Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 05-008

Exploiting a Page Level Upper Bound for Multi-Type Nearest Neighbor Queries

Xiaobin Ma, Shashi Shekhar, Hui Xiong, and Pusheng Zhang

March 31, 2005

Exploiting a Page-Level Upper Bound for Multi-Type Nearest Neighbor Queries

Xiaobin Ma[*], Shashi Shekhar, Hui Xiong, Pusheng Zhang
Computer Science Department, University of Minnesota - Twin Cities
[$xiaobin, shekhar, huix, pusheng$]@cs.umn.edu

Abstract

Given a query point and a collection of spatial features, a multi-type nearest neighbor query finds the shortest tour for the query point in a way such that only one instance of each feature type is visited during the tour. For example, a tourist may be interested in finding the shortest tour which starts at a hotel and passes through a post office, a gas station, and a grocery store. The multi-type nearest query problem is different from the traditional nearest neighbor query problem, since there are many objects for each feature type and the shortest tour should pass through only one object from each feature type. In this paper, we propose R-tree based optimal solutions, which exploit a page-level upper bound for efficient computation. Also, since this problem is a generalized Traveling Salesman Problem (TSP) and is NP-hard, we provide several heuristic methods for the case that there are a large number of feature types in the data. Finally, experimental results are provided to show the strength of the proposed algorithms and design decisions related to performance tuning.

1 Introduction

Widespread use of spatial search engines, such as MapQuest [†], is leading to an increasing interest in developing intelligent spatial query techniques. For example, a traveler may be interested in finding the shortest tour which starts at a hotel and passes through a post office, a gas station, and a grocery store. Therefore, it is critical to design an intelligent map query technique to efficiently find such a shortest tour. Indeed, in this paper, we formalize the above intelligent map query problem as a multi-type nearest neighbor (MTNN) query problem. Specifically, given a query point and a collection of spatial features, a multi-type nearest neighbor query finds the shortest tour for the query point such that only one instance of each feature type is visited during the tour.

[*]The contact author. Email: xiaobin@cs.umn.edu. Tel: 1-612-626-7703
[†]http://www.mapquest.com

In the real world, many spatial data sets include a collection of instances of spatial features (e.g. post office, grocery store, and hotel). Figure 1 illustrates a multi-type nearest neighbor query. In the figure, different shapes represent different spatial feature types. Given the query point **q** and a collection of spatial events {*, ∘, +, ×, □}, a multi-type nearest neighbor query is to find the shortest tour that starts at point **q**, passes through only one instance of each spatial event in the collection, and ends at point **q**, as the shortest route shown in the Figure.

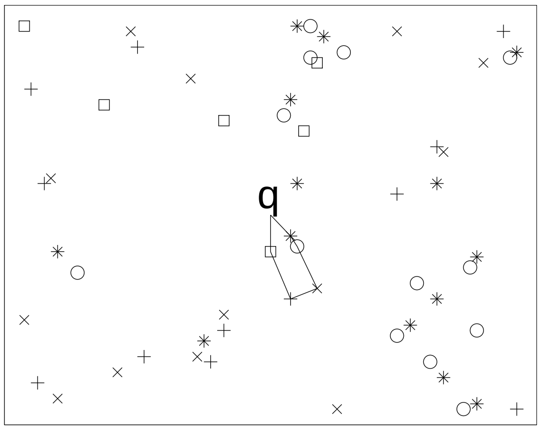


Figure 1: Point Spatial Co-location Patterns Illustration (Shapes represent different spatial feature types).

Indeed, the nearest neighbor (NN) query problem [6, 20, 5, 4, 23, 10, 24] has been studied extensively in the field of computer science. Traditional nearest neighbor query can be stated as follows: given a set $P = \{p_1, p_2, ...p_n\}$ and a query point q in a vector space, the NN query finds a point p_k such that the distance from q to p_k is minimized among the distances from q to $p_i \in P$. Many application domains are related to the NN query. For example, in Geographic Information System (GIS), "find the nearest gas station from my location" is a typical query that uses a nearest neighbor query technique. In addition, NN queries are used for some data analysis approaches such as clustering technique.

Recently, many other nearest neighbor query problems have attracted great research interests. All nearest neighbor (ANN) query [17, 2, 11, 12, 16] searches a nearest neighbor in a dataset A for every point in a dataset B. K-closest pair query [1, 3, 11, 12] discovers K-closest pairs within which a different point comes from a different dataset. Reverse nearest neighbor (RNN) query [18, 9, 13, 14, 7] finds a set of data that is the nearest neighbor of a given query point. Group nearest neighbor (GNN) query [8] retrieves a nearest neighbor for a given dataset. All of these problems focus on one or two data types and try to find relationships among data points within one or two object types. However, the relationship among more than two types of objects are important for many application domains.

In this paper, we study the multi-type nearest neighbor (MTNN) query problem. The MTNN problem can have many variations if spatial and/or time constraints are imposed on it. For instance, we may constrain the range of selected object set PO within a given circle or rectangle, the path can be from a query point q to all points in PO and return to q. If we know the visit order for part or all of the different feature types, it is a (partially) fixed order MTNN problem. Time constraint can also be part of the problem. For example, the post office might be open from 9:00am to 5:00pm so a visit has to be made during this period. However, our focus is on the generalized MTNN problem.

Related Work. One is the main memory algorithms that are mainly proposed in computational geometry. The other is the secondary memory algorithms using R-tree index.

The simplest brute force algorithm can find a nearest neighbor in $O(n)$ time. In the early period the main memory algorithms focused on developing efficient algorithms for datasets with specific distributions. Cleary analyzed on a uniformly distributed dataset the algorithms that partition the space into a regular grid in [6]. Bentley *et al.* used k-d tree to get an $O(n)$ space and $O(log(n))$ time query result [15]. Another partition based approach[21] used the well-known Voronoi graph. It first precomputed the Voronoi graph for the given dataset. For a given query point q, it just needed to use a fast point location algorithm to determine the cell that contain the query point q.

The first secondary memory algorithm[20] for the NN problem is based on R-tree index. It is a branch-and-bound algorithm in that it searches the R-tree using a depth first strategy and prunes the search space with the nearest neighbor found so far. It basically uses two metrics, the MINDIST and MINMAXDIST, to prune the impossible R-tree node in the search as soon as possible. MINDIST is the distance from query point q to an object O and MINMAXDIST is the face of MBR containing the object O.

The R-tree search begins at root node downward the leaf node. When necessary, the search will be upward. In downward search, all MBR with a MINDIST greater than the MINMAXDIST of another MBR will be discarded. In upward search, an object with a distance to query point q greater than the MINMAXDIST of query point q to a MBR will be discarded and the MBR with a MINDIST greater than the distance from query point q to an object is also discarded.

