On Trip Planning Queries in Spatial Databases

Feifei Li, Dihan Cheng, Marios Hadjieleftheriou,
George Kollios, and Shang-Hua Teng

Computer Science Department
Boston University
{lifeifei, dcheng, marioh, gkollios, steng}@cs.bu.edu

Abstract. In this paper we discuss a new type of query in Spatial Databases, called the Trip Planning Query (TPQ). Given a set of points of interest P in space, where each point belongs to a specific category, a starting point S and a destination E, TPQ retrieves the *best* trip that starts at S, passes through at least one point from each category, and ends at E. For example, a driver traveling from Boston to Providence might want to stop to a gas station, a bank and a post office on his way, and the goal is to provide him with the best possible route (in terms of distance, traffic, road conditions, etc.). The difficulty of this query lies in the existence of multiple choices per category. In this paper, we study fast approximation algorithms for TPQ in a metric space. We provide a number of approximation algorithms with approximation ratios that depend on either the number of categories, the maximum number of points per category or both. Therefore, for different instances of the problem, we can choose the algorithm with the best approximation ratio, since they all run in polynomial time. Furthermore, we use some of the proposed algorithms to derive efficient heuristics for large datasets stored in external memory. Finally, we give an experimental evaluation of the proposed algorithms using both synthetic and real datasets.

1 Introduction

Spatial databases has been an active area of research in the last two decades and many important results in data modeling, spatial indexing, and query processing techniques have been reported [29, 17, 40, 37, 42, 26, 36, 4, 18, 27]. Despite these efforts, the queries that have been considered so far concentrate on simple range and nearest neighbor queries and their variants. However, with the increasing interest in intelligent transportation and modern spatial database systems, more complex and advanced query types need to be supported.

In this paper we discuss a novel query in spatial databases, the Trip Planning Query (TPQ). Assume that a database stores the locations of spatial objects that belong to one or more categories from a fi xed set of categories $\mathcal{C}$. The user specifi es two points in space, a starting point S and a destination point E, and a subset of categories $\mathcal{R}$,

This work was partially supported by NSF grants IIS-0133825, IIS-0308213, CCR-0311430, and ITR CCR-0325630.

($\mathcal{R} \subseteq \mathcal{C}$), and the goal is to find the *best* trip (route) that starts at S, passes through at least one point from each category in $\mathcal{R}$ and ends at E. An example of a TPQ is the following: A user plans to travel from Boston to Providence and wants to stop at a supermarket, a bank, and a post office. Given this query, a database that stores the locations of objects from the categories above (as well as other categories) should compute efficiently a feasible trip that minimizes the total traveling distance. Another possibility is to provide a trip that minimizes the total traveling time.

Efficient TPQ evaluation could become an important new feature of advanced navigation systems and can prove useful for other geographic applications as has been advocated in previous work [12]. For instance, state of the art mapping services like MapQuest, Google Maps, and Microsoft Streets & Trips, currently support queries that specify a starting point and only one destination, or a number of user specified destinations. The functionality and usefulness of such systems can be greatly improved by supporting more advanced query types, like TPQ. An example from Streets & Trips is shown in Figure 1, where the user has explicitly chosen a route that includes an ATM, a gas station and a Greek restaurant. Clearly, the system could not only optimize this route by re-arranging the order in which these stops should be made, but it could also suggest alternatives, based on other options available (e.g., from a large number of ATMs that are shown on the map), that the user might not be aware of.

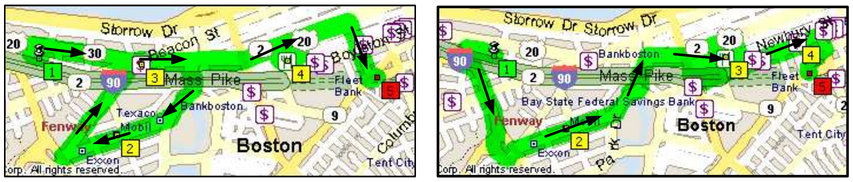


Fig. 1. A route from Boston University (1) to Boston downtown (5) that passes by a gas station (2), an ATM (3), and a Greek restaurant (4) that have been explicitly specified by the user in that order. Existing applications do not support route optimization, nor do they give suggestions of more suitable routes, like the one presented to the right.

TPQ can be considered as a generalization of the Traveling Salesman problem (TSP) [2, 1, 10] which is NP-hard. The reduction of TSP to TPQ is straightforward. By assuming that every point belongs to its own distinct category, any instance of TSP can be reduced to an instance of TPQ. TPQ is also closely related to the group minimum spanning/steiner tree problems [24, 20, 16], as we discuss later. From the current spatial database queries, TPQ is mostly related to *time parameterized* and *continuous* NN queries [5, 41, 36, 37], where we assume that the query point is moving with a constant velocity and the goal is to incrementally report the nearest neighbors over time as the query moves from an initial to a final location. However, none of the methods developed to answer the above queries can be used to find a good solution for TPQ.

Contributions. This paper proposes a novel type of query in spatial databases and studies methods for answering this query effi ciently. Approximation algorithms that achieve various approximation ratios are presented, based on two important parameters: The total number of categories m and the maximum category cardinality ρ. In particular:

– We introduce four algorithms for answering TPQ queries, with various approximation ratios in terms of m and ρ. We give two practical, easy to implement solutions better suited for external memory datasets, and two more theoretical in nature algorithms that give tighter answers, better suited for main memory evaluation.
– We present various adaptations of these algorithms for practical scenarios, where we exploit existing spatial index structures and transportation graphs to answer TPQs.
– We perform an extensive experimental evaluation of the proposed techniques on real transportation networks and points of interest, as well as on synthetic datasets for completeness.

In parallel and independently with our work, Sharifzadeh et al. [31], addressed a similar query called the Optimal Sequenced Route (OSR) Query. The main difference between the TPQ and the OSR query is that in the latter, the user has to specify the order of the groups that must be visited.

2 Preliminaries

This section defi nes formally the general TPQ problem and introduces the basic notation that will be used in the rest of the paper. Furthermore, a concise overview of related work is presented.

2.1 Problem Formulation

We consider solutions for the TPQ problem on *metric graphs*. Given a connected graph $G(\mathcal{V}, \mathcal{E})$ with n vertices $\mathcal{V} = \{v_1, \dots, v_n\}$ and s edges $\mathcal{E} = \{e_1, \dots, e_s\}$, we denote the cost of traversing a path $v_i, \dots, v_j$ with $c(v_i, \dots, v_j) \geq 0$.

