(P, Q, R, 1, QG). It is not difficult to see that the correct result for this query consists of the 3-tuple <p_3, q_1, r_2>.

Since we assume that all our datasets are indexed by R-trees, we conclude that a strategy that can lead to the correct result of the MWDJ query is to use a synchronous tree traversal, or generalized tree traversal [Map01]. 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 to avoid 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 [PrS85]. We extend the distance-based plane-sweep technique proposed in [CMT+01] 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 [MaP01].

Plane-sweep is a common technique for computing intersections [PrS85]. The basic idea is to move a line, the so-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 the *n* R-trees. If we do not use this technique, then we must create a list with all

possible combinations of *n*-tuples of MBRs or spatial objects from *n* R-tree nodes and process it. In the worst case,

$$\prod_{i=1}^{n} \left| C_{R_{Si}} \right| \text{ is the number of } n\text{-tuples that we must consider,}$$

where $|C_{Rsi}|$ is the R-tree node capacity for the R-tree R_{Si}, indexing the spatial dataset S_i ($1 \le i \le n$).

The *distance-based plane-sweep technique* starts by sorting the entries of the *n* current R-tree nodes N_i ($1 \le i \le n$) from *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 [SML00]). 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 $E_{ix} \in N_i$ ($1 \le x \le C_{Ni}$, where C_{Ni} 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* **P** = E_{ix}. The MBR of the pivot **P** must be paired up with all the MBRs of the entries of the other *n – 1* R-tree nodes N_j ($1 \le j \le n$ and $j \ne i$) from left to right that satisfy *MINMINDIST*(**P**.MBR, E_{jy}.MBR, Sweeping_Dimension) ≤ z, where E_{jy} ($1 \le y \le C_{|Nj|}$) is an entry of the R-tree node N_j and z is the $D_{distance}$-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, …} (empty at the beginning), is obtained. After all these *n*-tuples of MBRs have been processed, the pointer currently pointing E_{jy} is advanced to the next entry of N_j (according to X-order), **P** is updated with the next smallest value of a lower left corner of MBRs 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 [CMT+01]). Moreover, the searching is restricted only to the closest MBRs (belonging to the remainder *n* – 1 R-tree nodes) from the pivot **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 **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 **P** and the others MBRs from the remainder *n* – 1 R-tree nodes that fall into the current *sliding window*.

For example, Figure 4.2 illustrates three sets of MBRs in three (*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}}, respectively. Without applying this technique we should generate 6*7*6 = 252 triplets of MBRs and process them. If we apply the previous method over the X-axis (sweeping dimension), this number of possible triplets will be considerably reduced. First of all, we fix the MBR of the pivot **P** = M_{P1} and it must be paired up with {M_{Q1}, M_{Q2}, M_{Q3} and M_{Q4}} and {M_{R1}, M_{R2} and M_{R3}} because all triplets that can be formed from them have *MINMINDIST*(M_{P1}, M_{Ry}, Sweeping_Dimension) ≤ z and the other MBRs can be discarded: {M_{Q5}, M_{Q6}, and M_{Q7}} and {M_{R4}, M_{R5} and M_{R6}}. In this case, we will obtain a set of 12 triplets of MBRs with the form {(M_{P1}, M_{Q1}, M_{R1}), (M_{P1}, M_{Q1}, M_{R2}), (M_{P1}, M_{Q1}, M_{R3}), (M_{P1}, M_{Q2}, M_{R1}), …, (M_{P1}, M_{Q4}, M_{R3})}. When processing is finished with **P** = M_{P1}, the algorithm must establish the pivot **P** = M_{Q1} that is the next smallest value of lower left corner and the process is repeated. At the end, the number of triplets of MBRs is 193 = |ENTRIES| (we save 59 *3*-tuples).

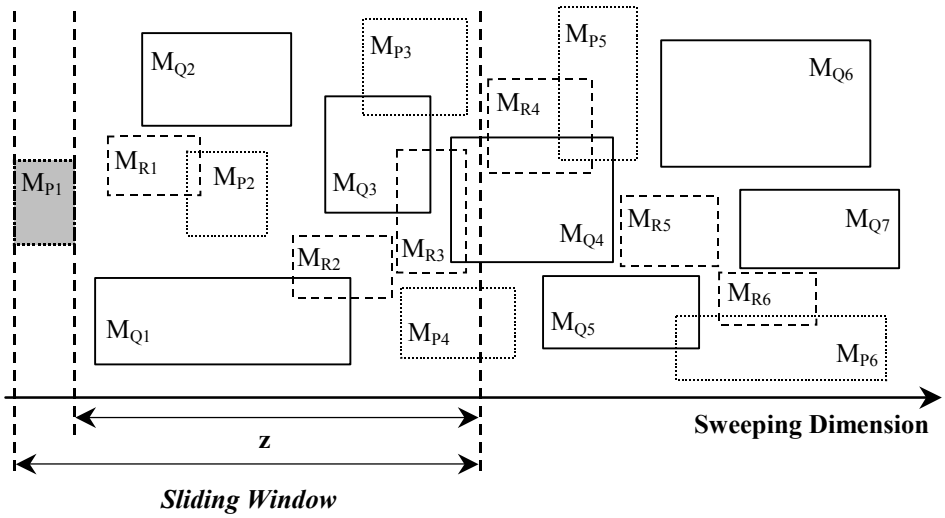

Figure 4.2 Using the plane-sweep technique over the MBRs from three R-tree nodes.