Hjalason *et al.* employed a priority queue to implement a best first search strategy in [12]. This algorithm is optimal in the sense that it just visits the nodes along the path from the root to the leaf node that contains the nearest neighbor.

After reaching the leaf node, our proposed algorithm needs to find the MTNN from the remaining subsets each of which contains at least one object from different types. This is similar to the traveling salesman problem (TSP)[22] that tries to find the shortest path from a given dataset such that every data object is visited exactly one time. TSP is a NP-hard problem and the time complexity of the optimal solutions to TSP are exponential. Our problem is not a standard TSP problem and harder than TSP. Some heuristic methods were proposed for large datasets. For example, the construction heuristic methods[22] construct a

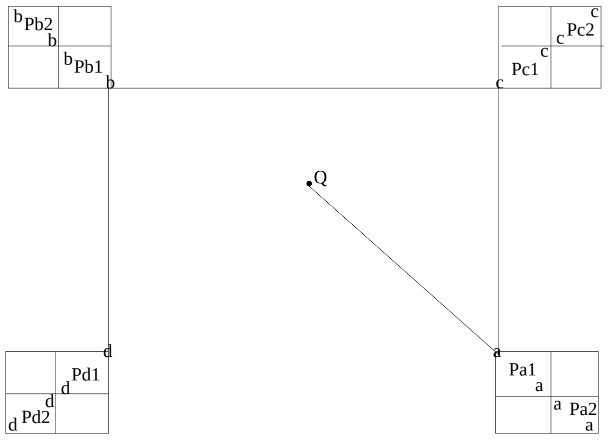


Figure 2: A Comparison of R-LORD and Our Approach

tour according to some construction rule and the improved methods [22] try to alter or improve the current tour according to some improving rules. Noe of these heuristic algorithms guarantee the optimal solutions.

In parallel of our work, Sharifzadeh *et al.* [19] recently proposed an Optimal Sequenced Route (OSR) query problem and provided three optimal solutions: Dijkstra-based, LORD and R-LORD. Essentially, the OSR problem is a special case of the MTNN problem investigated in this paper. Indeed, the OSR problem can be thought of as imposing a spatial constraint on the MTNN problem. Specifically, the order of feature types is fixed for the OSR problem.

In this paper, we study a generalized MTNN problem and provide both optimal and heuristic solutions to the problem. Based on R-tree index, we design algorithms which exploit a page-level upper bound for efficient pruning at the R-tree node level. In contrast, algorithms proposed for the OSR problem [19] apply instance-level pruning techniques for reducing the computation cost. Indeed, the R-tree node-level pruning method can be served as a nice complimentary technique to the instance-level pruning method, since the R-tree node-level pruning technique make better use of R-tree index for reducing I/O cost.

Let us consider the example shown in Figure 2, there are a query point Q and four feature types: a, b, c, and d. If we fix the visit order as $a \Rightarrow c \Rightarrow b \Rightarrow d$, our approach generates page sequence P_{a1}, P_{c1}, P_{b1}, P_{d1} and the search for the optimal route is restricted in this sequence. In contrast, for algorithms proposed for the OSR problem [19], all points in page P_{d1}, P_{b1}, P_{b2}, P_{c1}, P_{c2}, $P_{a1}\, and\, P_{a2}$ need to be examined in order to find an optimal route. In other words, more computation is required.

Our Contributions. We study a generalized multi-type nearest neighbor query (MTNN) problem and show that this problem is closely related to the Traveling Salesman Problem (TSP) [22], which is one of the class of "NP Complete" combinatorial problems. Indeed, if there is only one instance for each feature type, the MTNN problem becomes the TSP problem. Since the MTNN problem is essentially an NP-hard problem, we provide some optimal solutions as well as some heuristic algorithms to approach this problem.

In our algorithm, a page-level upper bound is exploited for efficient pruning at the R-tree node level. Our experiments demonstrate that optimal solutions become computationally intractable when the number of query feature types is large. In contrast, our heuristic algorithms, which are based on R-tree can be more applicable for a large number of query feature types.

Overview. The remainder of this paper is organized as follows. Section 2 presents some optimal solutions for the MTNN problem. Heuristic approaches are given in Section 3. Section 4 describes the experiment setup and experiment results. Finally, in Section 5, we conclude our discussion and suggest further work.

2 R-Tree Based Optimal Algorithms

In spatial databases, R tree and their variants are widely used for indexing spatial data. In this paper, we propose R-tree based algorithms for the MTNN query problem. Specifically, we design two types of algorithms. The first type gives optimal solutions and has exponential time complexity with respect to the number of feature types. The optimal algorithms work well when the number of feature types is small ($<$ 6). The second type gives heuristic solutions that are not guaranteed to find the optimal solution, i.e. the distance from query point q through each point from every feature type is not shortest. We will discuss the heuristic algorithms in the next section.

We have many feature types in MTNN problem. In order to find the optimal solution, we have to search the space consisting of all permutations of all feature type objects. For every permutation, we do same search steps, get a route with a shortest distance and finally find MTNN. In the following, we work on the search space consisting of one permutation of all feature type objects.

For one permutation of feature types, for example, $t_1, t_2, \ldots, t_k$, we need to find the optimal route from the query point through one point in every type in the order of $t_1, t_2, \ldots, t_k$. In the R-tree based optimal algorithms we use a branch and bound strategy to prune and search the space. The algorithms can be divided into three parts. The first part finds an upper bound for the R-tree search by using a simple fast algorithm. The second part prunes the search space based on R-tree using the current upper bound. The output of this part is candidate sequences consisting of leaf nodes each of which is from one of the R trees. The third part finds the current MTNN shortest distance from the current candidate sequence. Figure 3 illustrates these three parts. We will discuss them in detail in the rest of this section.

2.1 First Upper Bound Search

The first step of MTNN algorithms is to find the first upper bound for pruning the search space. This upper bound will determine the pruning efficiency for the R-tree search. The general requirements for the first upper bound search strategy are time efficiency and upper bound accuracy. Trade-offs will be made when

```
Input : K types of spatial objects and R-tree
        Distance metrics
Output : MTNN and the shortest path
MTNN
1.   step 1: First Upper Bound Search Find the first upper bound of MTNN shortest distance
2.       by using a fast greedy algorithm and set current upper bound to be this first upper bound
3.   step 2: R-Tree Search Prune search space to find subsets of objects that may contain MTNN
4.       and get a candidate sequence
5.   step 3: Subset Search Calculate current MTNN shortest distance in the current candidate sequence
6.       if current calculated MTNN shortest distance shorter than current upper bound
7.       then set current upper bound to be current calculated MTNN shortest distance
8.       if Some search space is not examined
9.       then Go to step 2
10.      else Report current upper bound as final MTNN shortest distance
```

Figure 3: Optimal R-tree based MTNN algorithm

designing an MTNN algorithm. In most cases, we prefer an algorithm with high time efficiency and normal upper bound accuracy. In this paper, we use a simple greedy algorithm as follows.