Definition 1. *G is a metric graph if it satisfies the following conditions:*
1. *$c(v_i, v_j) = 0$ iff $v_i = v_j$*
2. *$c(v_i, v_j) = c(v_j, v_i)$*
3. *The triangle inequality $c(v_i, v_k) + c(v_k, v_j) \geq c(v_i, v_j)$*

Given a set of m categories $\mathcal{C} = \{C_1, \dots, C_m\}$ (where $m \leq n$) and a mapping function $\pi : v_i \longrightarrow C_j$ that maps each vertex $v_i \in \mathcal{V}$ to a category $C_j \in \mathcal{C}$, the TPQ problem can be defi ned as follows:

Definition 2. *Given a set $\mathcal{R} \subseteq \mathcal{C}$ ($\mathcal{R} = \{R_1, R_2, \dots, R_k\}$), a starting vertex S and an ending vertex E, identify the vertex traversal $\mathcal{T} = \{S, v_{t_1}, \dots, v_{t_k}, E\}$ (also called a trip) from S to E that visits at least one vertex from each category in $\mathcal{R}$ (i.e., $\cup_{i=1}^{k} \pi(v_{t_i}) = \mathcal{R}$) and has the minimum possible cost $c(\mathcal{T})$ (i.e., for any other feasible trip $\mathcal{T}'$ satisfying the condition above, $c(\mathcal{T}) \leq c(\mathcal{T}')$).*

In the rest, the total number of vertices is denoted by n, the total number of categories by m, and the maximum cardinality of any category by ρ. For ease of exposition, it will be assumed that $\mathcal{R} = \mathcal{C}$, thus $k = m$. Generalizations for $\mathcal{R} \subset \mathcal{C}$ are straightforward (as will be discussed shortly).

2.2 Related Work

In the context of spatial databases, the TPQ problem has not been addressed before. Most research has concentrated on traditional spatial queries and their variants, namely range queries [18], nearest neighbors [15, 19, 29], continuous nearest neighbors [5, 37, 41], group nearest neighbors [26], reverse nearest neighbors [22], etc. All these queries are fundamentally different from TPQ since they do not consider the computation of optimal paths connecting a starting and an ending point, given a graph and intermediate points.

Research in spatial databases also addresses applications in spatial networks represented by graphs, instead of the traditional Euclidean space. Recent papers that extend various types of queries to spatial networks are [27, 21, 30]. Most of the solutions therein are based on traditional graph algorithms [10, 23]. Clustering in a road network database has been studied in [43], where a very efficient data structure was proposed based on the ideas of [32]. Likewise, here we study the TPQ problem on road networks, as well.

The Traveling Salesman Problem (TSP) has received a lot of attention in the last thirty years. A simple polynomial time 2-approximation algorithm for TSP on a metric graph can be obtained using the Minimum Spanning Tree (MST) [10]. The best constant approximation ratio for metric TSP is the $\frac{3}{2}$-approximation that can be derived by the Christofides algorithm [9]. Recently, a polynomial time approximation scheme (PTAS) for Euclidean TSP has been proposed by Arora [1]. For any fixed $\varepsilon > 0$ and any n nodes in $\mathbb{R}^2$ the randomized version of the scheme can achieve a $(1+\varepsilon)$-approximation in $O(n \log^{O(\frac{1}{\varepsilon}} n)$ running time. Unfortunately, it seems that the TPQ does not admit a PTAS. Furthermore, there are many approximation algorithms for variations of the TSP problem, e.g., TSP with neighborhoods [11]. Nevertheless, the solutions to these problems cannot be applied directly to TPQ, since the problems are fundamentally different. For more approximation algorithms for different versions of TSP, we refer to [2] and the references therein. Finally, there are many practical heuristics for TSP [33], e.g., genetic and greedy algorithms, that work well for some practical instances of the problem, but no approximation bounds are known about them.

TPQ is also closely related to the Generalized Minimum Spanning Tree (GMST) problem. The GMST is a generalized version of the MST problem where the vertices in a graph G belong to m different categories. A tree T is a GMST of G if T contains at least one vertex from each category and T has the minimum possible cost (total weight or total length). Even though the MST problem is in P, it is known that the GMST is in NP. There are a few methods from the operational research and economics community that propose heuristics for solving this problem [24] without providing a detailed analysis on the approximation bounds. The GMST problem is a special instance of an even harder problem, the Group Steiner Tree (GST) problem [16, 20]. For example, polylogarithmic approximation algorithms have been proposed recently [14, 13]. Since the GMST problem is a special instance of the GST problem, such bounds apply to GMST as well.

3 Fast Approximation Algorithms

In this section we examine several approximation algorithms for answering the trip planning query in main memory. For each solution we provide the approximation ratios in terms of m and ρ. For simplicity, consider that we are given a complete graph G^c, containing one edge per vertex pair v_i, v_j ($1 \leq i, j \leq n$) representing the cost of the shortest path from v_i to v_j in the original graph G. Let $\mathcal{T}_k = \{v_{t_0}, v_{t_1}, \ldots, v_{t_k}\}$ denote the partial trip that has visited k vertices, excluding S (where $S = v_{t_0}$). Trivially, it can be shown that a trip $\mathcal{T}_k$ constructed on the induced graph G^c, has exactly the same cost as in graph G, with the only difference being that a number of vertices visited on the path from a given vertex to another are hidden. Hiding irrelevant vertices by using the induced graph G^c guarantees that any trip $\mathcal{T}$ produced by a given algorithm will be represented by exactly m significant vertices, which will simplify exposition substantially in what follows. In addition, by removing from graph G^c all vertices that do not belong to any of the m categories in R, we can reduce the size of the graph and simplify the construction of the algorithms. Given a solution obtained using the reduced graph and the complete shortest path information for graph G^c, the original trip on graph G can always be acquired. In the following discussion, $\mathcal{T}_a^P$ denotes an approximation trip for problem P, while $\mathcal{T}_o^P$ denotes the optimal trip. When P is clear from context the superscript is dropped. Furthermore, due to lack of space the proofs for all theorems appear in the full version of this paper.

3.1 Approximation in Terms of m

In this section we provide two greedy algorithms with tight approximation ratios with respect to m.

