The Multi-Rule Partial Sequenced Route Query

Haiquan Chen
Dept. of Computer Science and Software Engineering
Auburn University
Auburn, AL 36849, USA
chenhai@auburn.edu

Wei-Shinn Ku
Dept. of Computer Science and Software Engineering
Auburn University
Auburn, AL 36849, USA
weishinn@auburn.edu

Min-Te Sun
Dept. of Computer Science and Information Engineering
National Central University
Taoyuan 320, Taiwan
msun@csie.ncu.edu.tw

Roger Zimmermann
Dept. of Computer Science
National University of Singapore
Singapore 117590
rogerz@comp.nus.edu.sg

ABSTRACT

Trip planning search (TPS) represents an important class of queries in Geographic Information Systems (GIS). In many real-world applications, TPS requests are issued with a number of constraints. Unfortunately, most of these constrained TPS cannot be directly answered by any of the existing algorithms. By formulating each restriction into rules, we propose a novel form of route query, namely the multi-rule partial sequenced route (MRPSR) query. Our work provides a unified framework that also subsumes the well-known trip planning query (TPQ) and the optimal sequenced route (OSR) query. In this paper, we first prove that MRPSR is *NP*-hard and then present three heuristic algorithms to search for near-optimal solutions for the MRPSR query. Our extensive simulations show that all of the proposed algorithms can answer the MRPSR query effectively and efficiently. Using both real and synthetic datasets, we investigate the performance of our algorithms with the metrics of the route distance and the response time in terms of the percentage of the constrained points of interest (POI) categories. Compared to the LORD-based brute-force solution, the response times of our algorithms are remarkably reduced while the resulting route length is only slightly longer than the shortest route.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search Process*; H.2.8 [**Database Management**]: Database Application—*spatial databases and GIS*

General Terms

Algorithms

Keywords

Advanced traveler information systems, location-based services.

ACM GIS '08, November 5-7, 2008, Irvine, CA, USA.

1. INTRODUCTION

Geographic Information Systems (GIS) have been an active field of research during the past decade and many significant research results have been reported [21, 18, 2, 4, 20]. In GIS systems, spatial data types are first-class citizens and much work has focused on efficiently answering spatial queries. Past research efforts have mostly concentrated on nearest neighbor (NN) as well as range queries and their variants [22, 24, 11, 12]. While these basic query types are fundamental for many applications, more complex spatial query types must be considered for advanced GIS systems.

In this study we propose a novel query type for spatial databases in support of travel-planning GIS applications. The objective is to assist users in the planning of trips that involve several destinations, possibly belonging to different POI categories. Based on a number of traveling rules (or constraints) that are expressed as sub-sequences of locations, users aim to find the route with the relatively shortest traveling distance. Note that it is possible that the traveling rules may only involve a subset of the user-requested locations. Consider the following example. Alice is planning a trip around town that involves visiting the following locations: a bank, a restaurant, a gas station, and a movie theater. In addition, Alice also observes the following rules:

1. Visit a bank to withdraw money before having lunch at a restaurant.

2. Fill up gas before going to watch a movie.

In order to not violate the two rules, the planned trip must include two sub-sequences: (a) traveling to a bank before going to a restaurant and (b) visiting a movie theater after filling up the gas tank. Aside from these two sequences, Alice is free to visit any of the other POI categories in any order she pleases and furthermore, they can be interleaved in any order with the two rule-based sequences. Figure 1 illustrates two possible routes which can fulfill this query. We name this new query type *Multi-Rule Partial Sequenced Route* (MRPSR) query. Note that the MRPSR query differs from the *Traveling Salesman Problem* (TSP). In both cases a least-cost route is sought. However, with TSP a set of POIs (e.g., cities) is given and each element must be visited exactly once. In contrast, with MRPSR each POI is associated with a category and one may select any element of that category. For example, if the route should include a bank visit, then one may choose any one of

the available banks. Furthermore, with MRPSR an ordered subsequence may be imposed among some of the POIs. Such subsequences occur naturally in many GIS applications. Therefore, MRPSR queries are useful in numerous fields such as automotive navigation systems, supply chain management, online Web mapping services, and more.