Randomly generate one permutation of feature types, for example, generate permutation $R = r_1, r_2, \ldots, r_k$. Search the nearest neighbor r_{1,i_1} of query point q in feature type r_1 by using basic R-tree based nearest neighbor search method. Then search the nearest neighbor r_{2,i_2} of r_{1,i_1} in feature type r_2. Repeat this procedure until all types of features are visited. Finally we get a path from q going through exact single point in each feature type. Calculate the distance of this path and use it as the first upper bound in MTNN search. We call this distance greedy distance.

2.2 R-Tree Search

The task of R-tree search is to prune the search space using a branch and bound approach on the R-tree index in spatial databases. We call the pruning methods used in this part R-tree node level pruning. For permutation $R = \{r_1, r_2, \ldots, r_k\}$ we first prune the search space using R-tree of type r_1, then using R-tree of type r_2, etc. More specifically, we first determine the possible leaf node rectangles to which the distance from the query point is less than the upper bound distance in R-tree of type r_1 using a general nearest neighbor search strategy. Next these rectangles are used to determine all possible leaf node rectangles to

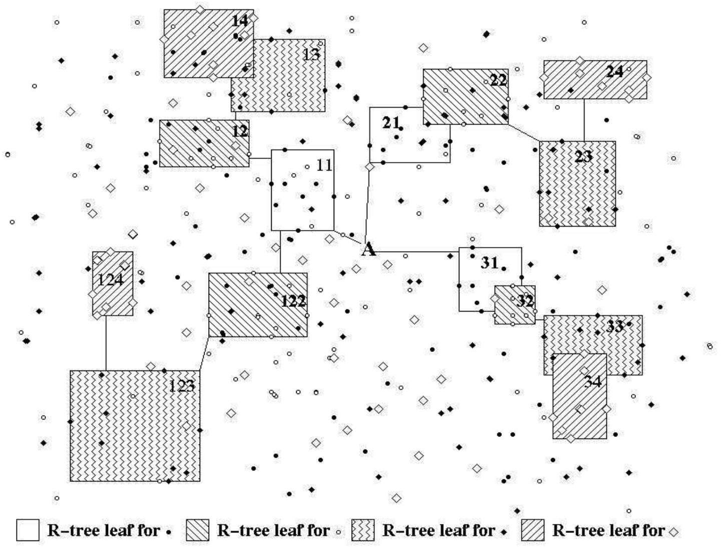


Figure 4: R Tree Search

which the distance from the query point is less than the upper bound distance in R-tree of type r_2. This procedure continues until all R-trees are visited. Finally, we get a list of candidate leaf node sequences among which each leaf node contains one type of feature objects. When searching R tree we choose to use a Depth First Search(DFS) strategy since DFS generates a route distance faster and we may use the new generated route distance as upper bound if it is smaller than current upper bound and thus prune R-tree nodes more efficiently.

Figure 4 illustrates this procedure. In this figure, A is the query point. All rectangles represent leaf nodes in different R-trees. Rectangles with an ID ending with 1 come from R tree 1 of feature type 1. Rectangles with an ID ending with 2 come from R tree 2 of feature type 2. etc. At the beginning of R tree search, an upper bound was calculated from step 1, First Upper Bound Search. The pruning algorithm first searches R tree 1 and finds leaf nodes, for example, rectangles 11, 21 and 31 etc., that possibly contains part of the path in the optimal route. The distances from the query point to these rectangles are less than the current upper bound. The shortest distance from the query point to rectangle 11 (or 21, 31) is deducted from the current upper bound to get a remaining upper bound distance and store it along with the R tree node in the following R tree search. At the next step, rectangles from last search are used to search R tree 2 with their remaining upper bound distance. This procedure continues until the newly calculated upper bound is less than 0, which means that MTNN cannot be included in the current rectangle sequence and we give up this rectangle sequence or we finish searching all R-trees. The result of this step is a set of leaf node rectangle sequences. Figure 4 shows rectangle sequences such as (11, 12, 13, 14), (11, 122, 123, 124), (21, 22, 23, 24) and (31, 32, 33, 34) etc that will contain the optimal route for this permutation.

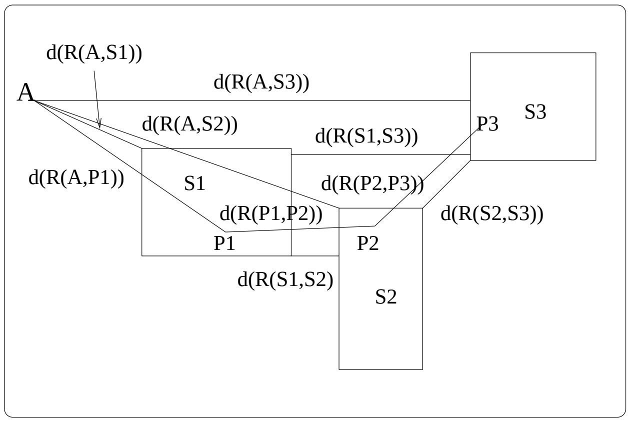


Figure 5: PLUB Pruning

2.2.1 Improvement and Pruning

When searching for the MTNN candidate sequences, we need an efficient pruning strategy to prune the search space. In this part, we introduce a newly proposed pruning method, page level upper bound(PLUB) pruning.

Page Level Upper Bound Pruning We use an example to illustrate the PLUB method. Assume $R(.)$ represent the route sequence and $d(R(.))$ represent the shortest distance from the the first object on $R(.)$ through all points or nodes in the road sequence. In Figure 5, assume S_1, S_2 and S_3 are three R tree nodes that can be internal nodes or leaf nodes coming from different R trees. Assume point P_1 in S_1, point P_2 in S_2 and point P_3 in S_3 are points in the optimal route, i.e. the route $R(A, P_1, P_2, P_3)$ is the optimal route. In the example, $d(R(A, S_1))$ is the shortest distance from A to rectangle S_1 and $d(R(A, S_1, S_2))$ is the shortest distance from A through S_1 to S_2, $d(R(A,\, S_1,\, S_2)) = d(R(A,\, S_1)) + d(R(S1,\, S_2))$. The general formula for the shortest distance d is $d(R(O_1,\, O_2, \ldots,\, O_n)) = \sum_{i=1}^{n-1} d(R(O_i,\, O_{i+1}))$.