Nearest Neighbor Algorithm The most intuitive algorithm for solving TPQ is to form a trip by iteratively visiting the nearest neighbor of the last vertex added to the trip from all vertices in the categories that have not been visited yet, starting from S. Formally, given a partial trip $\mathcal{T}_k$ with $k < m$, $\mathcal{T}_{k+1}$ is obtained by inserting the vertex $v_{t_{k+1}}$ which is the nearest neighbor of v_{t_k} from the set of vertices in $\mathcal{R}$ belonging to categories that have not been covered yet. In the end, the final trip is produced by connecting v_{t_m} to E. We call this algorithm $\mathcal{A}_{NN}$, which is shown in Algorithm 1.

Algorithm 1 $\mathcal{A}_{NN}(G^c, \mathcal{R}, S, E)$

```
1: $v = S, I = \{1, \ldots, m\}, \mathcal{T}_a = \{S\}$
2: for $k = 1$ to $m$ do
3:     $v =$ the nearest $NN(v, R_i)$ for all $i \in I$
4:     $\mathcal{T}_a \leftarrow \{v\}$
5:     $I \leftarrow I - \{i\}$
6: end for
7: $\mathcal{T}_a \leftarrow \{E\}$
```

Theorem 1. *$\mathcal{A}_{NN}$ gives a $(2^{m+1} - 1)$-approximation (with respect to the optimal solution). In addition, this approximation bound is tight.*

Minimum Distance Algorithm This section introduces a novel greedy algorithm, called $\mathcal{A}_{MD}$, that achieves a much better approximation bound, in comparison with the previous algorithm. The algorithm chooses a set of vertices $\{v_1, \dots, v_m\}$, one vertex per category in $\mathcal{R}$, such that the sum of costs $c(S, v_i) + c(v_i, E)$ per v_i is the minimum cost among all vertices belonging to the respective category R_i (i.e., this is the vertex from category R_i with the minimum traveling distance from S to E). After the set of vertices has been discovered, the algorithm creates a trip from S to E by traversing these vertices in nearest neighbor order, i.e., by visiting the nearest neighbor of the last vertex added to the trip, starting with S. The algorithm is shown in Algorithm 2.

Algorithm 2 $\mathcal{A}_{MD}(G^c, \mathcal{R}, S, E)$

```
 1: $U = \emptyset$
 2: for $i = 1$ to $m$ do
 3:     $U \leftarrow \pi(v) = R_i : c(S, v) + c(v, E)$ is minimized
 4: $v = S, \mathcal{T}_a \leftarrow \{S\}$
 5: while $U \neq \emptyset$ do
 6:     $v = NN(v, U)$
 7:     $\mathcal{T}_a \leftarrow \{v\}$
 8:     Remove $v$ from $U$
 9: end while
10: $\mathcal{T}_a \leftarrow \{E\}$
```

Theorem 2. *If m is odd (even) then $\mathcal{A}_{MD}$ gives an m-approximate (m+1-approximate) solution. In addition this approximation bound is tight.*

3.2 Approximation in Terms of ρ

In this section we consider an Integer Linear Programming approach for the TPQ problem which achieves a linear approximation bound w.r.t. ρ, i.e., the maximum category cardinality. Consider an alternative formulation of the TPQ problem with the constraint that $S = E$ and denote this problem as Loop Trip Planning Query(LTPQ) problem. Next we show how to obtain a $\frac{3}{2}\rho$-approximation for LTPQ using Integer Linear Programming.

Let $A = (a_{ji})$ be the $m \times (n+1)$ incidence matrix of G, where rows correspond to the m categories, and columns represent the $n+1$ vertices (including $v_0 = S = E$). A's elements are arranged such that $a_{ji} = 1$ if $\pi(v_i) = R_j$, $a_{ji} = 0$ otherwise. Clearly, $\rho = max_j \sum_i a_{ji}$, i.e., each category contains at most ρ distinct vertices. Let indicator variable $y(v) = 1$ if vertex v is in a given trip and 0 otherwise. Similarly, let $x(e) = 1$ if the edge e is in a given trip and 0 otherwise. For any $\mathcal{S} \subset \mathcal{V}$, let $\delta(\mathcal{S})$ be the edges contained in the cut $(\mathcal{S}, \mathcal{V} \setminus \mathcal{S})$. The integer programming formulation for the LTPQ problem is the following:

Problem $IP_{LTPQ} =$ minimize $\sum_{e \in \mathcal{E}} c(e)x(e)$, subject to:

1. $\sum_{e \in \delta(\{v\})} x(e) = 2y(v)$, for all $v \in \mathcal{V}$,
2. $\sum_{e \in \delta(\mathcal{S})} x(e) \geq 2y(v)$, for all $\mathcal{S} \subset \mathcal{V}, v_0 \notin \mathcal{S}$, and all $v \in \mathcal{S}$,
3. $\sum_{i=1}^{n} a_{ji} y(v_i) \geq 1$, for all $j = 1, \ldots, m$,
4. $y(v_0) = 1$,
5. $y(v_i) \in \{0, 1\}, x(e_i) \in \{0, 1\}$

Condition 1 guarantees that for every vertex in the trip there are exactly two edges incident on it. Condition 2 prevents subtrips, that is the trip cannot consist of two disjoint subtrips. Condition 3 guarantees that the chosen vertices cover all categories in $\mathcal{R}$. Condition 4 guarantees that v_0 is in the trip. In order to simplify the problem we can relax the above Integer Programming into LP_{LTPQ} by relaxing Conditions 5 to: $0 \leq y(v), x(e) \leq 1$. Any effi cient algorithm for solving Linear Programming could now be applied to solve LP_{LTPQ} [34]. In order to get a feasible solution for IP_{LTPQ}, we apply the randomized rounding scheme stated below:

Randomized Rounding: For solutions obtained by LP_{LTPQ}, set $y(v_i) = 1$ if $y(v_i) \geq \frac{1}{\rho}$. If the trip visits vertices from the same category more than once, randomly select one to keep in the trip and set $y(v_j) = 0$ for the rest.

Theorem 3. *LP_{LTPQ} together with the randomized rounding scheme above finds a $\frac{3}{2}\rho$-approximation for IP_{LTPQ}, i.e., the integer programming approach is able to find a $\frac{3}{2}\rho$-approximation for the LTPQ problem.*

We denote any algorithm for LTPQ as $\mathcal{A}_{LTPQ}$. A TPQ problem can be converted into an LTPQ problem by creating a special category $C_{m+1} = E$. The solution from this converted LTPQ problem is guaranteed to pass through E. Using the result returned by $\mathcal{A}_{LTPQ}$, a trip with constant distortion could be obtained for TPQ:

Lemma 1. *A β-approximation algorithm for LTPQ implies a 3β-approximation algorithm for TPQ.*

Therefore, by combining Theorem 3 and Lemma 1:

Lemma 2. *There is a polynomial time algorithm based on Integer Linear Programming for the TPQ problem with a $\frac{9}{2}\rho$-approximation.*

3.3 Approximation in Terms of m and ρ

In Section 2 we discussed the Generalized Minimum Spanning Tree (GMST) problem which is closely related to the TPQ problem. Recall that the TSP problem is closely related to the Minimum Spanning Tree (MST) problem, where a 2-approximation algorithm can be obtained for TSP based on MST. In similar fashion, it is expected that one can obtain an approximate algorithm for TPQ problem, based on an approximation algorithm for GMST problem.