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 choose for processing only those *n*-tuples t of MBRs that satisfy $D_{MINMINDIST}$(t, Sweeping_Dimension) ≤ 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 such dimension). In the previous example of Figure 4.2, 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 **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 **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) ≤ *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) ≤ z, since $D_{MINMINDIST}$(t, i) ≤ $D_{MINMINDIST}$(t), 1≤i≤d. Therefore, ENTRIES = ENTRIES – {t ∈ ENTRIES: $D_{MINMINDIST}$(t, Sweeping_Dimension) > z}.

4.2 A Recursive Branch-and-Bound Algorithm for K-Multi-way Distance Join Query

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

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 smaller than or equal to 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 the *K-heap* and insert the new one, updating z and the *K-heap*.

Figure 4.3. MPSR algorithm.

addition, we employ the distance-based plane-sweep technique and $D_{MINMINDIST}$-Sweeping_Dimension filter for obtaining a reduced set of candidate n-tuples of entries from n R-tree nodes (ENTRIES). Then, it iterates in 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, which stores the K $D_{distance}$-smallest n-tuples and helps us to update z (pruning distance). The MPSR algorithm (extension of the PSR algorithm [CMT+01] for the 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 is illustrated in Figure 4.3.

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 which $D_{MINMINDIST}$-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 into smaller local subproblems 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*; this order is quite good, since it leads to very accurate results quickly (see the comments of Figure 6.4). 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 *recursion*. A disadvantage of this algorithm (Depth-First search) is that it tends to consume time to exit, once it deviates to branches where no optimal solutions of the initial problem are located and the recursion gets more expensive with the increase of n.

5 Approximate Algorithms for K-Multi-way Distance Join Queries

The MPSR algorithm solves the K-Multi-way Distance Join Query accurately, i.e. it focuses on the retrieval of the exact result with no time limitation for the query processing. Depending on the query nature, data properties, cardinalities of the datasets and the number of inputs involved on the query

exhaustive processing of K-MWDJQ can be prohibitively expensive due to the exponential nature of the problem (i.e. in the worst case, the complexity of the algorithm over *n* R-trees is $O(N^n)$).

Since this kind of distance-based query (K-MWDJQ) has an exponential nature, there may not be enough processing time to find the exact result. Therefore, algorithms that obtain one or more approximate solutions can be used. In many situations, for practical purposes, the users are willing to sacrifice the algorithm accurateness for improving performance; approximate solutions, that can be obtained faster than the precise ones, are usually as valuable as them. In order to obtain sufficiently good results quickly and restrict the search space, we can adopt one (or a combination) of the two following modifications of the query algorithm:

- To include into the recursive non-incremental branch-and-bound algorithm (MPSR) a combination of approximation techniques (N-consider and α-allowance [CCV02]) in order to try to control the trade-off between cost and accuracy of the result. This combination consists of two consecutive filters at internal level of the R-trees. In the first filter, we adopt the N-consider approximate technique, producing a set of candidates. Each candidate is examined by the second filter, using the α-allowance technique.

  - **N-consider** (based on the structure of the access method) only takes into account a specified portion, or percentage of the total number of items examined from the combination of *n* internal nodes ($0.0 < N_I \leq 1.0$).

  - **α-allowance** (distance-based approximate technique) is applied over the pruning heuristic and it consists of reducing the pruning distance (z) with an allowance function $\alpha(z)$ that depends on z. That is, an *n*-tuple of MBRs t is discarded if $D_{MINMINDIST}(t) > z - \alpha(z)$. A typical form of $\alpha(z)$ is $\alpha(z) = z*\gamma$ ($0.0 \leq \gamma \leq 1.0$); therefore, the modified pruning heuristic is "if $D_{MINMINDIST}(t) > z*(1 - \gamma)$, then the *n*-tuple of MBRs t can be pruned".

  This combination is appropriate for tuning the trade-off between cost and accuracy of the result and the algorithmic parameters (N_I and γ) can act as adjusters of such a balance (the exact result is obtained when $N_I = 1.0$ and $\gamma = 0.0$). In [CCV02], it was detected that low values of N_I ($0.2 \leq N_I \leq 0.6$) and high values for γ ($0.6 \leq \gamma \leq 1.0$) are good choices for obtaining K approximate closest pairs of points with an acceptable balance between cost and accuracy in high-dimensional data spaces using R-trees.

AMPSR1 Start from the *n* roots of the *n* R-trees and set z to ∞.

AMPSR2 If you access to a set of *n* internal nodes and the consumed time is larger than *total_time* (and *K-heap* is full), then stop. Else, choose only a portion (Total' = N_I*Total) of all possible *n*-tuples of MBRs (Total) stored in the nodes, and apply the distance-based plane-sweep technique *and* the $D_{MINMINDIST}$-Sweeping_Dimension filter over Total' 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 smaller than or equal to z*(1–γ).

AMPSR3 If you access to a set of *n* leaf nodes and the consumed time is larger than *total_time* (and *K-heap* is full), then stop. Else, apply the distance-based plane-sweep technique and the $D_{MINMINDIST}$-Sweeping_Dimension filter in order 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, then remove the *n*-tuple of spatial objects in the root of the *K-heap* and insert the new one, updating z and the *K-heap*.