There are many routes between two spatial objects through different other spatial objects. In our case, we enforce the route order to be the order of a fixed R tree permutation $S_1\, S_2\, S_3$. In Figure 5, routes from the query point A to node S_3 are $R(A,\, S_3)$, $R(A,\, S_1,\, S_3)$, $R(A,\, S_2,\, S_3)$ and $R(A,\, S_1,\, S_2,\, S_3)$. The distances of all these routes are shorter than $d(R(A,\, P_1,\, P_2,\, P_3))$. We define the PLUB from A to S_3 to be $max(d(R(A, S_3)), d(R(A, S_1, S_3)), d(R(A, S_2, S_3)), d(R(A, S_1, S_2, S_3)))$ and the path having PLUB to be the PLUB path. For example, if $d(R(A,\, S_2,\, S_3))$ is longest, the path $(R(A,\, S_2,\, S_3))$ is the PLUB path from A to S_3. We can extend this property to more R-trees and prune the search space more efficiently by using this property.

From the above discussion we define some concepts and infer a PLUB lemma to prune R trees efficiently.

Given a spatial object sequence $O_1, O_2, O_3, \ldots, O_{n-1}, O_n$, assume it defines a strict precedence order.

Definition 1 Path *A path from spatial object O_1 to O_n is defined to be a sequence $O_1, O_i, O_j, \ldots, O_k, O_n$.*

The intermediate objects O_i, $O_j, \ldots$, and $O_k \in \{O_t, 1 < t < n\}$. Sequence $O_i, O_j, \ldots, O_k$ follows the strict precedence order defined by sequence $O_1, O_2, O_3, \ldots, O_{n-1}, O_n$.

Definition 2 Intermediate Sequence *Given a path $O_1, O_i, O_j, \ldots, O_k, O_n$ from spatial object O_1 to O_n. The intermediate sequence is defined to be $O_i, O_j, \ldots, O_k$.*

Definition 3 Whole Path *The path from object O_1 to O_n through the longest intermediate sequence is called a whole path.*

Sequence $O_i, O_j, \ldots, O_k$ in a path can be empty. There will be many different paths from object O_1 to object O_n through different intermediate sequence.

Definition 4 PLUB path *The PLUB path from O_1 to O_n is the longest one among all paths that go through all possible intermediate sequences.*

For simplicity we sometimes say the PLUB path in sequence $O_i, O_j, \ldots, O_k, O_n$.

Definition 5 Page Level Upper Bound(PLUB) *The distance of the PLUB path is the PLUB.*

From the previous example we have the following property.
Property 1 For a spatial object sequence $O_1, O_2, O_3, \ldots, O_{n-1}, O_n$, O_is$(1 < i <= n)$ are R-tree nodes containing many points. If the optimal route from query point O_1 to another point in the object set of feature type n is contained in the whole path of sequence $O_1, O_2, O_3, \ldots, O_{n-1}, O_n$, the PLUB from O_1 to O_n is shorter than the distance of the optimal route.

Proof: Assume $O_1, P_2, P_3, ..., P_{n-1}, P_n$ is the optimal route and P_i is contained in node $O_i(1 < i <= n)$. Assume $O_1, O_i, O_j, \ldots, O_{t1}, O_{t2}, \ldots, O_n$ is the PLUB path from O_1 to O_n. For any adjecent nodes O_{t1},O_{t2}, $d(R(P_{t1}, P_{t2})) > d(R(O_{t1}, O_{t2}))$ because P_{t1} is inside O_{t1} and P_{t2} is inside O_{t2}. If $t2 = t1 + 1$ then $d(R(O_1, P_2, P_3, \ldots, P_{t1}, P_{t2}, \ldots, P_{n-1}, P_n)) > d(R(O_1, O_2, O_3, \ldots, O_{t1}, O_{t2}, \ldots, O_{n-1}, O_n))$. If $t2 > t1+1$ then $d(R(O_1, P_2, P_3, \ldots, P_{t1}, P_{t2}, \ldots, P_{n-1}, P_n)) > d(R(O_1, O_2, O_3, \ldots, O_{t1}, O_{t2}, \ldots, O_{n-1}, O_n))$ and $d(R(O_1, P_2, P_3, \ldots, P_{n-1}, P_n)) > d(R(O_1, P_2, P_3, \ldots, P_{t1}, P_{t2}, \ldots, P_{n-1}, P_n))$. Thus we have $d(R(O_1, P_2, P_3, \ldots, P_{n-1}, P_n)) > d(R(O_1, O_2, O_3, \ldots, O_{t1}, O_{t2}, \ldots, O_{n-1}, O_n))$.

Then it is easy to infer the following lemmas.

Lemma 1 PLUB Pruning *Any R-tree node sequence from a query point to the current visited R-tree node with a PLUB path longer than the current pruning upper bound cannot contain an MTNN optimal route and can be removed from the MTNN candidate sequence list.*

Proof: Because the distance of a PLUB path in current visited R-tree node sequence is shorter than the distance of an optimal route if (partial) optimal route is contained in this R-tree node sequence and the

PLUB path distance of a current visited R tree node sequence is longer than the current pruning upper bound, the current visited sequence cannot contain an (partial) optimal route.

Lemma 2 Correctness of PLUB Pruning *An optimal route is in the MTNN R-tree leaf node candidata sequence.*

Proof: Because the removed R-tree node sequences do not contain an optimal route, the PLUB pruning method is correct.

When searching candidate leaf node sequences, every time we reach a node and get a partial MTNN candidate sequence, the PLUB from the query point to this node rectangle is calculated. If this distance is longer than the current pruning upper bound, this partial candidate sequence is removed. Every time we visit an R-tree, we calculated the PLUB and these PLUBs and some intermidiate results can be used in next level PLUB calculation. Thus the computation of a PLUB is distributed into every R-tree visit.

Backward Improvement and Pruning After we get a new MTNN shortest distance shorter than the current upper bound distance, we can use this shortest distance to do backward improvement and pruning.

In an MTNN search, when reaching a leaf node in an R-tree, we deduct the PLUB from the current upper bound distance and get a remaining shortest distance. This remaining shortest distance will be kept and used in the next R-tree search. After we get a new MTNN path and new MTNN shortest distance, the search returns to the upper R-tree level. If the difference between an old MTNN shortest distance and new shortest distance is bigger than the remaining distance kept along with this upper R-tree level node, the current partial path is pruned. Even if the difference is smaller than the remaining shortest distance, it can be used to deduct the current remaining distance kept along with the upper level R-tree node. The consequence is that we have a smaller remaining distance and it is possible to prune more R-tree nodes in the next level search.

2.3 Subset Search

The algorithms for the subset search discussed in this section can work on the whole dataset.

In a subset search, we are given subsets of all different types of objects for all permutations of different feature types. For a specific permutation, all these points in subsets form a multi-level bipartite graph. The legal route consists of points each of which is from a different level of the graph. Many search algorithms such as BFS, DFS, $Dijkstra$, A^*, IDA^*, SMA^*, simulate annealing etc can be updated and used to find the optimal route. We call the methods used in this part point pruning. In the following, we give the simplest brute force algorithm and propose a dynamic programming method. Withour loss of generality, we assume that we have a set of subsets $S = \{S_1, S_2, \ldots, S_k\}$ of objects such that these subsets have $n_1, n_2, \ldots, n_k$ data points. S represents the R-tree pruning result for one permutation of all different object types. In the

R-tree based search, $S = \{S_1, S_2, \dots, S_k\}$ is actually a sequence of leaf node rectangles that contain a small number of data points.