Unlike the MST problem which is in P, GMST problem is in NP. Suppose we are given an approximation algorithm for GMST problem, denoted $\mathcal{A}_{GMST}$. We can construct an approximation algorithm for TPQ problem as shown in Algorithm 3.

Algorithm 3 APPROXIMATION ALGORITHM FOR TPQ BASED ON GMST

```
1: Compute a $\beta$-approximation $Tree_a^{GMST}$ for $G$ rooted at $S$ using $\mathcal{A}_{GMST}$.
2: Let $LT$ be the list of vertices visited in a pre-order tree walk of $Tree_a^{GMST}$.
3: Move $E$ to the end of $LT$.
4: Return $\mathcal{T}_a^{TPQ}$ as the ordered list of vertices in $LT$.
```

Lemma 3. *If we use a β-approximation algorithm for GMST problem, then Algorithm 3 for TPQ problem is a 2β-approximation algorithm.*

We can get a solution for TPQ by using Lemma 3 and any known approximation algorithm for GST, as GMST is a special instance of GST. For example, the $O(\log^2 \rho \log m)$ algorithm proposed in [14], which yields a solution to TPQ with the same complexity.

4 Algorithm Implementations in Spatial Databases

In this section we discuss implementation issues of the proposed TPQ algorithms from a practical perspective, given disk resident datasets and appropriate index structures. We show how the index structures can be utilized to our benefit, for evaluating TPQs efficiently. We opt at providing design details only for the greedy algorithms, $\mathcal{A}_{NN}$ and $\mathcal{A}_{MD}$ since they are simpler to implement in external memory, while the Integer Linear Programming and GMST approaches are more appropriate for main memory and are not easily applicable to external memory datasets.

4.1 Applications in Euclidean Space

First, we consider TPQs in a Euclidean space where a spatial dataset is indexed using an R-tree [18]. We show how to adapt $\mathcal{A}_{NN}$ and $\mathcal{A}_{MD}$ in this scenario. For simplicity, we analyze the case where a single R-tree stores spatial data from all categories.

Implementation of $\mathcal{A}_{NN}$. The implementation of $\mathcal{A}_{NN}$ using an R-tree is straightforward. Suppose a partial trip $\mathcal{T}_k = \{S, p_1, \ldots, p_k\}$ has already been constructed and let $\mathcal{C}(\mathcal{T}_k) = \cup_{i=1}^{k} \pi(p_i)$, denote the categories visited by $\mathcal{T}_k$. By performing a nearest neighbor query with origin p_k, using any well known NN algorithm, until a new point p_{k+1} is found, such that $\pi(p_{k+1}) \notin C(\mathcal{T}_k)$, we iteratively extend the trip one vertex at a time. After all categories in $\mathcal{R}$ have been covered, we connect the last vertex to E and the complete trip is returned. The main advantage of $\mathcal{A}_{NN}$ is its efficiency. Nearest neighbor query in R-tree has been well studied. One could expect very fast query performance for $\mathcal{A}_{NN}$. However, the main disadvantage of $\mathcal{A}_{NN}$ is the problem of "searching without directions". Consider the example shown in Figure 2. $\mathcal{A}_{NN}$ will find the trip $T1 = \{S \rightarrow A1 \rightarrow B1 \rightarrow C1 \rightarrow E\}$ instead of the optimal trip $T2 = \{S \rightarrow C2 \rightarrow A2 \rightarrow B2 \rightarrow E\}$. In $\mathcal{A}_{NN}$, the search in every step greedily expands the point that is closest to the last point in the partial trip without considering the end destination, i.e., without considering the direction. The more intuitive approach is to limit the search within a vicinity area defined by S and E. The next algorithm addresses this problem.

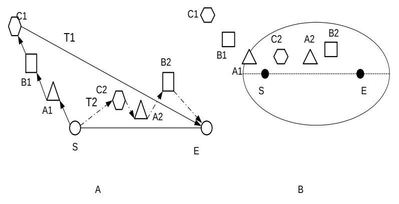


Fig. 2. Intuition of vicinity area

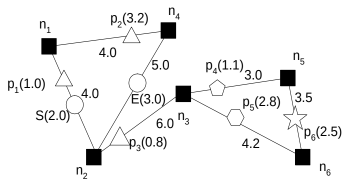


Fig. 3. A simple road network.

Implementation of $\mathcal{A}_{MD}$. Next, we show how to implement $\mathcal{A}_{MD}$ using an R-tree. The main idea is to locate the m points, one from each category in $\mathcal{R}$, that minimize the Euclidean distance $\mathcal{D}(S, E, p) = c(S, p) + c(p, E)$ from S to E through p. We call this the minimum distance query. This query meets our intuition that the trip planning query should be limited within the vicinity area of the line segment defined by S, E (as in the example in Figure 2). The minimum distance query can be answered by modifying the NN search algorithm for R-trees [29], where instead of using the traditional $MinDist$ measure for sorting candidate distances, we use $\mathcal{D}$. In that case, the vicinity area is an ellipse and not a circle (Figure 2). Given S and E we run the modified NN search once for locating all m points incrementally, and report the final trip.

All NN algorithms based on R-trees compute the nearest neighbors incrementally using the tree structure to guide the search. An interesting problem that arises in this case is how to geometrically compute the minimum possible distance $\mathcal{D}(S, E, p)$ between points S, E and any point p inside a given MBR M (similar to the $MinDist$ heuristic of the traditional search). This problem can be reduced to that of finding the point p on line segment AB (where AB is a boundary of M) that minimizes $\mathcal{D}(S, E, p)$, which can then be used to find the minimum distance from M, by applying it on the MBR boundaries lying closer to line segment SE. Point p can be computed by projecting the mirror image E' of E, given AB. It can be proved that:

Lemma 4. *Given line segments AB and SE, the point p that minimizes $\mathcal{D}(S, E, p)$ is:*
Case A: If EE' intersects AB, then p is the intersection of AB and SE'.
Case B: If EE' and SE do not intersect AB, then p is either A or B.
Case C: If SE intersects AB, then p is the intersection of SE and AB.