Figure 5.1. AMPSR algorithm.

- To retrieve the best possible (exact or approximate) result within a given global processing time threshold, *total_time*, (i.e. the algorithm is stopped at the time point *total_time*, reporting the result found so far). Obviously, *total_time* must be large enough for reaching at least one complete approximate solution (i.e. *K-heap* must contain at least K *n*-tuples of spatial objects). We adopt this time-based approximate technique, since the users, in many occasions, prefer fast retrieval of sufficiently good approximate solutions to accuracy of the answer. We obtain the exact result when *total_time* = ∞.

To obtain a time-based approximate K-MWDJQ algorithm, we will apply the previous approximate techniques over the MPSR algorithm based on Depth-First search using recursion in its implementation. Since, this search policy sets higher priority to the subproblems is larger depth, approximate solutions are usually available even if the computation is stopped before the normal termination. As an example, an approximate version of MPSR (AMPSR) based on time for processing K-MWDJQ between *n* R-trees with the same height is illustrated in Figure 5.1 (where N_I, γ and *total_time* are given by the user).

For example, if we want to obtain the exact solution of K-MWDJQ, we can run AMPSR using N_I = 1.0, γ = 0.0 and *total_time* = ∞, i.e. MPSR is a special case of AMPSR for the previous values of the approximation parameters.

6 Experimental Results

This section provides the results of an extensive experimentation study aiming at measuring and evaluating the efficiency of the K-MWDJQ algorithms proposed in Sections 4 and 5, namely MPSR (exact) and AMPSR (approximate). In our experiments, we have used the R*-tree [BKS+90] 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 have the same size. If the R*-trees have different heights, we use the *fix-at-leaves* technique [CMT+00]. All experiments were run on an Intel/Linux workstation at 450 MHz with 256 Mbytes RAM and several Gbytes of secondary storage. The programs were created using the GNU C++ compiler (gcc).

In order to evaluate the K-MWDJQ algorithms, we have used four real spatial datasets of North America in the same workspace from [DCW97], representing (a) populated places consisting of 24,493 2d-points, (b) cultural landmarks consisting of 9,203 2d-points, (c) roads consisting of 569,120 line-segments, and (d) railroads consisting of 191,637 line-segments. Besides, we have generated (e) a 'pseudo-real' dataset from the 'populated places', *simulating* archeological places of North America and consisting of 61,012 2d-points. With these datasets, we have designed the following configurations for our experiments, where SQ represents a sequential query and CY represents a query with cycles in the query graph and the weights ($w_{i,j}$) of the directed edges in such query graphs were equal to 1.0 (i.e. only the distances were considered).

- n = 2: *K-MWDJQ(NApp, NAcl, QG, K)*: QG = (NApp → NAcl).
- n = 3: *K-MWDJQ(NApp, NArd, NAcl, QG, K)*: QG_{SQ} = (NApp → NArd → NAcl) and QG_{CY} = (NApp → NArd → NAcl → NApp).
- n = 4: *K-MWDJQ(NApp, NArd, NArr, NAcl, QG, K)*: QG_{SQ} = (NApp → NArd → NArr → NAcl) and QG_{CY} = (NApp → NArd → NArr → NAcl → NArr → NApp).
- n = 5: *K-MWDJQ(NApp, NArd, NAap, NArr, NAcl, QG, K)*: QG_{SQ} = (NApp → NArd → NAap → NArr → NAcl) and QG_{CY} = (NApp → NArd → NAap → NArr → NAcl → NArr → NApp).

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 segments as follows [Oro98]:

- *Point-point distance*. Given two points, p and q, the minimum Euclidean distance between them is defined as follows: PointsDistance(p, q) = $(\sum_i |p_i - q_i|^2)^{1/2}$ = MINMINDIST(p, q) [CMT+01], since, MINMINDIST is a generalization of the minimum distance between points and MBRs.

$$PointsDistance(p,q) = \sqrt{\sum_{i=1}^{d} |p_i - q_i|^2} = MINMINDIST(p,q)$$

- *Point-segment distance*. Consider a point p and a line-segment S = (s_1, s_2) characterized by two endpoints s_1 and s_2. Let ⊥(p, S) be the perpendicular line to S that passes through p. The computation of the minimum distance between p and S (MinimumDistance(p, S)) is defined as follows: If ⊥(p, S) intersects S at a point q, then MinimumDistance(p, S) = MINMINDIST(p, q), otherwise MinimumDistance(p, S) = **min**{MINMINDIST(p, s_1), MINMINDIST(p, s_2)}.
- *Segment-segment distance*. Given two line-segments, S = (s_1, s_2) and T = (t_1, t_2), we consider the previous case (point-segment distance) for <s_1, T>, <s_2, T>, <t_1, S> and <t_2, S>, taking the minimum of these distances. If there is no intersection between the perpendicular line of the segment passing through the point with the segment, then MinimumDistance(S, T) = **min**{MINMINDIST(s_1, t_1), MINMINDIST(s_1, t_2), MINMINDIST(s_2, t_1), MINMINDIST(s_2, t_2)}. That is, given the endpoints s_i, and t_i of two different line-segments S and T, MinimumDistance(S, T) is the minimum distance among all possible pairs of endpoints. Of course, if S and T intersect, then MinimumDistance(S, T) = 0.0.

