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

Xiaobin Ma[*]
University of Minnesota
xiaobin@cs.umn.edu

Shashi Shekhar
University of Minnesota
shekhar@cs.umn.edu

Hui Xiong
Rutgers University
hui@rbs.rutgers.edu

Pusheng Zhang
Microsoft Corporation
pzhang@microsoft.com

ABSTRACT

Given a query point and a collection of spatial features, a multi-type nearest neighbor (MTNN) query finds the shortest tour for the query point such that only one instance of each feature 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 MTNN query problem is different from the traditional nearest neighbor query problem in that 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 an R-tree based algorithm that exploits a page-level upper bound for efficient computation in clustered data sets and finds optimal query results. We compare our method with a recently proposed method, RLORD, which was developed to solve the optimal sequenced route (OSR) query. In our view, OSR represents a spatially constrained version of MTNN. Experimental results are provided to show the strength of our algorithm and design decisions related to performance tuning.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*

General Terms

Algorithms

Keywords

GIS, MTNN Query, location-based service

1. INTRODUCTION

Widespread use of search engines such as Google Maps and 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. 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 MTNN query finds the shortest tour for the query point such that only one instance of each feature type is visited during the tour.

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 MTNN query. In the figure, points with different colors represent different spatial feature types. Given the query point **q** and a collection of spatial events represented by black(b) points, white(w) points and green/gray(g) points, a MTNN query is to find the shortest tour that starts at point **q** and passes through only one instance of each spatial event in the collection as the shortest route shown in the Figure 1. In this figure, the solid line string route (q, w_{12}, g_3, b_{11}) is a shortest path. All other dashed line strings represent alternative routes from q through one point from each feature type.

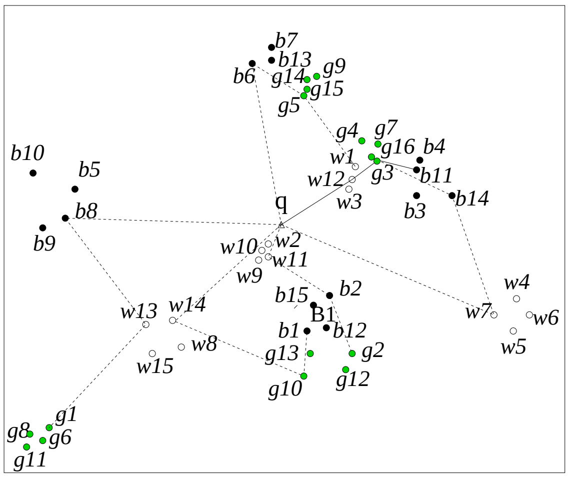

Figure 1: Illustration of a Multi-Type Nearest Neighbor Query

The nearest neighbor (NN) query problem [1, 2, 4, 10, 18, 20, 24] has been studied extensively in the field of computer science. A traditional NN query can be stated as follows: given a point 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

[*]Xiaobin Ma is currently with NCR Corp.

distance from q to $p_k \in P$ 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 Systems (GIS), "find the nearest gas station from my location" is a typical query that uses a NN query technique. In addition, NN queries are used for some data analysis techniques such as clustering.

Recently, many other NN query problems have attracted great research interest. All nearest neighbor (ANN) query [3, 5, 11, 12, 25] searches a nearest neighbor in a data set A for every point in a data set B. K-closest pair query [7, 6, 11, 12] discovers K-closest pairs within which a different point comes from a different data set. Reverse nearest neighbor (RNN) query [8, 13, 14, 22, 23] finds a set of data that is the NN of a given query point. Group nearest neighbor (GNN) query [17] retrieves a nearest neighbor for a given set of query points. 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, for many application domains, it is the relationship among more than two types of objects that is important.

In this paper, we study the 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, and 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. Previous work on NN can be classified in two groups. One consists of the main memory algorithms that are mainly proposed in computational geometry. The other is the category of secondary memory algorithms which use R-tree index.

The simplest brute force algorithm can find a NN in $O(n)$ time. In the early period the main memory algorithms focused on developing efficient algorithms for data sets with specific distributions. Cleary analyzed algorithms on a uniformly distributed data set that partition the space into a regular grid in [4]. Bentley *et al.* used k-d tree to get an $O(n)$ space and $O(log(n))$ time query result [9]. Another partition based approach[19] used the well-known Voronoi graph. It first precomputed the Voronoi graph for the given data set. For a given query point q, it just needed to use a fast point location algorithm to determine the cell that contained the query point q.

The first R-tree based algorithm[20] for the NN query problem was a branch-and-bound algorithm in that it searches the R-tree using a depth first strategy and prunes the search space with the NN 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 minimum of the maximum possible distances from p to a face of the Minimum Bounding Box(MBR) containing the object O.