2.3.1 Brute Force Algorithm for Subset Search

Query point and points from different candidate leaf nodes form a multi-level bi-partite graph. The simple brute force algorithm searches every branch in the graph exhaustively and find the MTNN and corresponding shortest path. Even for small numbers of object type, this algorithm is very time-comsuming.

2.3.2 Dynamic Programming for Subset Search

Using a dynamic programming technique, we are able to save search time by storing partial results in the search procedure.

Define $M(S_i(t),\, S_j(l))$ to be the shortest distance between tth point in set i and lth point in set j. Our final result can be expressed as

$$min_{0 \le h \le n_k}(M(S_0(0),\, S_k(h)))$$

We can infer the optimal structure of dynamic programming as:

$$M(S_i(t), S_j(l)) = min_{i<o<j,\, 0 \le m < n_0}(M(S_i(t),\, S_o(m)) + M(S_o(m),\, S_j(l)))$$

3 R-Tree Based Heuristic Algorithms

As we analyzed above, the complexity of the subset search is at least NP-hard in terms of data type k. When there are many numbers of data types in an MTNN problem, we have to design heuritic approaches in order to calculate good results within a reasonable time.

First we shrink the search space by removing most of the permutations of feature types. We only keep one among permutations with same first feature type. In the following, we work on the search space consisting of one permutation of all feature type objects.

Heuristics can be used in two aspects of an MTNN search. One is used in R-tree pruning and selection of the MTNN candidate leaf node sequence. The other is used in MTNN search after the MTNN candidate sequence is calculated. In this paper we focus on the first heuristic category.

As a simple heuristic method, greedy nearest neighbor method(Greedy) is described in first upper bound search part in last section. We discuss other heuristic methods in this section.

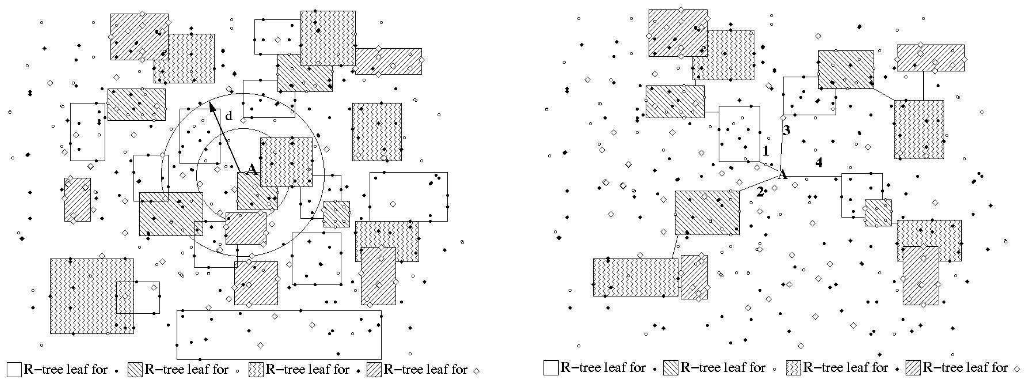
The procedure for the heuristic algorithms is very similar to that of the optimal R-tree based MTNN algorithms. Similarly we use a greedy method to find the first upper bound. We use this upper bound to prune search space of all possible heuristic MTNN leaf node sequences by searching R-trees and get candidate heuristic MTNN sequences. When searching the R-trees, we calculate the greedy distance in possible candidate heuristic MTNN sequences and use the candidate sequence with smallest greedy distance as the final candidate sequence. Finally we calculate heuristic MTNN distance in this final candidate sequence and get heuristic MTNN for this permutation.

In the following, we introduce three heuristic R-tree search strategies.

3.1 Heuristic MTNN Candidate Sequence Selection

3.1.1 Zone Expansion Method(Zone)

The Zone method tries to find heuristic MTNN in the MTNN candidate sequence consisting of nearer leaf nodes to the query point. First we search R-trees and sort the leaf nodes according to their distances to the query point in all these R-trees. All nearest leaf nodes from different R-trees in the order of current permutation form the first heuristic MTNN candidate sequence. Then the next nearer leaf node to q in an R-tree is selected and replaces the leaf node from the same R-tree in current heuristic MTNN candidate sequence to get a new MTNN candidate sequence. This procedure continues until reaching the stop criteria. Figure 6 (a) illustrates this method. The method tries to expand a circle zone around the query point and find a heuristic MTNN inside this zone. In the figure, the inner circle is the minimum circle zone that includes all kinds of page nodes. The outer circle is the circle with the first upper bound distance d. With the zone's expansion, leaf nodes farther from the query point are visited and we get new heuristic MTNN candidate sequences.



(a) Zone Expansion Strategy

(b) BFS Stratedgy

Figure 6: Zone Expansion and BFS MTNN Candidate Sequence Selection

3.1.2 Breadth First Search Method(BFS)

The BFS method tries to find a heuristic MTNN route in the R tree leaf node sequence with the shorter longest path distance. We search the R tree of first feature type in the current permutation of all feature types with a BFS strategy and sort all leaf nodes to get a queue of leaf nodes according to the PLUB to the query point. The head leaf node in the queue is then used as a range query rectangle to search the R-tree which is of the next feature type and get another queue of leaf nodes. This procedure continues until all R-trees are searched and we get a MTNN candidate sequence. We repeat this procedure and get more MTNN candidate sequences. Figure 6 (b) gives an example. Note the shorter path among R tree level leaf node paths has the shorter PLUB. Thus path 1 has shorter PLUB path than path 2.

3.1.3 Depth First Search Method(DFS)

This method is similar to BFS approach. We search the R-trees with DFS strategy and get a series of MTNN candidate sequences until reaching stopping criteria.

4 Experimental Results

In this section, we present the results of various experiments to evaluate both the optimal and heuristic algorithms for the multi-type nearest neighbor query. Specifically, we demonstrate: (1) a comparison of the optimal algorithms with respect to the execution time. (2) a comparison of the heuristic algorithms with respect to the execution time. (3) a comparison of the heuristic algorithms with respect to some statistical measures, such as average shortest distance.

4.1 The Experimental Setup

4.1.1 Experiment Implementation Platform.

Our experiments were performed on a PC with a 3.20GHz CPU and 1 GBytes memory running the GNU/Linux Ubuntu 1.0 operating system. All algorithms are implemented in the C programming language.