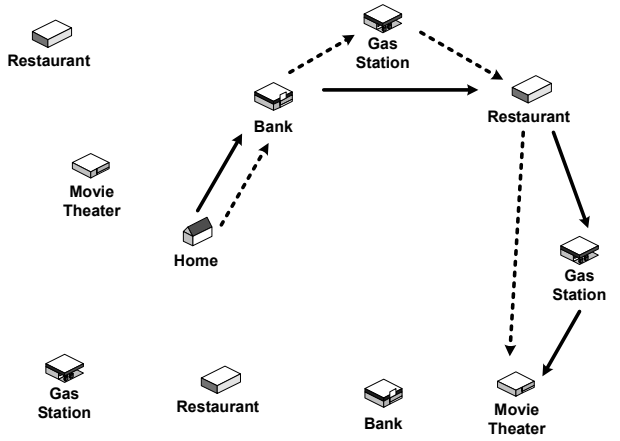

Figure 1: Two possible routes (solid and dashed arrows) of a partial sequenced route query.

In this paper we put forth three novel MRPSR query algorithms which are designed to efficiently compute feasible trips with the near-optimal travel distance. The contributions of our study are as follows:

- We formally define the partial sequenced route planning question and prove it to be a member of the *NP*-complete class of problems.
- We propose a Nearest Neighbor-based Partial Sequence Route query (NNPSR) algorithm. Our method utilizes topological sort [9] for combining multiple travel rules and executes a nearest neighbor query for planning the complete trip.
- We integrate NNPSR with the Light Optimal Route Discoverer (LORD) algorithm [20] to create NNPSR-LORD that further reduces the trip distance.
- We also design an Advanced A* Search-based Partial Sequence Route query (AASPSR) algorithm. AASPSR employs distance heuristic functions to generate efficient trip plans.
- We compare the performance of the proposed algorithms with the LORD-based brute-force solution through extensive simulation experiments.

The rest of the paper is organized as follows. Section 2 surveys the related work. The problem is formally defined in Section 3. In Section 4 we introduce the NNPSR, the NNPSR-LORD, and AASPSR algorithms. The experimental validation of our design is presented in Section 5. Section 6 concludes the paper with a discussion of future work.

2. RELATED WORK

In this section we review previous work related to nearest neighbor queries and route planning queries.

2.1 Nearest Neighbor Query

The nearest neighbor query is a very important query type for supporting Geographic Information Systems applications. With the R-tree family [5, 19, 1] of spatial indices, depth first search (DFS) [16] and best first search (BFS) [6] have been the prevalent branch-and-bound techniques for processing nearest neighbor queries. The DFS method recursively expands the intermediate nodes for searching NN candidates. At each newly visited index node, DFS computes the ordering metrics for all its child nodes and applies pruning strategies to remove non-promising branches. When the search reaches a leaf node, the data objects are retrieved and the NN candidates are updated. On the other hand, the BFS method employs a priority queue to store nodes to be explored through the search process. The nodes in the queue are sorted according to their minimum distance (MINDIST) to the query point. During the search process, BFS repeatedly dequeues the top entry in the queue and enqueues its child nodes with their MINDIST into the queue. When a data entry is dequeued, it is included in the result set.

Recently nearest neighbor search solutions have been extended to support queries on spatial networks. Jensen et al. [8] proposed data models and graph representations for NN queries in road networks and designed corresponding solutions. Papadias et al. [14] presented solutions for NN queries in spatial network databases by progressively expanding road segments around a query point. A network Voronoi diagram based solution for NN search in road network databases was proposed in [10].