We have measured the performance of our algorithms based on the following two metrics: (1) *number of Disk Accesses* (DA), which represents the number 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 or total elapsed time).

	½ Kbyte		1 Kbyte		2 Kbytes		4 Kbytes		8 Kbytes	
	DA	*RT*	*DA*	*RT*	*DA*	*RT*	*DA*	*RT*	*DA*	*RT*
***n* = 2**	2029	0.35	996	0.41	492	0.43	237	0.46	122	0.51
***n* = 3**	32158	16.95	17884	19.39	9436	27.99	4477	34.28	2250	49.75
***n* = 4**	39720	423.38	24551	932.36	13292	2765.07	6384	16603.11	3041	17723.21

Table 6.1. Comparison of the MPSR algorithm for K-MWDJQ varying the R*-tree node size.

Apart of the previous performance metrics, we have taken into account two additional quality measurements of the approximate results: (1) *Average Relative Distance Error* (ARDE); to obtain ARDE, we calculate the exact result for the K-MWDJQ off-line, then apply the approximate algorithm and calculate the average relative distance error of all the K items of the result. (2) *Quality of the Approximate Result* (QAR); QAR calculates the percentage of the K items of the approximate solution that also appear in the exact result (i.e. values of QAR close to 1.0 indicate a good quality of the approximate solution, since QAR = 1.0 is the value for the exact result):

$$QAR = \frac{1}{K}\sum_{i=1}^{K}\left\{\begin{array}{ll} 1, & \textit{if the approximate item "i" appears in the exact result} \\ 0, & \textit{otherwise} \end{array}\right\}$$

The first experiment studies the best page size for the K-MWDJQ algorithms, 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 algorithms. We have adopted the following query configurations for MPSR: n = 2, 3 and 4; QG_{SQ} (sequential query graphs) and K = 100. Table 6.1 compares the performance measurements for different R*-tree node sizes, where M is the maximum R*-tree node capacity: ½ Kbyte (M = 25), 1 Kbyte (M = 50), 2 Kbytes (M = 102), 4 Kbytes (M = 204) and 8 Kbytes (M = 409). m = $\lfloor$M*0.4$\rfloor$ was used as minimum R*-tree node capacity, according to [BKS+90], 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 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 involved, we rather focus on the algorithmic issues and not on this 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 M = 50 and m = 20 branching factors for the R*-trees). The reader can extrapolate the method performance for other page sizes. For example, if we compare the page sizes of 1 Kbyte and 4 Kbytes for n = 4, the algorithm for the 1 Kbyte size becomes faster than the 4 Kbytes size by a factor of 17.8, although the increase of disk accesses is only by a factor of 3.8, using a global LRU buffer with 512 pages.

Moreover, an important observation about the results appearing Table 6.1 concerns the processing cost to obtain the *exact* result (e.g. for n = 4 and a page size of 4 Kbytes, the branching factors are M = 204 and m = 82, the number of disk accesses is 6384 and the response time is 16603.11 seconds). The CPU cost of K-MWDJQ is, in general, exponential to the number of datasets *(n)* involved in the query, mainly due to the number of distance computations (for n = 4 and page size of 4 Kbytes, in the worst case, the number of distance computations can vary from 204^4 = 1731891456 to 82^4 = 45212176, when 4 R-tree nodes are considered). Since this problem is computationally very demanding (in terms of distance computations, which implies a very high cost in execution time), our main target is to

	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	1169.25	38947	120550.21

Table 6.2. Comparison of the MPSR algorithm, as a function of the number of inputs.

study ways for its most efficient computation. This motivates the need for efficient discovery of approximate solutions through approximation techniques that restrict the search space. Note that, since the performance achieved, although improved, may still not be acceptable for on-line applications, the (exact, or approximate) algorithms for answering the K-MWDJQ are mainly useful in offline applications (even in these applications, performance is a big issue).

The second experiment studies the behavior of the MPSR algorithm (exact results) 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 6.2 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, 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, which is an exponential function of the R-tree structures (fan-outs, heights, etc.) and n. For example, if we compare n = 4 and n = 5 with QG_{SQ}, the increases of DA and RT are by factors of 1.7 and 100.7, respectively. Therefore, we can conclude that the response time is more affected than the number of disk accesses with the increase of the number of inputs in this kind of distance-based query; hence, approximate approaches that aim at obtaining sufficiently good results quickly are worth considering.

The third 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 6.1 illustrates the performance measurements for the following configuration: n = 4 (for n = 3, the tendencies 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 notice that the larger the K values are, the slower the MPSR algorithm becomes, mainly for large K values. For example, 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 (K = 100000) and the very small number of required disk accesses. This suggests that the MPSR algorithm reaches many intermediate solutions and it takes a long time to obtain the exact solution, if it does not traverse the search path in the right direction; we have to avoid this unnecessary work by using a time-based approximate algorithm, obtaining sufficiently good results quickly.