The R-tree search begins at a root node downward to the leaf node. When necessary, the search will be upward. In a downward search, all MBRs with a MINDIST greater than the MINMAXDIST of another MBR will be discarded. In an 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 visits only the nodes along the path from the root to the leaf node that contains the NN.

In parallel with our work, Sharifzadeh *et al.* [21] 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 visiting order of feature types is fixed for the OSR problem.

Another recently published work [15] proposed a number of fast approximate algorithms to give sub-optimal solutions in metric space for Trip Planning Queries(TPQ); this is the same type of query we call a MTNN query in our paper.

In this paper, we study a generalized MTNN query problem and provide an optimal solution to the problem. Based on an R-tree index, we design an algorithm which exploits a page-level upper bound(PLUB) for efficient pruning at the R-tree node level. We originally formalized the MTNN query problem and presented algorithms for both optimal results and sub-optimal results in a technical report [16]. These algorithms are based on a page-level pruning strategy. In contrast, algorithms proposed for the OSR problem [21] apply instance-level pruning techniques for reducing the computation cost. In fact, the R-tree page-level pruning method can serve as a nice complimentary technique to the instance-level pruning method, since the R-tree page-level pruning technique makes better use of the R-tree index for reducing I/O cost. After discussion of our PLUB pruning strategy, we will give a detailed comparison of our method and the RLORD method, one of the solutions proposed by [21] for the OSR problem, introduced in [21]. Finally we give experiment results for both our method and the RLORD algorithm on clustered data sets.

Our Contributions. We provide an optimal solution algorithm for a generalized MTNN problem when the feature type number is small. In our algorithm, a page-level upper bound is exploited for efficient pruning at the R-tree node level. Our experiments show that our method outperforms RLORD with clustered data sets and that the optimal solution becomes computationally intractable when the number of query feature types is large.

Overview. The remainder of this paper is organized as follows. Section 2 formalizes the MTNN problem. Section 3 presents an R-tree based optimal solution for the MTNN problem. Section 4 compares the difference of our method with the RLORD algorithm, using a specific example. The experimental setup and experiment results are provided in Section 5. Finally, in Section 6, we conclude our discussion and suggest further work.

2. PROBLEM FORMULATION

In this section, we introduce some basic concepts, describe some symbols used in the rest of the paper and give a formal problem statement for the MTNN query problem.

Let $< P_1, P_2, ..., P_k >$ be an ordered point sequence and $P_1, P_2, ..., P_k$ be from k different (feature) types of data sets. $R(q, P_1, P_2, ..., P_k)$ is a route from q though points $P_1, P_2, ...,$ and P_k and $d(R(q, P_1, P_2, ..., P_k))$ represent the distance of route $R(q, P_1, P_2, ..., P_k)$. Similarly, with R_i representing the tree node of feature type i we define a page-level upper bound(PLUB) as $d(R(q, R_1, R_2, ..., R_k))$, the longest distance of route $R(q, R_1, R_2, ..., R_k)$.

A Multi-Type Nearest Neighbor (MTNN) is defined to be the ordered point sequence $< P'_1, P'_2, ..., P'_k >$ such that $d(R(q, P'_1, P'_2, ..., P'_k))$ is minimum among all possible routes. $d(R(q, P'_1, P'_2, ..., P'_k))$ is the MTNN distance. A MTNN query is a query finding MTNNs in given spatial data sets.

The following descriptions characterize a formal definition for the MTNN query problem.

Problem: The Multi-type Nearest Neighbor (MTNN) Query

Given:

- A query point, distance metric, k feature types of spatial objects and R-tree for each data set

Find:

- Multi-type Nearest Neighbor (MTNN)

Objective:

- Minimize the length of route from a query point covering an instance of each feature

Constraints:

- Correctness: The tour should be the shortest path for the query point and the given collection of spatial query feature types.
- Completeness: Only the shortest path is returned as the query result.

3. A PAGE-LEVEL UPPER BOUND (PLUB) ALGORITHM

In spatial databases, R trees and theirs variants are widely used for indexing spatial data. In this paper, we propose PLUB, an R-tree based algorithm, for the MTNN query problem. Specifically, we design an R-tree based page-level pruning method to filter out large numbers of spatial objects. PLUB identifies an optimal solution and its exponential time complexity grows exponentially with respect to the number of feature types, but not the average number of instances for feature types.

We have many feature types in a MTNN problem. In order to find the optimal solution, we have to search a space consisting of all permutations of all feature type objects. For every permutation, we do the same search steps and get a route with a shortest distance. Thus for total N permutations, we get N routes. Finally we find the solution to the MTNN problem by taking the route with the shortest distance from these N routes. For the sake of convenience, our discussions are based on a search space consisting of one permutation of all feature type objects in the following.