4.1.2 Experimental Data Sets

We evaluated the performance of both the optimal and heuristic algorithms for the multi-type query with synthetic data sets, which allow better control towards studying the effects of interesting parameters. All data points in the synthetic data sets were randomly distributed over a 10000x10000 plane. There were four types of synthetic data sets with 1000, 2000, 5000, and 10000 data points for each spatial feature respectively. For every type of synthetic data set, there was a different number of spatial features. In our experiments,

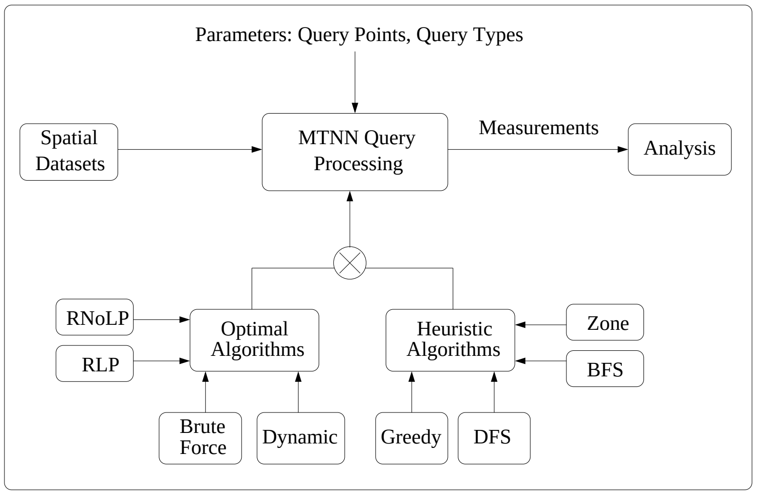


Figure 7: Experiment Setup and Design

we compared the performance of the optimal and heuristic algorithms with the increase of spatial features as well as data points for each spatial feature type.

4.1.3 Stability Measurements

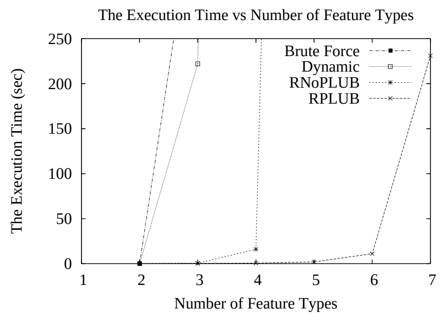
In order to evaluate the stability of the heuristic algorithms, we followed the classic assessment procedure used in the TSP algorithm [22]. Specifically, considering that a single run of a heuristic algorithm cannot guarantee a precise and correct assessment of the performance of the algorithm, we conducted the experiment for each permutation of query feature types and recorded a heuristic shortest distance for each run. Thus we get the same number of heuristic shortest distance as the number of query feature types. Finally, we calculated the average heuristic shortest distance and the standard deviation of the heuristic shortest distance to compare the performance of the investigated heuristic algorithms.

4.1.4 Experiment Design

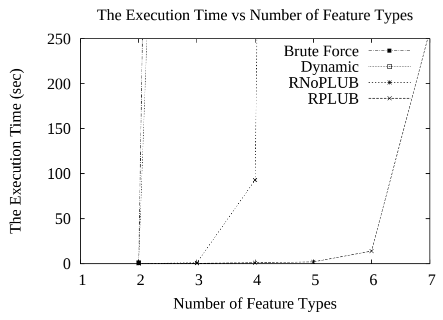
Figure 7 describes the experiment setup to evaluate the impact of design decisions on the relative performance of both the optimal and heuristic algorithms for the multi-type nearest neighbor query. We evaluated the performance of the algorithms with synthetic data sets. As shown in the figure, we considered four optimal algorithms: brute force, dynamic, RNoPLUB, and RPLUB. We also consider four heuristic algorithms, namely, the greedy method, Zone, DFS, and BFS. We observed the performance of the optimal algorithms with respect to the execution time. For the heuristic algorithms, we reported some stability measures as well as the execution time.

4.2 A Performance Comparison of Optimal Algorithms

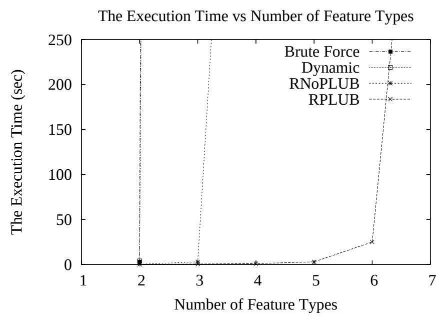
In this subsection, we present a performance comparison of optimal algorithms for the multi-type nearest neighbor query. In the experiment, we considered four algorithms, the brute force(Brute Force) method, the dynamic programming (Dynamic) method, the R-tree based method without PLUB pruning (RNoPLUB), and the R-tree based method with PLUB pruning (RPLUB).



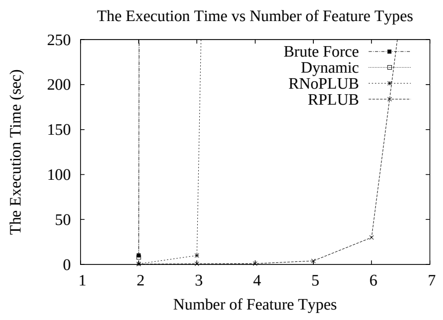
(a) 1000 Data Points



(b) 2000 Data Points



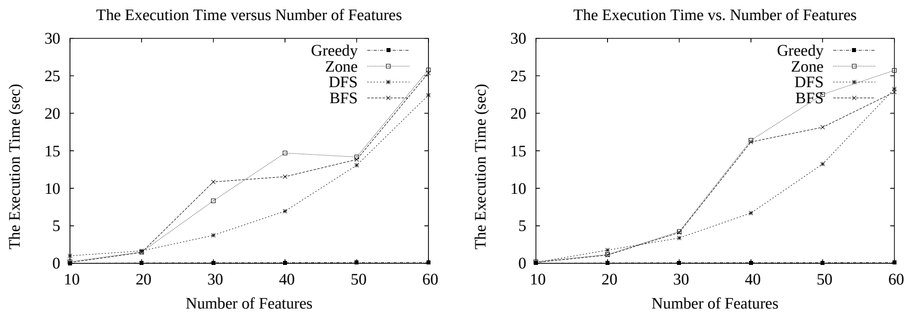
(c) 5000 Data Points



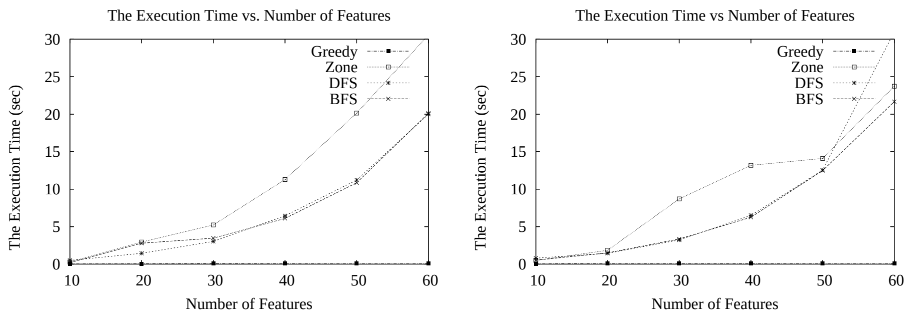
(d) 10000 Data Points

Figure 8: A Comparison of Optimal Algorithms with respect to the Execution Time.