In the forth experiment, we are going to study the behavior of AMPSR algorithm (time-based approximate version of MPSR) with respect to the performance measurements and the quality of the results as a function of the approximate parameters N_I and γ (*total_time* = ∞, i.e. no time control). The main objective of this experiment is to determine the best values of N_I and γ for AMPSR. Figure 6.2 shows the performance measurements (DA and RT) for the following experiment configuration: n = 4,

SQ (for CY, the trends were similar), K = 100, *total_time* = ∞, 0.2 ≤ N_I ≤ 0.6, and 0.6 ≤ γ ≤ 1.0. It is clear that the smaller the value of N_I is, the faster and the smaller the I/O activity of the AMPSR algorithm is; although for N_I ≥ 0.5 the AMPSR algorithm does not obtain any improvement (i.e. N_I = 0.5 is a performance threshold). On the other hand, the saving in DA is almost negligible with the increase of γ, but the algorithm becomes faster (the RT is reduced considerably). For example, using N_I = 0.5 and γ = 1.0, we save only a 4.54% of disk accesses (23436 / 24551), but the algorithm becomes 2.6 times faster in seconds (361.37 / 932.36) than the exact one. In addition, this result is quite interesting, because the approximate result has an excellent quality (ARDE = 0.0095 and QAR = 0.98, i.e. very close to the exact solution) and much time (570.99 seconds) was saved. Also, we must highlight that for N_I = 0.5, the algorithm works with half of all possible combinations of MBR *n*-tuples, when *n* internal nodes are visited; however these combinations are chosen by the distance-based plane-sweep technique that produces a quite good ordering of all possible combinations.

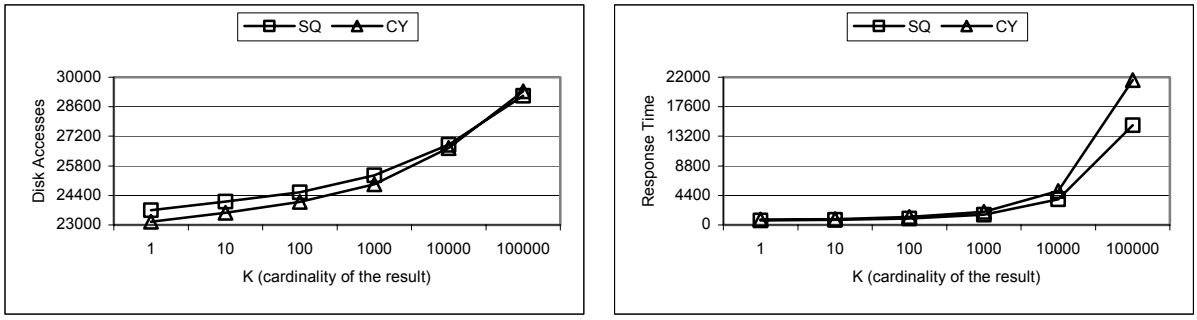

Figure 6.1. Performance comparison of the MPSR algorithm for K-MWDJQ varying K for disk accesses (left) and response time (right).

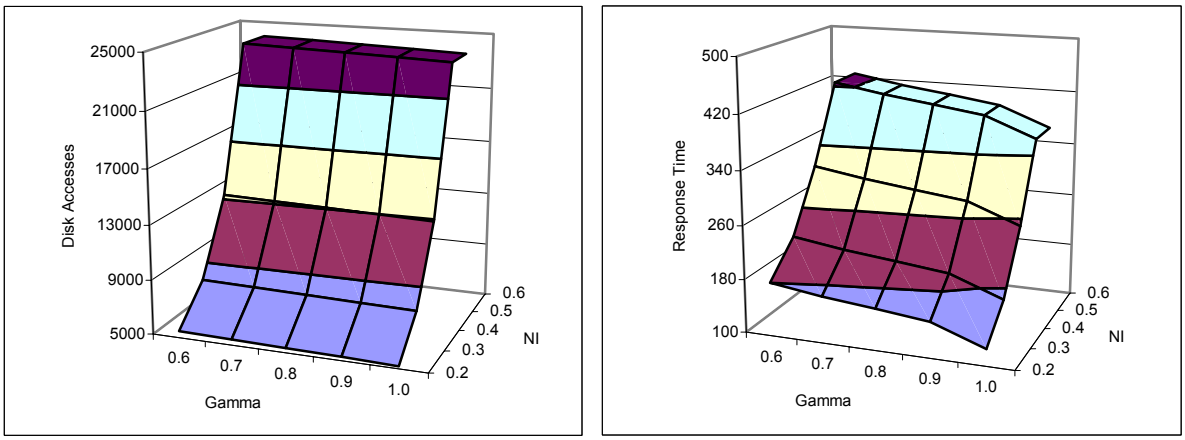

Figure 6.2. Number of disk accesses (left) and response time (right) of the AMPSR algorithm for K-MWDJQ, as a function of N_I in the range [0.2, 0.6] and γ in the range [0.6, 1.0].

We can observe that N_I = 0.5 and γ = 1.0 are the most promising approximate parameter values in order to obtain a good performance. Which is the *quality* of the reported result with respect to the exact one? Figure 6.3 shows the quality measurements (ARDE and QAR) of the same experiment configuration as the one of Figure 6.2. Note that in the left chart of Figure 6.3 (ARDE), we have changed the order of N_I values (decreasing) to observe clearly the behavior (evolution) of this parameter, since the bars for ARDE when N_I = 0.2 and 0.3 can hide the bars for the other N_I values (0.4, 0.5 and 0.6). Moreover, values of ARDE close to 0.0 indicate good quality if the approximate solutions, because ARDE = 0.0 represents that the approximate result coincides with the exact one. For N_I = 0.5 and γ = 1.0, the quality is very high, since ARDE = 0.009488648 and QAR = 0.98; very

close to the values for the exact result (ARDE = 0 and QAR = 1.0). In general, the quality metrics are more affected by N_I parameter than γ (the increase of γ values does not affect significantly the quality of the approximate result). The reduction of N_I values saves DA and RT, but it loses quality of the approximate solutions (N_I = 0.5 is a quality threshold); and the increase of γ saves DA and RT, but it almost does not affect quality. Thus, N_I = 0.5 and γ = 1.0 is an interesting combination of values of the approximate parameters to obtain a good trade-off between cost of the algorithm and accuracy of the approximate solution.