For one permutation of feature types $t_1, t_2, \ldots, t_k$, 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 algorithm we use a branch and bound strategy to prune and search the space. The algorithm can be divided into three parts. The first part finds an upper bound for the R-tree search. 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 2 illustrates these three parts. We will discuss them in detail in the rest of this section.

```
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
 2.         first upper bound of MTNN shortest distance
 3.         by using a fast greedy algorithm and set current
 4.         upper bound to be this first upper bound
 5.     step 2: R-Tree Search Prune search space to
 6.         find subsets of objects and get candidate
 7.         sequences for one permutation of feature types
 8.     step 3: Subset Search Calculate current MTNN
 9.         shortest distance in current candidate sequences
10.         if current calculated MTNN shortest distance
11.             shorter than current upper bound
12.         then set current upper bound to be current
13.             calculated MTNN shortest distance
14.         if Search space of some permutations are not
15.             examined
16.         then Go to step 2
17.         else Report current upper bound as the final
18.             MTNN shortest distance
```

Figure 2: The PLUB Algorithm

3.1 First Upper Bound Search

The first step of the MTNN algorithm 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 designing a 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 NN r_{1,i_1} of query point q in feature type r_1 by using a basic R-tree based NN search method. Then search the NN 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 query point q going through an exact single point in each feature type. Calculate the distance of this path and use it as the first upper bound in the MTNN search. We call this distance the greedy distance r_g.

3.2 R-Tree Search

In spatial databases, the task of an R-tree search is to prune the search space using a branch and bound approach on the R-tree index. We call the pruning method used in this part R-tree page-level pruning. For permutation $R = \{r_1, r_2, \ldots, r_k\}$ we first use a general NN search strategy to determine in the R-tree of type r_1 the possible leaf node rectangle set S_1 such that $d(R(q, R_{s1})$ $(R_{s1} \in S_1)$ is less than the upper bound distance. Next the rectangle set S_1 is used to determine the possible leaf node rectangle set S_2 in the R-tree of type r_2 such that the distance $d(R(q, R_{s1}, R_{s2}))$ $(R_{s1} \in S_1, R_{s2} \in S_2)$ is less than the upper bound distance. 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 object. When searching R trees 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 an upper bound if it is shorter than the current upper bound and thus prune R-tree nodes more efficiently.

3.3 Subset Search

In a subset search, we are given subsets of all different types of objects for one permutation of feature types. For a specific permutation, all these points in the 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^* etc can be updated and used to find the optimal route. We call the methods used in this part point pruning. In [16], a simple brute force algorithm and a dynamic programming method were given. In this paper, we use the RLORD algorithm[21] as another search method in our subset search.

3.4 Solution Optimality

Lemma PLUB finds an optimal solution to the MTNN Query

Proof The first upper bound calculated in step 1 of Figure 2 is longer than the distance of the optimal solution. All candidate leaf node sequences pruned by step 2 for a specific permutation have a PLUB longer than the current upper bound. Since any $d(R(q, P_1, P_2, ..., P_k))$, $(P_1 \in R_1$, $P_2 \in R_2$,...,$P_k \in R_k)$, is longer than $d(R(q, R_1, R_2, ..., R_k))$, candidate solutions pruned by PLUB cannot be better than the solution chosen by PLUB finally. At step 3, we find the candidate MTNN with the shortest distance from the leaf node candidate sequences. The smaller of the shortest distance and the current upper bound becomes the new current upper bound. If other permutations are not examined, return step 2. Otherwise, the current upper bound is reported as the MTNN query optimal solution.

4. COMPARISON OF PLUB AND RLORD

4.1 Comparison by Example

Here, we illustrate our proposed PLUB-based MTNN algorithm and compare it to R-LORD by using an extended example from [21]. Basically, a MTNN problem reduces to an OSR problem for a fixed permutation of feature types. The following discussion is based on a fixed permutation.

In the example of Figure 3 we assume the permutation is (w, b, g) and the distance metric is the Euclidian distance. The order of the R-tree is 4. There are three different feature types represented by black(b), white(w) and green/gray(g) points. In Figure 3, $R(q, w_2, b_2, g_2)$ is the greedy route and the radius of the search circle is $d(R(q, w_2, b_2, g_2))$. **q** is the query point represented as $\triangle$ and the rectangles represent the leaf nodes of the R-tree indices for different feature types. Figure 4 gives the R-tree structure for feature types green, black and white.