Using the lemma, we can easily compute the minimum distances $\mathcal{D}(S, E, M)$ for appropriately sorting the R-tree MBRs during the NN search. The details of the minimum distance query algorithm is shown in Algorithm 4. For simplicity, here we show the algorithm that searches for a point from one particular category only, which can easily be extended for multiple categories. In line 8 of the algorithm, if c is a node then $\mathcal{D}(S, E, c)$ is calculated by applying Lemma 4 with line segments from the borders of the MBR of c; if c is a point then $\mathcal{D}(S, E, c)$ is the length $|Sc| + |cE|$. Straightforwardly, the algorithm can also be modified for returning the top k points.

4.2 Applications in Road Networks

An interesting application of TPQs is on road network databases. Given a graph $\mathcal{N}$ representing a road network and a separate set $\mathcal{P}$ representing points of interest (gas

Algorithm 4 ALGORITHM MINIMUM DISTANCE QUERY FOR R-TREES

```
Require: Points $S$, $E$, Category $R_i$, R-tree rtree
 1: PriorityQueue $QR = \emptyset$, $QS = \{(rtree.root, 0)\}$; $B = \infty$
 2: while $QS$ not empty do
 3:     $n = QS.top$;
 4:     if $n.dist \geq B$ then
 5:         return $QR.top$
 6:     for all children $c$ of $n$ do
 7:         $dist = \mathcal{D}(S, E, c)$
 8:         if $n$ is an index node then
 9:             $QS \leftarrow (c, dist)$
10:         else if $\pi(M) = R_i$ then                    ▷ ($c$ is a point)
11:             $QR \leftarrow (c, dist)$
12:             if $dist \leq B$ then $B = dist$
```

stations, hotels, restaurants, etc.) located at fi xed coordinates on the edges of the graph, we would like to develop appropriate index structures in order to answer effi ciently trip planning queries for visiting points of interest in $\mathcal{P}$ using the underlying network $\mathcal{N}$. Figure 3 shows an example road network, along with various points of interest belonging to four different categories.

For our purposes we represent the road network using techniques from [32, 43, 27]. In summary, the adjacency list of $\mathcal{N}$ and set $\mathcal{P}$ are stored as two separate flat fi les indexed by B^+-trees. For that purpose, the location of any point $p \in \mathcal{P}$ is represented as an offset from the road network node with the smallest identifi er that is incident on the edge containing p. For example, point p_4 is 1.1 units away from node n_3.

Implementation of $\mathcal{A}_{NN}$. Nearest neighbor queries on road networks have been studied in [27], where a simple extension of the well known Dijkstra algorithm [10] for the single-source shortest-path problem on weighted graphs is utilized to locate the nearest point of interest to a given query point. As with the R-tree case, straightforwardly, we can utilize the algorithm of [27] to incrementally locate the nearest neighbor of the last stop added to the trip, that belongs to a category that has not been visited yet. The algorithm starts from point S and when at least one stop from each category has been added to the trip, the shortest path from the last discovered stop to E is computed.

Implementation of $\mathcal{A}_{MD}$. Similarly to the R-tree approach, the idea is to fi rst locate the m points from categories in $\mathcal{R}$ that minimize the network distance $c(S, p_i, E)$ using the underlying graph $\mathcal{N}$, and then create a trip that traverses all p_i in a nearest neighbor order, from S to E. It is easy to show with a counter example that simply fi nding a point p that fi rst minimizes cost $c(S, p)$ and then traverses the shortest path from p to E, does not necessarily minimize cost $c(S, p, E)$. Thus, Dijkstra's algorithm cannot be directly applied to solve this problem. Alternatively, we propose an algorithm for identifying such points of interest. The procedure is shown in Algorithm 5.

The algorithm locates a point of interest $p : \pi(p) \in R_i$ (given R_i) such that the distance $c(S, p, E)$ is minimized. The search begins from S and incrementally expands all possible paths from S to E through all points p. Whenever such a path is computed and all other partial trips have cost smaller than the tentative best cost, the search stops. The key idea of the algorithm is to separate partial trips into two categories: one that contains only paths that have not discovered a point of interest yet, and one that contains paths that have. Paths in the first category compete to find the shortest possible route from S to any p. Paths in the second category compete to find the shortest path from their respective p to E. The overall best path is the one that minimizes the sum of both costs.

Algorithm 5 ALGORITHM Minimum Distance Query FOR ROAD NETWORKS

```
Require: Graph $\mathcal{N}$, Points of interest $\mathcal{P}$, Points $S$, $E$, Category $R_i$
 1: For each $n_i \in \mathcal{N} : n_i.c_p = n_i.c_{\neg p} = \infty$
 2: PriorityQueue $PQ = \{S\}$, $B = \infty$, $\mathcal{T}_B = \emptyset$
 3: while $PQ$ not empty do
 4:     $\mathcal{T} = PQ.top$
 5:     if $\mathcal{T}.c \geq B$ then return $\mathcal{T}_B$
 6:     for each node $n$ adjacent to $\mathcal{T}.last$ do
 7:         $\mathcal{T}' = \mathcal{T}$                                   ▷ (create a copy)
 8:         if $\mathcal{T}'$ does not contain a $p$ then
 9:             if $\exists p : p \in \mathcal{P}, \pi(p) = R_i$ on edge $(\mathcal{T}'.last, n)$ then
10:                 $\mathcal{T}'.c+ = c(\mathcal{T}'.last, p)$
11:                 $\mathcal{T}' \leftarrow p$, $PQ \leftarrow \mathcal{T}'$
12:             else
13:                 $\mathcal{T}'.c+ = c(\mathcal{T}'.last, n)$, $\mathcal{T}' \leftarrow n$
14:                 if $n.c_{\neg p} > \mathcal{T}'.c$ then
15:                     $n.c_{\neg p} = \mathcal{T}'.c$, $PQ \leftarrow \mathcal{T}'$
16:         else
17:             if edge $(\mathcal{T}', n)$ contains $E$ then
18:                 $\mathcal{T}'.c+ = c(\mathcal{T}'.last, E)$, $\mathcal{T}' \leftarrow E$
19:                 Update $B$ and $\mathcal{T}_B$ accordingly
20:             else
21:                 $\mathcal{T}'.c+ = c(\mathcal{T}'.last, n)$, $\mathcal{T}' \leftarrow n$
22:                 if $n.c_p > \mathcal{T}'.c$ then
23:                     $n.c_p = \mathcal{T}'.c$, $PQ \leftarrow \mathcal{T}'$
24:         endif
25:     endfor
26: endwhile
```