Using this combination of the approximate parameter values (N_I = 0.5 and γ = 1.0), we have executed experiments for n = 5, SQ (for CY, the results were very similar), K = 100 and *total_time* = ∞. The AMPSR algorithm saved a 9.3% in DA (38210 / 42124) with respect to MPSR, but it was around 3 times faster than the exact one (28669.28 / 93942.51), and the quality of the approximate result was ARDE = 0.0 and QAR = 1.0 (i.e. the approximate solution was identical to the exact one); for this particular case (n = 5 and real data), all the n-tuples that make up the exact result exist in the approximate result (note that it is possible for the approximate algorithm, AMPSR, to reach the exact result). In this specific experimental setting, the ordering produced by the distance-based plane-sweep technique was excellent. For other experimental settings, the approximate algorithm did not discover the exact result, although we noticed that the accuracy of the approximate result was always very high. This is the case of the results presented in the next experiment.

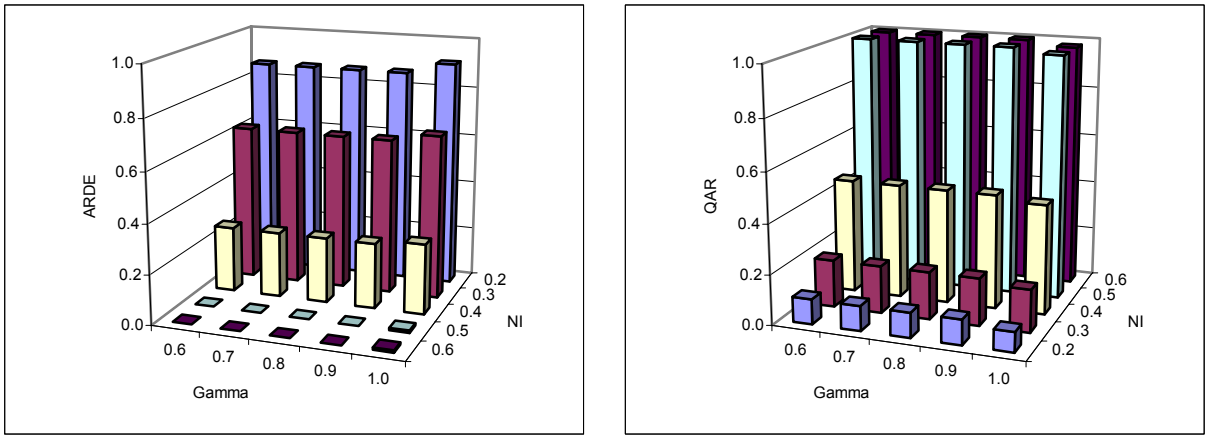

Figure 6.3. ARDE (left) and QAR (right) of the AMPSR algorithm for K-MWDJQ, as a function of N_I in the range [0.2, 0.6] and γ in the range [0.6, 1.0].

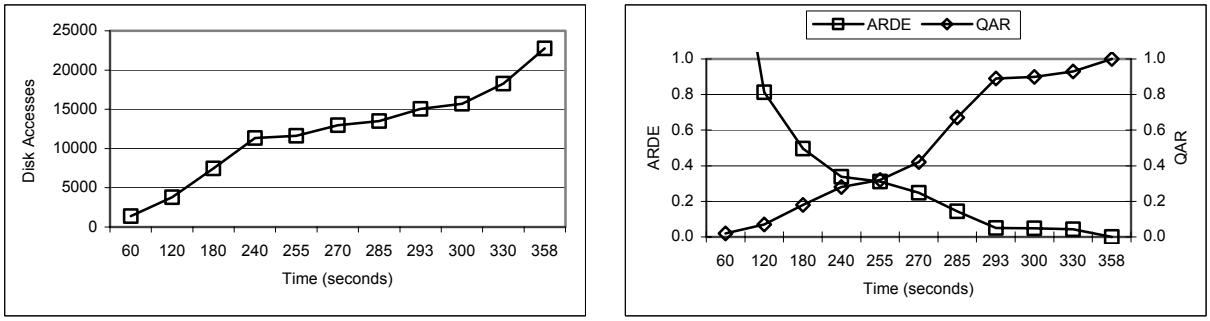

Figure 6.4. Comparison of the AMPSR algorithm for K-MWDJQ varying the processing *total_time* for disk accesses (left) and quality metrics: ARDE and QAR (right).