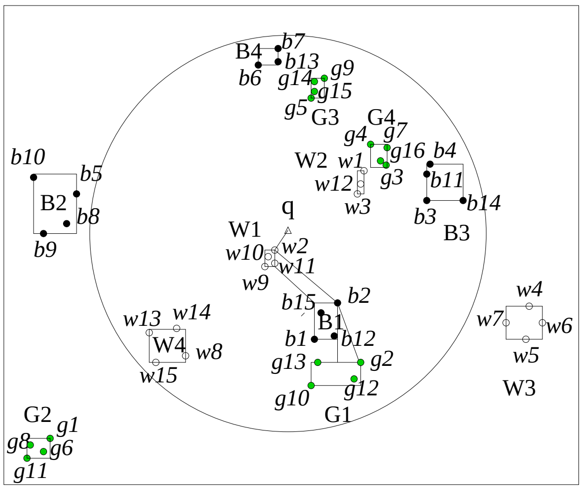

Figure 3: An Example for PLUB and RLORD

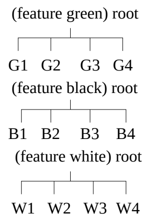

Figure 4: R-trees

The first step in PLUB is the same as in R-LORD: look for the first upper bound distance. The algorithm first finds NN w_2 of q in all objects of feature type w. Then b_2 of feature type b is found as the NN of w_2. Next, g_2 of feature type g is found as the NN of b_2. Finally we get greedy route R_g (q, w_2, b_2, g_2) with greedy distance $D_g = d(R(q, w_2, b_2, g_2)) = 3.37$ as the current upper bound D_u.

In the R-tree search, leaf node W_1 is inside the upper bound circle, so the partial route is expanded to be (q, W_1). Next, the R-tree of feature type b is searched, and leaf nodes B_1, B_3, B_4 are added to the current partial route (q, W_1) because the PLUB of partial routes (q, W_1, B_1), (q, W_1, B_3) and (q, W_1, B_4) is less than the current upper bound. Then we search the R-tree of feature type g and find that the PLUB of only one route (q, W_1, B_1, G_1) is less than the current upper bound. Thus in the subset search step, we only need to look for the shortest route from query point q through points inside leaf nodes W_1, B_1 and G_1. Table 1 gives the detailed calculation results.

			Upper Bound	Eliminated
W_1	B_1	G_1	2.04	N
W_1	B_1	G_3	6.2	Y
W_1	B_1	G_4	4.27	Y
W_1	B_3	G_1	7.53	Y
W_1	B_3	G_3	6.54	Y
W_1	B_3	G_4	4.29	Y
W_1	B_4	G_1	4.02	Y
W_2	B_1		3.7	Y
W_2	B_3	G_4	3.43	Y
W_2	B_4		5.17	Y
W_4	B_1		4.08	Y
W_4	B_3		7.94	Y
W_4	B_4		7.56	Y

Table 1: Calculation Results of PLUB Leaf Node Sequences

When searching candidate MTNNs in route $R(q, W_1, B_1, G_1))$, the first iteration does 4 point-to-point$(P-P)$ calculations and finds partial routes $R(q, g_2)$, $R(q, g_{10})$, $R(q, g_{12})$ and $R(q, g_{13})$. Similarly, iteration 2 gives partial routes $R(q, b_{12}, g_{13})$, $R(q, b_1, g_{13})$, $R(q, b_2, g_2)$ and $R(q, b_{15}, g_{13})$ with 20 $(P-P)$ calculations. Finally we get $R(q, w_{10}, b_{15}, g_{13})$, $R(q, w_9, b_{15}, g_{13})$, $R(q, w_2, b_2, g_2)$, and $R(q, w_{11}, b_1, g_{13})$ with 20 P-P calculations. After this step, the current MTNN is $R(q, w_{11}, b_1, g_{13})$ with distance 3.16. This procedure takes 44 total P-P calculations.

In R-LORD, initially the partial route set is $S = \{(g_2), (g_3), (g_4), (g_5), (g_7), (g_9), (g_{10}), (g_{12}), (g_{13}), (g_{14}), (g_{15}), (g_{16})\}$. In the first iteration, every black point x inside T_c (range query Q1) and MBR(Q2) (range query Q2) is checked for every green/gray point in S. If $D(p, x)+D(x, P_1)+L(PSR) \leq T_c$, then point x is added to the head of the partial route. When x is b_1, for example, we get partial route $(b_1, g_{10}), (b_1, g_{13})$. By using property 2, only partial routes with shortest length will be kept. So, (b_1, g_{13}) is put into a new partial route set. At the end of iteration 1, we have partial route set $\{(b_1, g_{13}), (b_2, g_2), (b_3, g_3), (b_4, g_3), (b_6, g_{14}), (b_7, g_{14}), (b_{11}, g_3), (b_{12}, g_{13}), (b_{13}, g_{14}), (b_{14}, g_3), (b_{15}, g_{13})\}$. By using property 2, we dramatically reduce the size of the partial route set. However, property 2 can only be used in iteration 1. Following a similar procedure, each of subsequent $(m-2)$ iterations will check every point of the feature type inside T_v (range query Q1) and MBR(Q2) (range query Q2) for every partial route in the current partial route set S. Finally we get route set $\{(w_1, b_{11}, g_3), (w_2, b_2, g_2), (w_3, b_{11}, g_3), (w_8, b_1, g_{13}), (w_9, b_{15}, g_{13}), (w_{10}, b_{15}, g_3), (w_{11}, b_1, g_{13}), (w_{12}, b_{11}, g_3), (w_{13}, b_1, g_{13}), (w_{14}, b_1, g_{13}), (w_{15}, b_1, g_{13})\}$ and $R(q, w_{11}, b_1, g_{13})$ is shortest among all routes. This procedure takes a total of 298 P-P calculations.