2.2 Route Planning Query

In many GIS applications (e.g., logistics and supply chain management), users have to plan a trip to a number of locations with several sequence rules and the goal is to find the optimal route that minimizes the total traveling distance. One related query type is named the optimal sequenced route (OSR) query proposed by Sharifzadeh et al. [20]. An OSR query retrieves a route of minimum length starting from a given source location and passing through a number of locations (with different types) in a particular order imposed on the types of the locations. A multi-type nearest neighbor (MTNN) query solution was proposed in [13] by Ma et al. Given a query point and a collection of locations (with difference types), a MTNN query finds the shortest path for the query point such that only one instance of each type is visited during the trip. From a spatial query perspective, MTNN is an extended solution of OSR by exploiting a page-level upper bound. Li et al. [12] designed solutions for a new query type – Trip Planning Queries (TPQ). With a TPQ, the user specifies a subset (not a sequence) of location types R and asks for the optimal route from her starting location to a specified destination which passes through at least one database point of each type in R. Terrovitis et al. [23] illustrated a-autonomy shortest path and k-stops shortest path problems for spatial databases. Given a source point and a destination point, the first query retrieves a sequence of points from the database where the distance between any two consecutive points in the path is not greater than a. The second query searches for the optimal path from a origin to an end which passes through exactly k intermediate points in the database. However, all the aforementioned solutions cannot support MRPSR queries.

The most similar problem to MRPSR is the sequential ordering problem (SOP) [3] and it is stated as follows. Given a graph G with n vertices and directed weighted edges, find a minimal cost *Hamiltonian path* from the start vertex to the terminal vertex which also observes precedence constraints. Nevertheless, a Hamilton path is not required in MRPSR and the types of visited locations are con-

sidered by our solution.

3. THE MULTI-RULE PARTIAL SEQUENCED ROUTE QUERY

In this section, we formulate the proposed multi-rule partial sequenced route query and then discuss the properties of the proposed query. The definitions of the partial sequence rules and the multi-rule partial sequenced route query are introduced in Section 3.1. The properties of the multi-rule partial sequenced route query are discussed in Section 3.2.

3.1 Problem Formulation

Given n disjoint sets of POI categories $\{C_1, C_2, \ldots, C_n\}$, each containing a number of POIs in R^2, the MRPSR query is to ask for a route that satisfies the following three requirements:

1. The route will traverse through exactly one POI in each category;
2. The total traveling distance is minimized;
3. The route conforms with the given constraints (i.e., traveling rules).

While the first two requirements are commonly seen in the other types of TPS queries [20, 12], the third requirement is unique. Here, the issue is how we should properly define a constraint. Without loss of generality, we assume that each constraint can be mapped into a set of partial sequence rules, defined as follows.

Definition 3.1 A partial sequence rule is defined as an ordered subset of categories $C_{k_1} \rightarrow C_{k_2} \rightarrow \cdots \rightarrow C_{k_m}$, which specifies the order of visits between $< C_{k_i} >$ in the subset.

For instance, a user may issue a MRPSR query with a constraint that he would like to withdraw money before going for grocery shopping and dinner. This constraint can be converted to the following two partial sequence rules:

1. $C_{ATM} \rightarrow C_{Supermarket}$ and
2. $C_{ATM} \rightarrow C_{Restaurant}$.

These two rules enforce that an ATM will be visited before a supermarket and a restaurant on the route, but do not put a restriction on the order between the supermarket and the restaurant.

Notice that if no restriction is placed on the format of the user's constraints, the translation itself is a challenging artificial intelligence research problem [15]. The human natural language can be ambiguous and non-grammatical. The automatic translation requires the creation of algorithms that can deal with not only the ambiguity but also with parsing and interpretation of a large dynamic vocabulary, which is not likely to be accomplished in real time. With the help of input forms, the types of the user's constraints can be limited so that the translation from the constraints to the partial sequence rules can be handled with ease. With the notion of the partial sequence rules, the original definition of the MRPSR query can be formulated as follows.

Definition 3.2

Input: A set of POI categories and a set of traveling rules.