In the fifth experiment, we measure the quality of the approximate solutions retrieved over time. Since the AMPSR algorithm starts with a solution which has very low quality, as the processing time passes the algorithm improves the solution until the accurate one is reached. Figure 6.4 shows the

performance metric (DA) and the quality measurements (ARDE and QAR) of the experimental configuration: $n = 4$ (similar tendencies were obtained for $n = 5$), SQ (for CY, the trends were similar), $K = 100$, $N_I = 0.5$, $\gamma = 1.0$, varying the *total_time* (*total_time* ≤ 358.01 seconds, which was the total time for AMPSR when *total_time* = ∞). As expected, if we give more processing time to the AMPSR algorithm, then the number of disk accesses is also increased. On the other hand, the behavior of the quality measurements is very interesting: ARDE and QAR follow opposite trends; the smaller the ARDE (larger the QAR) values are, the more accurate of the approximate solutions are; it is easy to see that QAR(t) ≈ 1 – ARDE(t), where ≈ stands for 'follows a similar trend to'. Moreover, at the beginning of the execution of the algorithm, the quality of the approximate solution is low, but close to the end it is very high, and after a time point the approximate solution found so far is so good that it gets difficult for the algorithm to obtain a better one.

If we observe both charts of Figure 6.4, we can see that a relation between performance (DA) and quality (QAR) measurements over time. The overall performance of a time-based approximate algorithm (AMPSR) can be related to the quality of the result found and the amount of computation time consumed to obtain such a result. DA and QAR follow a similar trend: the larger the number of disk accesses is, the larger the QAR values are; the relation between them is DA(t) ≈ QAR(t).

7 Conclusions and Ideas for Future Extensions

In this paper, we have examined the problem of finding the K *n*-tuples between *n* spatial datasets that have the smallest $D_{distance}$-values, the 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). Due to the exponential nature of this kind of distance-based query, we have also proposed a time-based approximate version of MPSR that combines approximation techniques to adjust the quality of the approximate result and the global processing time (AMPSR). To the best of our knowledge, these are the first algorithms that solve 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 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 with the increase of K for a given *n*.

(2) For AMPSR, we have found values of the approximate parameters that are useful for the users who seek a good balance between cost and quality of the approximate result, when the total time of processing is not limited ($N_I = 0.5$ and $\gamma = 1.0$).

(3) As expected, the quality of the best approximate solution found so far using AMPSR is successively improved as long as more computation time is given and follows a similar trend to the behavior of the performance (number of disk accesses) of the approximate algorithm.

Future work may include:

- Extending our recursive non-incremental branch-and-bound algorithms to K-Self-MWD Join Query, Semi-MWD Join Query, as well as, finding the K $D_{distance}$-largest *n*-tuples. We can also easily adapt our algorithms to support the Multi-way Similarity Join Query and obtain all the *n*-tuples of objects that do not exceed a given distance threshold ε.

- Using in our recursive algorithms for K-MWDJQ other approximation techniques that combine local and evolutionary search with underlying indexes to prune the search space [PaA02], or randomized search methods like iterative improvement, random sampling, simulated annealing, etc. [MaP01].
- Supporting an arbitrary graph (or tree), not just sequences or sequences with cycles, considering other alternatives for $D_{distance}$ and not just a linear function with positive multipliers.
- Extending our algorithms to support spatio-temporal databases (i.e., one or some spatial datasets, consisting of moving objects).

REFERENCES

[BEK+98] S. Berchtold, B. Ertl, D. Keim, H.P. Kriegel and T. Seidl: "Fast Nearest Neighbor Search in High-Dimensional Spaces", *Proceedings of 14th International Conference on Data Engineering* (ICDE'98), pp.209-218, 1998.

[BKS+90] N. Beckmann, H.P. Kriegel, R. Schneider and B. Seeger: "The R*-tree: an Efficient and Robust Access Method for Points and Rectangles", *Proceedings of ACM SIGMOD Conference*. pp.322-331, 1990.

[BKS93] T. Brinkhoff, H.P. Kriegel and B. Seeger: "Efficient Processing of Spatial Joins Using R-trees", *Proceedings of ACM SIGMOD Conference*, pp.237-246, 1993.

[BoK02] C. Böhm and F. Krebs: "High Performance Data Mining Using the Nearest Neighbor Join", *Proceedings of 2nd International Conference on Data Mining* (ICDM'02), pp.43-50, 2002.

[Bro01] P. Brown: *Object-Relational Database Development: a Plumber's Guide*, Prentice Hall, 2001.

[CCV02] A. Corral, J. Cañadas and M. Vassilakopoulos: "Approximate Algorithms for Distance-Based Queries in High-Dimensional Data Spaces using R-trees", *Proceedings of 6th Conference on Advances in Databases and Information Systems* (ADBIS'02), pp.163-176, 2002.

[CMT+00] A. Corral, Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos: "Closest Pair Queries in Spatial Databases", *Proceedings of ACM SIGMOD Conference,* pp.189-200, 2000.

[CMT+01] A. Corral, Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos: "Algorithms for Processing K Closest Pair Queries in Spatial Databases", *Data & Knowledge Engineering*, to appear.

[DCW97] Digital Chart of the World: Real spatial datasets of the world at 1:1,000,000 scale. 1997. Downloadable from: http://www.maproom.psu.edu/dcw.

[GaG98] V. Gaede and O. Günther: "Multidimensional Access Methods", *ACM Computing Surveys*, Vol.30, No.2, pp.170-231, 1998.

[Gut84] A. Guttman: "R-trees: a Dynamic Index Structure for Spatial Searching", *Proceedings of ACM SIGMOD Conference*, pp.47-57, 1984.