To summarize, PLUB needs 17 rectangle-to-rectangle distance calculations and 44 $P-P$ distance calculations in this example. RLORD takes 298 $P-P$ calculations. Apparently PLUB requires less computation.

As seen in the above illustration, the PLUB method uses a page-based pruning approach, while R-LORD uses a point-based search method. If the number of points inside the query range in R-LORD becomes big, the size of the partial route set will increase significantly. For every partial route inside a current partial route set, every point of the following feature type inside the current query range in R-LORD needs to be checked, which takes a lot of time.

4.2 Comparison by Cost Models

In this section, we provide algebraic cost models for PLUB and RLORD. The MTNN query is a CPU intensive task, and the CPU cost is at least as important as the I/O cost for data sets with medium and high numbers of feature types of spatial data. We will explore the I/O cost model and give a whole cost analysis for PLUB and RLORD.

As we discussed in section 3, the costs for the proposed PLUB include (1) the search of the R-tree leaf nodes inside the current search range, (2) page-level leaf node candidate sequence pruning and (3) a point-level candidate MTNN search. Therefore, the cost model also has three components: (1) cost of the page-level R-tree traversal, (2) cost of the page-level leaf node candidate sequence search, and (3) cost of the point-level candidate MTNN search. We also identify the cost components of RLORD as (1) cost of the page-level R-tree traversal and (2) cost of the point-level candidate MTNN search. In PLUB, the page-level leaf node candidate sequence search will possibly prune many more candidate sequences so the cost of the point-level candidate MTNN search will possibly be much smaller than that in RLORD. In the following, we discuss the cost models for PLUB and RLORD respectively.

4.2.1 A Cost Model for PLUB

Let C_{R-T} be the cost of the R-tree traversal to find all R-tree leaf nodes intersected by the circle with radius of the current upper bound, centered at the query point. In addition, let C_{LF} be the page-level leaf node search cost for the R-tree candidate leaf node sequences and C_{PN} be the point-level search cost for candidate MTNNs in candidate leaf node sequences. Thus the total cost of PLUB is expressed as $C_{R-T} + C_{LF} + C_{PN}$.

PLUB first traverses all the R-trees to look for leaf node rectangles that intersect with the current search range. Let C_{PR} be the cost of the point-to-rectangle distance calculations and $N_{t,i}$ be the number of all the tree nodes visited in the feature type i tree traversal. Thus $C_{R-T} = C_{PR} \times \Sigma N_{t,i} (i = 1, ..., k)$ (k is the number of feature types). It is worth noting that C_{R-T} is the same for PLUB and RLORD.

Next PLUB does a page-level search for leaf node candidate sequences. Let N_{R-R} be the number of leaf nodes visited in candidate leaf node sequences, and C_{R-R} be the cost of rectangle-to-rectangle distance calculation. Then we can get the cost of leaf node candidate sequence pruning as $C_{LF} = N_{R-R} \times C_{R-R}$.

Finally we search for candidate MTNNs in the remaining leaf node candidate sequences. Let F_{LS} be the leaf node candidate sequence filtering ability ratio, n_l be the average point number in leaf node for all feature types and p_i be the page number of feature type i. We use C_{ls} to denote the cost of the MTNN search in single leaf node sequence to arrive at the following cost:

$$C_{ls} = n_l + (n_l \times n_l) + n_l + (n_l \times n_l) + ... + n_l + (n_l \times n_l)$$
$$(k-1 \text{ items})$$

The condensed form is:

$$(k-1)(n_l \times (n_l + 1))$$

Thus the total point-level candidate MTNN search cost is $C_{PN} = C_{ls} \times \Pi p_i \times (1 - F_{LS}), (i = 1, ..., k)$

4.2.2 A Cost Model for RLORD

Let C_{R-T} be the cost of R-tree based coarse pruning, i.e., finding all data points inside the initial upper bound, and let C_{PS} be the cost of the candidate MTNN search in the remaining subsets. The cost C_{R-T} is the same as for PLUB. The cost C_{PS} is:

$$n_l \times (p_1 + n_l \times p_1 \times p_2 + (p_2 + n_l \times p_2 \times p_3) + ... + (p_{k-1} + n_l \times p_{k-1} \times p_k)$$

The total cost of RLORD is $C_{R-T} + C_{PS}$.