Output: The route with the minimal total traveling distance that satisfies the order specified in each of the partial sequence rules.

3.2 Properties of The MRPSR Query

The MRPSR query is by far the most general format of the route planning query. The following theorem shows that our query provides a unified framework that subsumes the well-known trip planning queries, including the trip planning queries (TPQ) [12] and the optimal sequenced route (OSR) queries [20].

THEOREM 3.1. *The problems of the trip planning query and the optimal sequenced route query are special cases of the problem of the multi-rule partial sequenced route query.*

PROOF. According to [12], the problem of the trip planning query is identical to the problem of the multi-rule partial sequenced route query when the set of partial sequence rules is empty. In addition, according to [20], the problem of the optimal sequenced route for a given sequence of categories of POIs is the same as the problem of the multi-rule partial sequenced route query when the set of partial sequence rules contains one partial sequence rule specifying the same order. □

From Theorem 3.1, we obtain the following important property for the MRPSR query.

COROLLARY 3.2. *The problem of the multi-rule partial sequence route query is NP-hard.*

PROOF. According to [12], the problem of the trip planning query is NP-hard. Since the problem of the trip planning query is a special case of the problem of the multi-rule partial sequenced route query (Theorem 3.1), it follows immediately that the problem of the multi-rule partial sequenced route query is NP-hard. □

Corollary 3.2 implies that when the search space is large, it is satisfactory to quickly find a *suboptimal* route that satisfies the given partial sequence rules instead of the route with the minimal total distance.

The set of the partial sequence rules plays an important role in the MRPSR query. As indicated in Theorem 3.1 and Corollary 3.2, if the set is empty, the search space will be large and the MRPSR query is NP-hard. However, if the rules specify a *total order* of the categories, the MRPSR problem can be solved in polynomial time [20]. Intuitively, the tighter the set of rules is, the smaller the search space will be and the easier the MRPSR query can be answered. While it is difficult to quantify the level of tightness for a set of partial sequence rules, we provide the following definition to see if a given set of rules will possibly lead to a solution.

Definition 3.3 A set of the partial sequence rules is defined to be compatible if and only if there is a total order of $< C_i >$ that satisfies the order specified in each of the rules in the set.

For instance, the set of rules $\{C_1 \rightarrow C_2, C_2 \rightarrow C_3, C_3 \rightarrow C_1\}$ is not compatible since it will be impossible to satisfy all these three rules at the same time. The following theorem shows the relationship between the solvability of a MRPSR query and the compatibility of a given set of rules.

THEOREM 3.3. *If a multi-rule partial sequenced route query is solvable, then the corresponding set of the partial sequence rules must be compatible.*

PROOF. The proof is done by contradiction. Assume that the set of rules is not compatible, then according to Definition 3.3 there is no ordered sequence of categories that satisfies all the rules. In other words, no matter how POIs are selected, it will be impossible to order them so that the ordered sequence meets all of the constraints. □

Notice that Theorem 3.3 does not guarantee that a compatible set of partial sequence rules can always lead to a solution for a corresponding MRPSR query because some categories may contain no POI. If each category contains at least one POI, the inverse of Theorem 3.3 (i.e., the compatible set of rules implies the solvability of the corresponding MRPSR query) will also be true. In Section 4, we will elaborate how to verify if a set of partial sequence rules is compatible.

4. SYSTEM DESIGN

We propose three multi-rule partial sequenced route query algorithms in this section: the Nearest Neighbor-based Partial Sequenced Route (NNPSR) algorithm, the Nearest Neighbor-based Partial Sequenced Route with Light Optimal Route Discoverer (NNPSR-LORD) algorithm, and the Advanced A* Search-based Partial Sequenced Route (AASPSR) algorithm. NNPSR applies topological sort for combining traveling rules and utilizes a nearest neighbor query for planning the whole trip. NNPSR-LORD further applies the Light Optimal Route Discoverer algorithm on the route obtained from NNPSR to acquire a shorter route. Distance heuristic functions are adopted in AASPSR for planning route sequences to reduce the trip distance. All of the proposed algorithms aim to find the near-optimal route which follows all of the traveling rules. Table 1 summarizes our set of notations.