Figure 8 shows the execution time of the four optimal algorithms on data sets with different numbers of data points for each feature type. As can be seen, with the increase of query feature types, the execution time of all algorithms goes up sharply. Indeed, when the number of query feature types is more than eight, all algorithms become very time consuming. For the different number of data points observed, we notice that the RPLUB algorithm can handle as many as six feature types without exponential growth of the execution time. Indeed, the RPLUB algorithm is the most scalable compared to the other three algorithms. However, when the number of query features rises, all optimal algorithms tend not to perform well due to the fact that this problem is essentially a NP-hard problem (Note that the complexity of this problem is even higher than that of the TSP problem).



(a) 1000 Data Points (b) 2000 Data Points



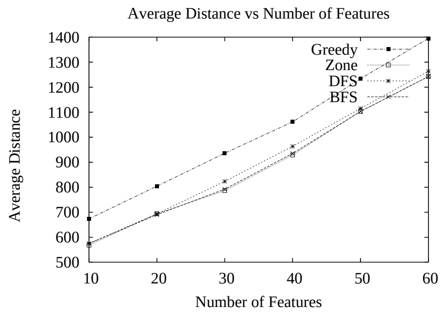
(c) 5000 Data Points (d) 10000 Data Points

Figure 9: A Comparison of Heuristic Algorithms with Respect to the Execution Time

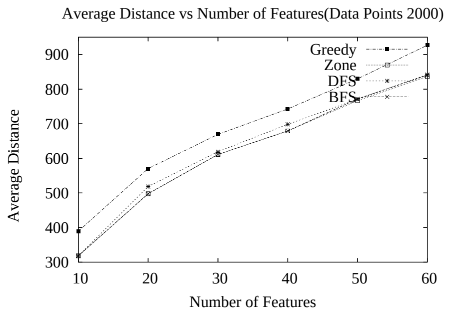
4.3 A Performance Comparison of Heuristic Algorithms

As discussed above, when the number of query feature types is large, all optimal algorithms become computationally intractable. However, real-life applications demand timely responds for MTNN queries. Towards solving this practical problem, we also investigate heuristic algorithms for the MTNN query problem. In this subsection, we compare the performance of four heuristic algorithms, namely greedy, Zone, DFS, and BFS. Specifically, we evaluate their time usage and their stability.

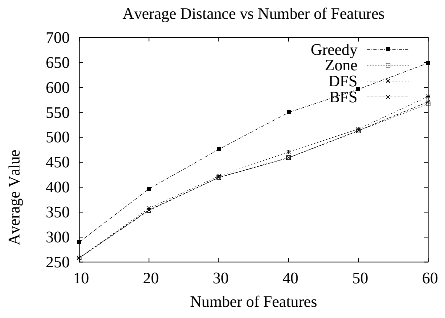
In the experiment, every heuristic algorithm searches five hundred candidate paths, which are generated as follows. Every candidate path starts with a different feature type. Other features types on the paths are selected randomly. Also, we have the same number of permutations as the number of query feature types.



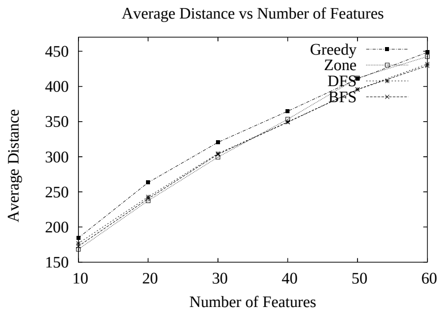
(a) 1000 Data Points



(b) 2000 Data Points



(c) 5000 Data Points



(d) 10000 Data Points

Figure 10: Average Shortest Distance by Heuristic Algorithms

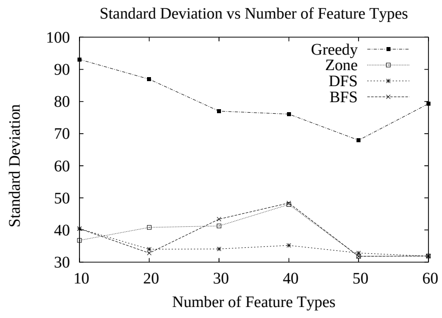
4.3.1 The Execution Time.

Figure 9 shows the execution time of the four heuristic algorithms on data sets with a different number of data points for each feature type. As shown in the figure, the execution time of all the algorithms rises with the increase of the number of query feature types. However, all the heuristic algorithms can easily handle the large number of query feature types compared to the optimal algorithms discussed in the previous section.

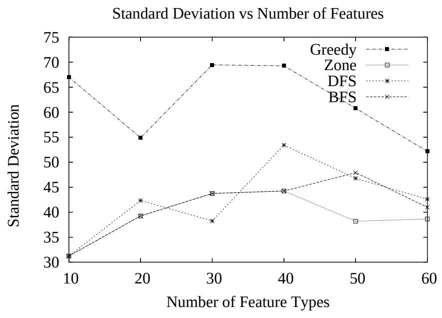
Another observation is that the greedy algorithm has the best run-time performance among the four algorithms, since fewer decisions need to be made for the greedy algorithm. Also, the Zone algorithm has the worst performance in terms of execution time. The other two algorithms have a similar trend for the execution time and the difference between them is not significant.

4.3.2 The Average Heuristic Shortest MTNN Distance

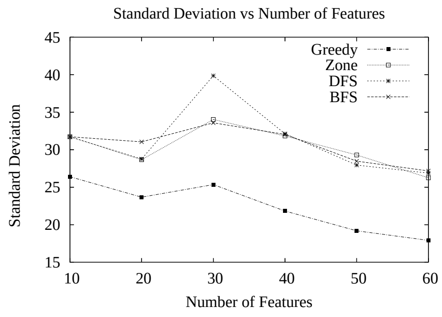
The average heuristic shortest MTNN distance measures the average quality of query results by heuristic algorithms. As shown in Figure 10, Zone and BFS have almost the same best performance in terms of the



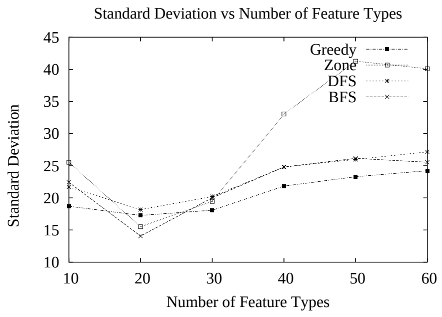
(a) Data Points 1000



(b) Data Pints 2000



(c) Data Points 5000



(d) Data Pints 10000

Figure 11: Standard Deviation of the Heuristic Shortest MTNN Distance

achieved average shortest MTNN results. The average query shortest distance of DFS is slighter worse that that of Zone and BFS. In addition, the Greedy method has the largest average shortest distance value for the query. This is the trade-off between the execution time and the quality of query results.