4.2.3 A Cost Model Comparison of PLUB and RLORD

Lemma PLUB is more efficient than RLORD for clustered data sets

Proof There are many factors to consider when comparing the cost models of PLUB and RLORD. We may consider

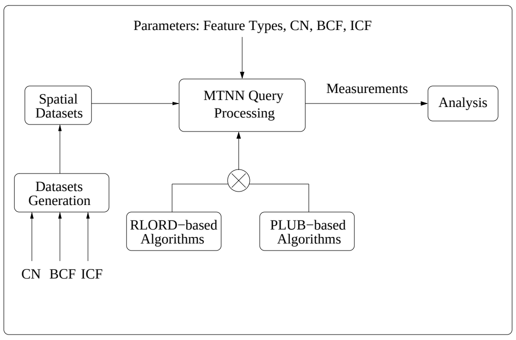

Figure 5: The Experiment Setup and Design

simplifying these models by focusing on the dominant factors and therefore removing some terms. In PLUB we may assume that $C_{R-T} + C_{LF} << C_{PL}$ and get the approximate cost model as:

$$(k-1)n_l \times (n_l + 1) \times \Pi p_i \times (1 - F_{LS}).$$

Similarly, if we assume $C_{R-T} << C_{PS}$, R-LORD's cost model becomes:

$$n_l \times (p_1 + n_l \times p_1 \times p_2 + (p_2 + n_l \times p_2 \times p_3) + ... + (p_{k-1} + n_l \times p_{k-1} \times p_k)$$

In random or approximate random data sets, F_{LS} is small, and PLUB takes more time. The opposite is true in clustered data sets, where F_{LS} tends to be bigger. That is when

$$1 - F_{LS} < n_l \times (p_1 + n_l \times p_1 \times p_2 + (p_2 + n_l \times p_2 \times p_3) + ... + (p_{k-1} + n_l \times p_{k-1} \times p_k))/((k-1)n_l \times (n_l + 1) \times \pi p_i)$$

PLUB runs more efficient than RLORD.

Later in the discussion of the experimental results, we'll refer to this formula 1, and refer to the left side as the remaining ratio ($r - ratio$), and the right side as the comparison ratio ($c - ratio$).

5. EXPERIMENTAL RESULTS

In this section, we present the results of various experiments to evaluate our PLUB based algorithm and the RLORD based algorithm, both of which give optimal solutions, for the MTNN query in different clustered data sets. Specifically, we demonstrate comparisons of the PLUB and RLORD based algorithms with respect to execution time under data sets with different properties such as feature type number, data set density and compactness of clusters.

5.1 The Experimental Setup

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

Experimental Data Sets We evaluated the performance of both the PLUB and RLORD based algorithms for the MTNN 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 distributed over a 10000X10000 plane and formed clustered data sets. In order to reduce the effect of query point positions, we took 25 query points on a sample data set space, each of whose x and y axis values were from 3000.00 to 7000.00 respectively and with each point placed 1000.00 away from its neighbor in the x and y axis directions, and calculated the average running time, $c - ratio$ and $r - ratio$ as the final reported values. There were four different parameters in our experimental setup.

- Feature Type(FT): Feature type numbers from 2 to 7 to show the scalability of both algorithms. In real GIS applications, the feature type number is normally less than 10.
- Between-cluster Compactness Factor(BCF): to control the minimum distance of cluster centers, i.e. the compactness between clusters.
- In-cluster Compactness Factor(ICF): to control the compactness within a cluster.
- Cluster Number(CN): to control the density of the data sets.

For a given cluster number ClusterNumber, we generated a data set as follows. First a simplified estimated maximum number of cluster center distance was determined by formula $maxCCDist = 10000.0/(int)(\sqrt{ClusterNumber} + 1)$. Next the minimum cluster center distance was calculated as follows $minCCDist = BCF \times maxCCDist$. Finally, we decided the cluster size by $ClusterSize = ICF \times minCCDist$ The number of objects inside each cluster is within $p/2$ and p, 84 in our experiment setting, that is the order of R-tree leaf node. Thus the expected number of objects inside a single cluster is about 61. For a data set of 20 clusters, the total object number is therefore about 1220.

Experiment Design Figure 5 describes the experimental setup to evaluate the impact of design decisions on the relative performance of both the PLUB and RLORD based algorithms for the MTNN query. We evaluated the performance of the algorithms with synthetic data sets generated according to the rules discussed above. We observed the performance of both PLUB and RLORD based algorithms under different data set settings in term of execution time. Our goal was to answer the following questions: (1) How do changes in feature type number affect scalability in PLUB and RLORD? (2) How do differences in data density affect the performance of PLUB and RLORD? (3) How does compactness between clusters affect the performance of PLUB and RLORD? (4) How does compactness within clusters affect the performance of PLUB and RLORD?