The algorithm proceeds greedily by expanding at every step the trip with the smallest current cost. Furthermore, in order to be able to prune trips that are not promising, based on already discovered trips, the algorithm maintains two partial best costs per node $n \in \mathcal{N}$. Cost $n.c_p$ ($n.c_{\neg p}$) represents the partial cost of the best trip that passes through this node and that has (has not) discovered an interesting point yet. After all k points(one from each category $R_i \in \mathcal{R}$) have been discovered by iteratively calling this algorithm, an approximate trip for TPQ can be produced. It is also possible to design an incremental algorithm that discovers all points from categories in $\mathcal{R}$ concurrently.

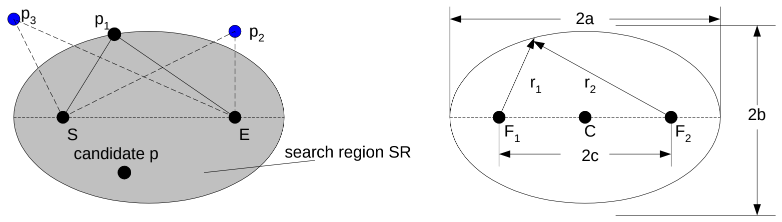


Fig. 4. The search region of a minimum distance query

5 Extensions

5.1 I/O Analysis for the Minimum Distance Query

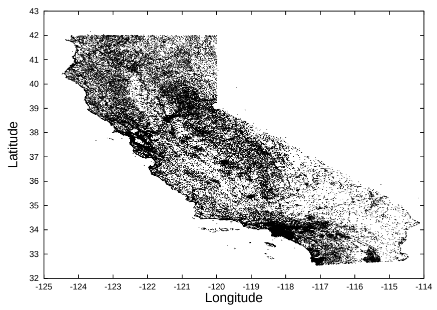
In this section we study the I/O bounds for the minimum distance query in Euclidean space, i.e., the expected number of I/Os when we try to find the point p that minimizes $\mathcal{D}(S, E, p)$ from a point set indexed with an R-tree. By carefully examining Algorithm 4 and Lemma 4, we can claim the following:

Claim. The lower bound of I/Os for minimum distance queries is the number of MBRs that intersect with line segment SE.

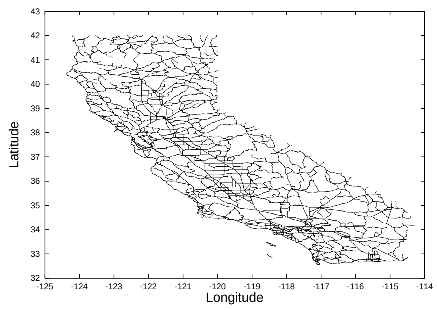
For the average case, the classical cost models for nearest neighbor queries can be used [39, 7, 6, 28, 38]. On average the I/O for any type of queries on R-trees is given by the expected node access: $NA = \sum_{i=0}^{h-1} n_i P_{NA_i}$ where h is the height of the tree, n_i is the number of nodes in level i and P_{NA_i} is the probability that a node at level i is accessed. The only peculiarity of minimum distance queries is that their search region *SR*, i.e., the area of the data space that may contain candidate results, forms an ellipse with focii the points S, E. It follows immediately that, on average, in order to answer a minimum distance query we have to visit all MBRs that intersect with its respective *SR*. Thus, if we quantify the size of *SR* we can estimate P_{NA_i}.

Consider the example in Figure 4, and suppose p_1 is currently the point that minimizes $\mathcal{D}(S, E, p_1)$. Then the ellipse defined by S, E, p_1 will be the region that contains possible better candidates, e.g., p in this example. This is true due to the property of the ellipse that $r_1 + r_2 = 2a$, i.e., any point p' on the border of the ellipse satisfies $\mathcal{D}(S, E, p') = 2a$. Therefore, to estimate the I/O cost of the query all we need to do is estimate quantity a. Assuming uniformity and a unit square universe, we have $Area_{SR} = k/|P|$. We also know that $Area_{SR} = Area_{ellipse} = 2\pi/\sqrt{4ac - b^2} = 2\pi/\sqrt{4ac - (a^2 - c^2)}$. Hence, $a = 2c + \sqrt{5c^2 - (\frac{2\pi|P|}{k})^2}$

With S, E, $c = |SE|/2$, and a, we could determine the search region for a k minimum distance query. With the search region being identified, one could derive the probability of any node of the R-tree being accessed. Then, the standard cost model analysis in [7, 6, 28, 38] can be straightforwardly be applied, hence the details are omitted. Generalizations for non-uniform distributions can also be addressed similarly to the analysis presented in [38], where few modifications are required given the ellipsoidal



(a) Collection of California's points of interests



(b) Road network of California(21048,22830)

Fig. 5. Real dataset from California

shape of the search regions. The I/O estimation for queries on road networks is much harder to analyze and heavily depends on the particular data structures used, therefore it is left as future work.

5.2 Hybrid Approach

We also consider a hybrid approach to the trip planning query for disk based datasets (in both Euclidean space and road networks). Instead of evaluating the queries using the proposed algorithms, the basic idea is to first select a sufficient number of good candidates from disk, and then process those in main memory. We apply the minimum distance query to locate the top k points from each respective category and then, assuming that the query visits a total of m categories, the $k \times m$ points are processed in main memory using any of the strategies discussed in Section 3. In addition, an exhaustive search is also possible. In this case, there are m^k number of instances to be checked. If m^k is large, a subset can be randomly selected for further processing, or the value of k is reduced. Clearly, the hybrid approach will find a solution at least as good as algorithm $\mathcal{A}_{MD}$. In particular, since the larger the value of k the closer the solution will be to the optimal answer, with a hybrid approach the user can tune the accuracy of the results, according to the cost she is willing to pay.

6 Experimental Evaluation

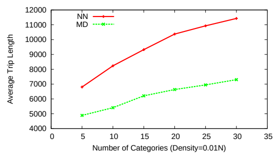
This section presents a comprehensive performance evaluation of the proposed techniques for TPQ in spatial databases. We used both synthetic datasets generated on real road networks and real datasets from the state of California. All experiments were run on a Linux machine with an Intel Pentium 4 2.0GHz CPU.