Another interesting observation is that, as the number of data points in the data sets increases, the difference of the average MTNN distance between the greedy algorithm and the other three algorithms becomes smaller. This is not surprising; for data sets with randomly distributed data points, it is more likely that the distance from a query point to the greedy nearest neighbor is closer to the optimal distance. Indeed, the greedy algorithm performs well for the data sets with randomly distributed data points and the result of greedy search becomes closer to the results of optimal solutions when the data sets become more dense.

4.3.3 Standard Deviation of the Heuristic Shortest MTNN Distance

Standard deviation of the heuristic shortest MTNN distance measures the stability of the heuristic algorithms. Figure 11 shows the standard deviation of the achieved shortest query distance by the four heuristic algorithms: Greedy, Zone, BFS, and DFS. When the number of data points is 1000 and 2000, the standard deviation of the greedy algorithm is much higher than that of the other three algorithms. In other words, the greedy algorithm is not a very stable heuristic algorithm for data sets with a smaller number of data points. In contrast, when the number of data points becomes large, the standard deviation of the greedy algorithm tends to become smaller than that of the other three algorithms. Especially when the number of data points is 5000, the greedy algorithm has the smallest standard deviation and is much more stable than the other three algorithms. When the number of data points increases to 10000, greedy, DFS and BFS have similar standard deviations. However, Zone becomes less stable.

5 Conclusions and Future Work

In this paper, we investigated a multi-type nearest neighbor (MTNN) query problem, which can be related to many application domains, such as intelligent map quest. We show that the MTNN problem is closely related to the TSP problem, but the computation complexity of the MTNN problem is much higher than that of the TSP problem. Consider that the MTNN problem is NP-hard, we propose R-tree based optimal solutions as well as heuristic solutions. In our algorithms, a page-level upper bound is exploited for efficient pruning at the R-tree node level. Finally, experimental results are provided to show the strength of the proposed algorithms and design decisions related to performance tuning.

As for future work, we plan to investigate heuristic algorithms from different perspectives. For instance, one direction is to design new heuristic algorithms using geometric properties of spatial data sets. Also, it is promising to design an algorithm which combines R-tree node-level pruning with point-level pruning [19].

References

[1] M. Vassilakopoulos A. Corral and Y. Manolopoulos. The impact of Buffering on Closest Pairs Queries Using R-Trees. In *Proceedings of Advances in Databases and Information Systems (ADBIS'01)*, pages 41–54. Springer-Verlag Lecture Notes in Computer Science, 2001.

[2] Y. Manolopoulos A. Corral and M. Vassilakopoulos. Closest Pair Queries in Spatial Databases. In *ACM SIGMOD*, 2000.

[3] Y. Manolopoulos A. Corral and M. Vassilakopoulos. Algorithms for Processing K-closest-pair Queries in Spatial Databases. *Data and Knowledge Engineering*, pages 67–104, 2004.

[4] A.N.Paradopoulos and Y.Manolopoulos. Performance of Nearest Neighbor Queries in R-trees. In *Proceedings of 6th ICDT Conference*, pages 394–408, 1997.

[5] K.L. Cheung and A.W.Fu. Enhanced Nearest Neighbor Search on the R-tree. *ACM SIGMOD Record*, pages 16–21, 1998.

[6] J. G. Cleary. Analysis of an algorithm for finding nearest neighbor in Euclidean space. *ACM Transactions on Mathematical Software*, pages 183–192, June 1979.

[7] C.Yang and K.-I. Lin. A index structure for efficient reverse nearest neighbor queries. In *ICDE*, 2001.

[8] Y. Tao D. Papadias, Q. Shen and K. Mouratidis. Group nearest neighbor queries. In *ICDE*, 2004.

[9] S. Muthukrishnan F. Korn and D. Srivastava. Reverse nearest neighbor aggregates over data stream. In *VLDB*, 2002.

[10] D.Agrawal H.Herhatosmanoglu, I.Stanoi and A.Abbadi. Constrained Nearest Nieghbor Queries. In *SSTD*, 2001.

[11] C. Hjaltason and H. Samet. Incremental Distance Join Algorithms for Spatial Databases. In *ACM SIGMOD*, 1998.

[12] C. Hjaltason and H. Samet. Distance Browsing for Spatial Databases. *ACM Transactions on Database Systems*, pages 265–318, 2 1999.

[13] D. Agrawal I. Stanoi and A. El Abbadi. Reverse nearest neighbor queries for dynamic databases. In *SIGMOD workshop on Research Issues in data mining and knowledge discovery*, 2000.

[14] D. Agrawal I. Stanoi, M. Riedewald and A. El Abbadi. Discovery of influence sets in frequently updated databases. In *VLDB*, 2001.

[15] J. L. Bentley J. H. Friedman and R. A. Finkel. An algorithm for finding best matches in logrithmic expected time. *ACM Transactions on Mathematical Software*, pages 209–226, September 1977.

[16] Dimitris Paradias Jun Zhang, Nikos Manoulis and Yufei Tao. All-Nearest-Neighbors Queries in Spatial Databases. In *16th International Conference on Scientific and Statistical Database Management (SSDBM)* , 2004.

[17] K.Clarkson. Fast Algorithms for the All-Nearest-Neighbors Problem. In *FOCS*, 1983.

[18] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *In Proc. of the 2000 ACM SIGMOD*, 2000.

[19] Mohammad Kolahdouzan Mehdi Sharifzadeh and Cyrus Shahabi. The optimal sequenced route query. In *University of Southern California, Computer Science Department, Technical Report 05-840*, January 2005.

[20] F. Vincent N. Roussopoulos, S. Kelly. Nearest Neighbor Queries. In *SIGMOD*, 1995.

[21] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer Verlag, New York, 1985.

[22] Gerhard Reinelt. *The Traveling Salesman - Computational Solutions for TSP Applications.* Springer Verlag, New York, 1994.

[23] D.A.Keim S.Berchtold, C.Bohm and H.P. Kriegel. A Cost Model for Nearest Neighbor in High-Dimensional Data Space. *PODS*, 1997.

[24] Dimitris Papadias Yufei Tao and Qiongman Shen. Continuous Nearest Neighbor Search. In *Proceedings of the 28th VLDB Conference*, 2002.