5.2 A Performance Comparison of PLUB and RLORD With Different Feature Types

This section describes the scalability improvement of PLUB in terms of feature types in clustered data sets, compared to RLORD. We set the fixed cluster number at 20, the BCF at 0.1, which means the minimum cluster center distance was 10% of maxCCDist and the ICF at 0.1, which means the size of a cluster was 10% of the minimum distance between two clusters. This is a highly clustered data set in that the size of clusters is 1% of maxCCDist. We change the number of feature types from 2 to 7 and don't show the results with feature type number 1 because that case would reduce the MTNN query problem to the classic NN problem, making PLUB and RLORD no more than classic NN algorithms.

Figure 6 compares the scalability of PLUB and RLORD in terms of numbers of feature types. More specifically, this

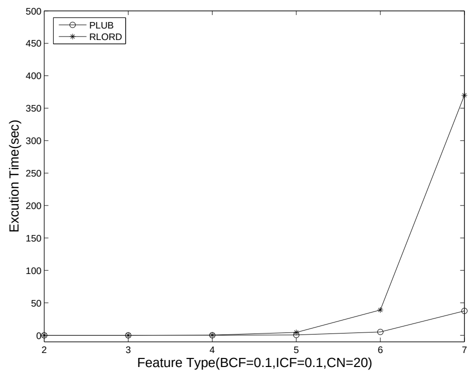

Figure 6: The Scalability of PLUB and RLORD in Terms of the Number of Feature Types

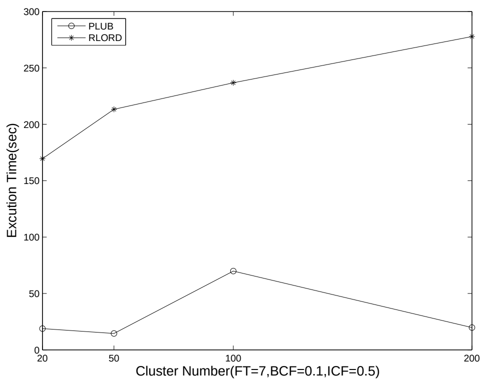

Figure 7: A Performance Comparison of PLUB and RLORD On Data Sets with Different Densities

figure illustrates that the execution time change with the increase of data types from 2 to 7 when the minimum distance between clusters is small (BCF=0.1) and cluster size is small (ICF=0.1) When the data type number is 2,3,4 and 5, there is no big difference of performance between PLUB and RLORD. When the data type number is 6 and 7, PLUB runs less time than RLORD. This experiment shows that PLUB is more scalable than RLORD in highly clustered data sets.

5.3 The Effect of the Data Set Density

Here, we show how the density of data sets affects the performance of PLUB and RLORD. We tested PLUB and RLORD with feature type number 7, BCF 0.1, and ICF 0.5, which means the size of a cluster was 50% of the minimum distance between two clusters. The changing variable is the cluster number, with assigned values of 20, 50, 100 and 200. Because there are almost the same average number of data points inside clusters for data sets of different cluster numbers, these data sets on the same space represent data sets with different densities.

Figure 7 illustrates the performance of PLUB and RLORD on different densities of data sets. As can be seen, under all data set densities with cluster numbers 20, 50, 100 and 200, the execution time of PLUB is always less than RLORD. In this figure we cannot see significant change in execution time, or any apparent trend for either PLUB and RLORD, which means the data set density appears to have almost no effect on execution time of PLUB and RLORD in clustered data sets with current settings.

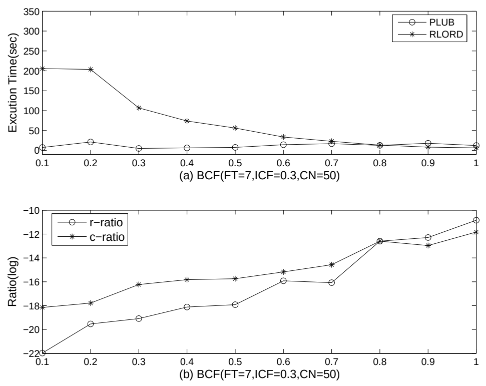

Figure 8: The Effect of the Between-Clusters Compactness Factor

5.4 The Effect of the Between-Cluster Compactness Factor

In this section, we show the effect of the between-cluster compactness factor (BCF) on the performance of PLUB and RLORD. We set the feature type number at 7, ICF at 0.3 and cluster number at 50, which is medium density in our experiments. We raised parameter BCF from 0.1 to its highest value 1.0.