Experimental Setup. To generate synthetic datasets we obtained two real road networks, the city of Oldenburg(OL) with 6105 nodes and 7035 edges and San Joaquin county(TG) with 18263 nodes and 23874 edges, from [8]. For each dataset, we generated uniformly at random a number of points of interest on the edges of the network. Datasets with varying number of categories, as well as varying densities of points per category were generated. The total number of categories is in the range $m \in [5, 30]$, while the category density is in the range of $\rho \in [0.01N, 0.25N]$, where N is the total number of edges in the network. For Euclidean datasets, points of interest are generated using the road networks, but the distances are computed as direct Euclidean distances between points, without the network constraints. Our synthetic dataset has the flexibility of controlling different densities and number of categories, however it is based on uniform distribution on road network (not necessarily uniform in the Euclidean space). To study the general distribution of different categories, we also obtain a real dataset for our experiments. First we get a collection of points of interests that fall into different categories for the state of California from [35] as shown in Figure 5(a), then we obtain the road network for the same state from [25] as shown in Figure 5(b). Both of them represent the locations in a longitude/latitude space, which makes the merging step straightforward. The California dataset has 63 different categories, including airports, hospitals, bars, etc., and altogether more than $100,000$ points. Different categories exhibit very different densities and distributions. The road network in California has $21,048$ nodes and $22,830$ edges. For all experiments, we generate 100 queries with randomly chosen S and E.

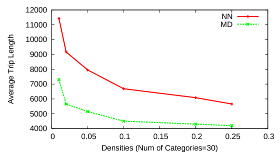
Road Network Datasets. In this part we study the performance of the two algorithms for road networks. First, we study the effects of m and ρ. Due to lack of space we present the results for the OL based datasets only. The results for the TG datasets were similar. Figure 6(a) plots the results for the average trip length as a function of m, for $\rho = 0.01N$. Figure 6(b) plots the average trip length as a function of ρ, for $m = 30$. In both cases, clearly $\mathcal{A}_{MD}$ outperforms $\mathcal{A}_{NN}$. In general, $\mathcal{A}_{MD}$ gives a trip that is 20%-40% better (in terms of trip length) than the one obtained from $\mathcal{A}_{NN}$. It is interesting to note that with the increase of m and the decrease of ρ the performance gap between the two algorithms increases. $\mathcal{A}_{NN}$ is greatly affected by the relative locations of points as it greedily follows the nearest point from the remaining categories irrespective of its direction with respect to the destination E. With the increase of m, the probability that $\mathcal{A}_{NN}$ wanders off the correct direction increases. With the decrease of ρ, the probability that the next nearest neighbor is close enough decreases, which in turn increases the chance that the algorithm will move far away from E. However, for both cases $\mathcal{A}_{MD}$ is not affected.

We also study the query cost of the two algorithms measured by the average running time of one query. Figure 7(a) plots the results as a function of density, and $m = 15$. In general, $\mathcal{A}_{NN}$ has smaller runtime. The reason is that the $\mathcal{A}_{MD}$ query in the road network is much more complex and needs to visit an increased number of nodes multiple times.

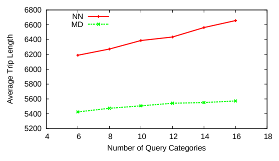
Euclidean Datasets. Due to lack of space we omit the plots for Euclidean datasets. In general, the results and conclusions were the same as for the road network datasets. A small difference is that the performance of the two algorithms is measured with respect



(a) Number of categories

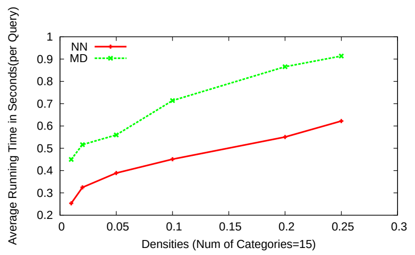


(b) Category Density

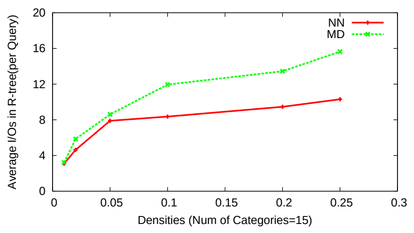


(c) General

Fig. 6. Average trip length of $\mathcal{A}_{NN}$ and $\mathcal{A}_{MD}$



(a) Runtime



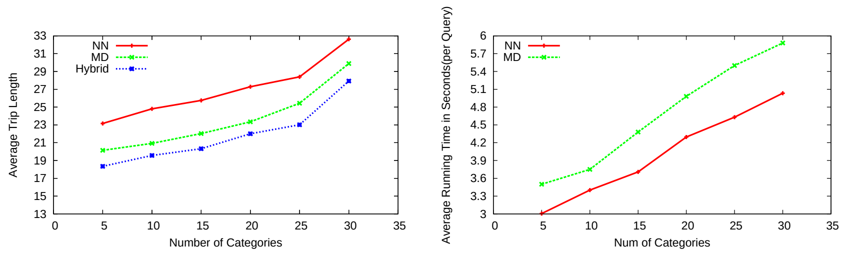
(b) I/O

Fig. 7. Query cost

to the total number of R-tree I/Os. In this case, $\mathcal{A}_{NN}$ was more effi cient than $\mathcal{A}_{MD}$, especially for higher densities as shown in Figure 7(b).

General Datasets and Query Workloads. In the previous experiments datasets had a fi xed density for all categories. Furthermore, queries had to visit all categories. Here, we examine a more general setting where the density for different categories is not fi xed and queries need to visit a subset $\mathcal{R}$ of all categories. Figure 6(c) summarizes the results. We set $m = 20$ and ρ uniformly distributed in $[0.01N, 0.20N]$. We experiment with subsets of varying cardinalities per query and measure the average trip length returned by both algorithms. $\mathcal{A}_{MD}$ outperforms $\mathcal{A}_{NN}$ by 15% in the worst case. With the increase of the cardinality of $\mathcal{R}$, the performance gain on $\mathcal{A}_{MD}$ increases.