[HJR97] Y.W. Huang, N. Jing and E.A. Rundensteiner: "Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations", *Proceedings of 23rd VLDB Conference*, pp.396-405, 1997.

[HjS98] G.R. Hjaltason and H. Samet: "Incremental Distance Join Algorithms for Spatial Databases", *Proceedings of ACM SIGMOD Conference*, pp.237-248, 1998.

[HjS99] G.R. Hjaltason and H. Samet: "Distance Browsing in Spatial Databases", *ACM Transactions on Database Systems*, Vol.24, No.2, pp.265-318, 1999.

[KoS97] N. Koudas and K.C. Sevcik: "Size Separation Spatial Join", *Proceedings of ACM SIGMOD Conference*, pp.324-335, 1997.

[KoS98] N. Koudas and K.C. Sevcik: "High Dimensional Similarity Joins: Algorithms and Performance Evaluation", *Proceedings of 14th International Conference on Data Engineering* (ICDE'98), pp.466-475, 1998.

[LaT92] R. Laurini and C Thomson: *Fundamentals of Spatial Information System*, Academic Press, 1992.

[LoR94] M.L. Lo and C.V. Ravishankar: "Spatial Joins Using Seeded Trees", *Proceedings of ACM SIGMOD Conference*, pp.209-220, 1994.

[LoR96] M.L. Lo and C.V. Ravishankar: "Spatial Hash-Joins", *Proceedings of ACM SIGMOD Conference*, pp.247-258, 1996.

[MaP99] N. Mamoulis and D. Papadias: "Integration of Spatial Join Algorithms for Processing Multiple Inputs", *Proceedings of ACM SIGMOD Conference*. pp.1-12, 1999.

[MaP01] N. Mamoulis and D. Papadias: "Multiway Spatial Joins", *ACM Transactions on Database Systems*, Vol.26, No.4, pp.424-475, 2001.

[MaP03] N. Mamoulis and D. Papadias: "Slot Index Spatial Join", *IEEE Transactions on Knowledge and Data Engineering*, Vol.15, No.1, pp.211-231, 2003.

[MTT99] Y. Manolopoulos, Y. Theodoridis and V. Tsotras: *Advanced Database Indexing*, Kluwer Academic Publishers, 1999.

[Ora01] Oracle Technology Network: *"Oracle Spatial"*, an Oracle Technical White Paper, 2001. Downloadable from: http://otn.oracle.com/products/oracle9i/pdf/OracleSpatial.pdf.

[Oro98] J. O'Rourke: *Computational Geometry in C*, Cambridge University Press, 1998.

[PaA02] D. Papadias and D. Arkoumanis: "Approximate Processing of Multiway Spatial Joins in Very Large Databases", *Proceedings of 8th EDBT Conference*, pp.179-196, 2002.

[PaD96] J.M. Patel and D.J. DeWitt: "Partition Based Spatial-Merge Join", *Proceedings of ACM SIGMOD Conference*, pp.259-270, 1996.

[PCC99] H.H. Park, G.H. Cha and C.W. Chung: "Multi-way Spatial Joins Using R-Trees: Methodology and Performance Evaluation", *Proceedings of 6th International Symposium on Spatial Databases* (SSD'99), pp.229-250, 1999.

[PMT99] D. Papadias, N. Mamoulis and Y. Theodoridis: "Processing and Optimization of Multiway Spatial Joins Using R-Trees", *Proceedings of 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (PODS'99), pp.44-55, 1999.

[PrS85] F.P. Preparata and M.I. Shamos: *Computational Geometry: an Introduction*, Springer-Verlag, 1985.

[PRS99] A. Papadopoulos, P. Rigaux, and M. Scholl: "A Performance Evaluation of Spatial Join Processing Strategies". *Proceedings of 6th Symposium on Large Spatial Databases* (SSD' 99), pp.286-307, 1999.

[RKV95] N. Roussopoulos, S. Kelley and F. Vincent: "Nearest Neighbor Queries", *Proceedings of ACM SIGMOD Conference*, pp.71-79, 1995.

[SCR+99] S. Shekhar, S. Chawla, S. Rivada, A. Fetterer, X. Lui and C. Lu: "Spatial Databases, Accomplishments and Research Needs", *IEEE Transactions on Knowledge and Data Engineering*, Vol.11, No.1, pp.45-55, 1999.

[SeK98] T. Seidl and H.P. Kriegel: "Optimal Multi-Step K-Nearest Neighbor Search", *Proceedings of ACM SIGMOD Conference*, pp.154-165, 1998.

[SMC+03] Y. Shou, N. Mamoulis, H. Cao, D. Papadias, and D.W. Cheung: "Evaluation of Iceberg Distance Joins", *Proceedings of 8th Symposium on Spatial and Temporal Databases* (SSTD'03), pp.270-288, 2003.

[SML00] H. Shin, B. Moon and S. Lee: "Adaptive Multi-Stage Distance Join Processing", *Proceedings of ACM SIGMOD Conference,* pp.343-354, 2000.

[YaL02] C. Yang and K.I. Lin: "An Index Structure for Improving Nearest Closest Pairs and Related Join Queries in Spatial Databases", *Proceedings of International Database Engineering and Applications Symposium* (IDEAS'02), pp.140-149, 2002.