Figure 8(a) illustrates the performance of PLUB and RLORD on data sets with different BCF. We can see that both the execution times and the trends of PLUB and RLORD are very different. The execution time of RLORD has an apparent down trend with the increase of BCF from 0.1 to 1.0. However, the execution time of PLUB doesn't change too much. With BCF values smaller than some value, about 0.8 in this specific experimental setting, PLUB runs faster. When BCF increases beyond this value, RLORD is faster.

Figure 8(b) gives the results of formula 1. The curve $r-ratio$ shows the ratio of the left side of formula 1 and the curve $c-ratio$ presents the ratio of the right side of formula 1. Both ratio values are log values because they are tiny numbers, which means the pruning ability is very high. A seemingly contradictory result evident in this figure is that increases in the $r-ratio$, which means there is a decrease in the pruning ratio, does not lead to increases in execution time. The explanation is that when BCF increases, there are fewer leaf nodes that intersected with the current search bound. Thus the total number of possible candidate leaf node sequences decreases dramatically, thereby reducing the execution time. The key point to note here is that when the $r-ratio$ is smaller than the $c-ratio$, PLUB runs faster but when the remaining ratio is greater than the comparison ratio, PLUB takes more time than RLORD. In other words, the relative trends of $r-ratio$ and $c-ratio$ only determine the relative execution time of PLUB and RLORD.

5.5 The Effect of the In-Cluster Compactness Factor

In this section, we show the effect of the in-cluster compactness factor (ICF) on the performance of PLUB and RLORD. We set the feature type number at 7, BCF at 0.1 and cluster number at 50, or medium density. We changed parameter ICF from 0.1 to 0.5.

Figure 9 illustrates the performance of PLUB and RLORD on data sets with different ICF. We can see the execution

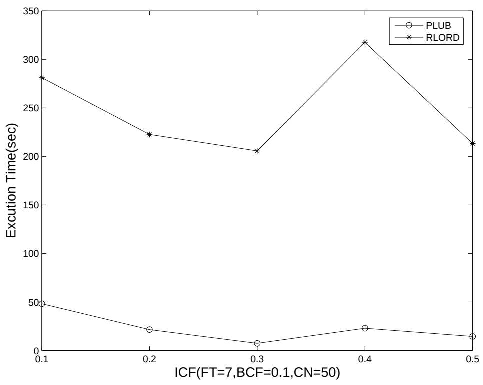

Figure 9: The Effect of the In-Cluster Compactness Factor

times of PLUB and RLORD are very different. With $BCF = 0.1$, ICF has little influence on the execution time of either PLUB or RLORD, which means if the minimum allowed distance $minCCDist$ of clusters is very small, compared to the maximum allowed distance $maxCCDist$, in our experimental settings, the effect of BCF is dominant among all factors. From this figure the only apparent trend is that PLUB always runs much faster than RLORD under these experimental settings.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated a multi-type nearest neighbor (MTNN) query problem which can be applied to many application domains. We proposed an R-tree based solution to the MTNN query. In our algorithm, a page-level upper bound (PLUB) is exploited for efficient pruning at the R-tree node level. Finally, experimental results are provided to show the strength of the proposed algorithm and design decisions related to performance tuning. In our experiments, we compare the performances of PLUB and RLORD in terms of execution time. When data sets are compact, PLUB outperforms RLORD. When data sets go to random-distributed in space, RLORD runs faster than PLUB.

As for future work, we plan to evaluate the I/O cost of PLUB and do further experiments using real data sets. In addition, we believe that the PLUB algorithm has potential for applications related to real road networks.

7. REFERENCES

[1] S. Berchtold, C. Bohm, D. Keim, and H. Kriegel. A cost model for nearest neighbor in high-dimensional data space. 1997.

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

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

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

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

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

[7] A. Corral, M. Vassilakopoulos, 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, 2001.

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

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

[10] H. Herhatosmanoglu, D. A. 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] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *ACM SIGMOD*, 2000.

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

[15] F. Li, D. Chen, M. Hadjieleftherious, G. Kollios, and S. Teng. On trip planning queries in spatial databases. In *SSTD*, 2005.

[16] X. Ma, S. Shekhar, H. Xiong, and P. Zhang. Exploiting a page-level upper bound for multi-type nearest neighbor queries. In *Technique Report,05-008, University of Minnesota*, 2005.

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

[18] A. Papadopoulos and Y. Manolopoulos. Performance of Nearest Neighbor Queries in R-trees. In *ICDT*, pages 394–408, 1997.

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

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

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

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

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

[24] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *VLDB*, 2002.

[25] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao. All-Nearest-Neighbors Queries in Spatial Databases. In *SSDBM*, 2004.