Real Datasets. So far we have tested our algorithm on synthetic datasets To compare the performance of the algorithms in a real setting, we apply $\mathcal{A}_{NN}$ and $\mathcal{A}_{MD}$ on the real dataset from California. There are 63 different categories in this dataset, hence we show the query workload that requires visits to a subset of categories (up to 30 randomly selected categories). Figure 8(a) compares the average trip length obtained by $\mathcal{A}_{NN}$ and $\mathcal{A}_{MD}$ in the road network case. In this case, we simply use longitude and latitude as the point coordinates and calculate the distance based on that. So the absolute value for the distance is small. As we have noticed, $\mathcal{A}_{MD}$ still outperforms $\mathcal{A}_{NN}$ in terms of trip length, however, with the price of a higher query cost as indicated



(a) Road network (b) Running Time

Fig. 8. Experiments with real dataset

in Figure 8(b). Notice that the running time in this experiment is much higher than the one in Figure 7(a) as we are dealing with a much larger network as well as more data points. Similar results have been observed for the same dataset in Euclidean space (where the cost is measured in I/Os) and they are omitted. It is interesting to note that the trip length is increasing w.r.t. the number of categories in a non-linear fashion (e.g., from 25 categories to 30 categories), as compared to the same experiment on the synthetic dataset shown in Figure 6(a). This could be explained by the non-uniformity property and skewness of the real dataset. For example, there are more than 900 airports and only about 50 harbors. So when a query category for harbors is included, one expect to see a steep increase in the trip length.

Study of the Hybrid Approach. We also investigate the effectiveness of the hybrid approach as suggested in Section 5.2. Our experiments on synthetic datasets show that the hybrid approach improves results over $\mathcal{A}_{MD}$ by a small margin (Figure 8(a)). This is expected due to the uniformity of the underlying datasets. With the real dataset, as we can see in Figure 8(a), there is a noticeable improvement with the hybrid approach over $\mathcal{A}_{MD}$ (we set $m = 5$). This is mainly due to the skewed distribution in different categories in the real dataset. The hybrid approach incurs additional computational cost in main memory (i.e., cpu time) but identifi es better trips. We omit the running time of hybrid approach from Figure 8(b) as it exhibits exponential increase($O(m^k)$) with the number of categories. However, when the number of categories is small, the running time of hybrid approach is comparable to $\mathcal{A}_{NN}$ and $\mathcal{A}_{MD}$, e.g., when $m = 5$ its running time is about 3.8 seconds for one query, on average.

7 Conclusions and Future Work

We introduced a novel query for spatial databases, namely the Trip Planning Query. First, we argued that this problem is NP-Hard, and then we developed four polynomial time approximation algorithms, with effi cient running time and varying worst case guarantees. We also showed how to apply these algorithms in practical scenarios, both for Euclidean spaces and Road Networks. Finally, we presented a comprehensive experimental evaluation. For future work we plan to extend our algorithms to support trips with user defi ned constraints. Examples include visiting a certain category during a specifi ed time period [3], visiting categories in a given order, and more.

References

1. S. Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *FOCS*, page 2, 1996.
2. S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Mathematical Programming*, 2003.
3. N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In *STOC*, pages 166–174, 2004.
4. N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 220–231, 1990.
5. R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest neighbor and reverse nearest neighbor queries for moving objects. In *IDEAS*, pages 44–53, 2002.
6. S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *PODS*, pages 78–86, 1997.
7. C. Böhm. A cost model for query processing in high dimensional data spaces. *TODS*, 25(2):129–178, 2000.
8. T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.
9. N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Computer Science Department,Carnegie Mellon University, 1976.
10. T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 1997.
11. A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for tsp with neighborhoods in the plane. In *SODA*, pages 38–46, 2001.
12. Max J. Egenhofer. What's special about spatial?: database requirements for vehicle navigation in geographic space. In *SIGMOD*, pages 398–402, 1993.
13. G. Even and G. Kortsarz. An approximation algorithm for the group steiner problem. In *SODA*, pages 49–58, 2002.
14. J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.
15. H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. E. Abbadi. Constrained nearest neighbor queries. In *SSTD*, pages 257–278, 2001.
16. N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *Journal of Algorithms*, 37(1):66–84, 2000.
17. R. Hartmut Guting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *TODS*, 25(1):1–42, 2000.
18. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.
19. G. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. *TODS*, 24(2):265–318, 1999.
20. E. Ihler. Bounds on the Quality of Approximate Solutions to the Group Steiner Problem. Technical report, Institut fur Informatik,Uiversity Freiburg, 1990.
21. M. R. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.

22. F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*, pages 201–212, 2000.
23. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
24. Y. S. Myung, C. H. Lee, and D. W. Tcha. On the Generalized Minimum Spanning Tree Problem. *Networks*, 26:231–241, 1995.
25. Digital Chart of the World Server. http://www.maproom.psu.edu/dcw/.
26. D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *ICDE*, pages 301–312, 2004.
27. D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, pages 802–813, 2003.
28. A. Papadopoulos and Y. Manolopoulos. Performance of nearest neighbor queries in r-trees. In *ICDT*, pages 394–408, 1997.
29. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.
30. C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. In *GIS*, pages 94–100, 2002.
31. M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The Optimal Sequenced Route Query. Technical report, Computer Science Department, University of Southern California, 2005.
32. S. Shekhar and D.-R. Liu. Ccam: A connectivity-clustered access method for networks and network computations. *TKDE*, 9(1):102–119, 1997.
33. TSP Home Web Site. http://www.tsp.gatech.edu/.
34. D. A. Spielman and S.-H. Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *STOC*, pages 296–305, 2001.
35. U.S. Geological Survey. http://www.usgs.gov/.
36. Y. Tao and D. Papadias. Time-parameterized queries in spatio-temporal databases. In *SIGMOD*, pages 334–345, 2002.
37. Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298, 2002.
38. Y. Tao, J. Zhang, D. Papadias, and N. Mamoulis. An Efficient Cost Model for Optimization of Nearest Neighbor Search in Low and Medium Dimensional Spaces. *TKDE*, 16(10):1169–1184, 2004.
39. Y. Theodoridis, E. Stefanakis, and T. Sellis. Efficient cost models for spatial queries using r-trees. *TKDE*, 12(1):19–32, 2000.
40. M. Vazirgiannis and O. Wolfson. A spatiotemporal model and language for moving objects on road networks. In *SSTD*, pages 20–35, 2001.
41. X. Xiong, M. F. Mokbel, and W. G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *ICDE*, pages 643–654, 2005.
42. X. Xiong, M. F. Mokbel, W. G. Aref, S. E. Hambrusch, and S. Prabhakar. Scalable spatio-temporal continuous query processing for location-aware services. In *SSDBM*, pages 317–327, 2004.
43. M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *SIGMOD*, pages 443–454, 2004.