Symbol	Meaning
$\mathbb{A}$	The adjacency list representation of an AOV
$\mathbb{C}$	The set of all the user selected categories
$\mathbb{R}$	The set of all the traveling rules
$\mathbb{P}$	A set of POIs
$\|\mathbb{S}\|$	The number of elements in set $\mathbb{S}$
S	The starting point of a trip
D	The destination of a trip
Q	The query point of a nearest neighbor query
T	The plan of a certain trip
C	A POI category
$C.\mathbb{P}$	All the POIs of a category
$p_x.C$	The category of a POI p_x
$p_x.L$	The location of a POI p_x
L_{zero}	A list of AOV vertices with a zero count
L_{route}	A list of the POI sequence of a trip plan
NN(Q, $\mathbb{P}$)	Nearest neighbor query of query point Q on a POI set $\mathbb{P}$
Dist(x, y)	The Euclidean distance between points x and y

Table 1: Symbolic notations.

4.1 Nearest Neighbor-based Partial Sequenced Route Algorithm

In order to plan a route which can fulfill all the user defined partial sequence rules, we need a solution to combine all the provided traveling rules and verify if they are compatible. The relationship between all the given traveling rules can be represented as a directed graph in which the vertices represent POI categories and the directed edges represent prerequisites. This graph has an edge $<i, j>$ if and only if category i is an immediate prerequisite for category j in one of the rules. The complete graph is named an Activity On Vertex (AOV) network [7]. The following theorem provides the relationship of an AOV network with the compatibility of the traveling rules.

THEOREM 4.1. *The rules are compatible if and only if the corresponding AOV network is a directed acyclic graph.*

PROOF. Definition 3.3 indicates that the rules are compatible if and only if there is a category sequence that satisfies the order specified in each of the traveling rules. Let that category sequence be the feasible sequence of tasks that satisfies all of the orders. According to [7], an AOV has a feasible sequence of tasks if and only if the precedence relations in the AOV network are both transitive and irreflexive. In other words, the corresponding AOV network must be directed and acyclic. □

Table 2 lists the POI categories covered by the traveling rules of a certain trip plan T_x. The AOV network corresponding to the traveling rules in Table 2 is shown in Figure 2.

Data Type	Name	Prerequisites
C1	Bank	None
C2	Bookstore	None
C3	Restaurant	C1, C2
C4	Gas Station	None
C5	Hospital	C4
C6	Shopping Center	C5
C7	Church	C3, C6
C8	Coffee Shop	C3
C9	Gift Shop	C7, C8
C10	Park	C7

Table 2: POI categories needed based on traveling rules for trip T_x.

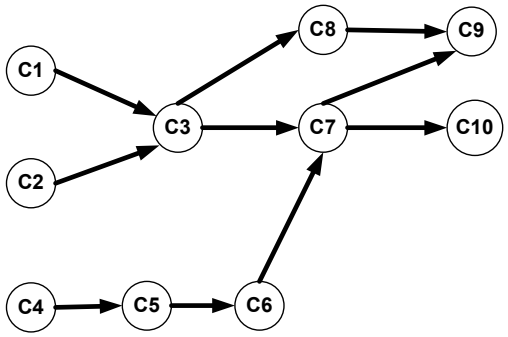

Figure 2: AOV network represents POI categories as vertices and prerequisites as edges.

After we obtain an AOV of a certain trip plan T_x, the following step is to generate a linear ordering, $v_{i0}, v_{i1}, \ldots, v_{n-1}$ of the vertices in the AOV, referred to as the *Topological Order*, provided the associated AOV network is directed and acyclic. In graph theory, a topological order of a directed acyclic graph (DAG) is a linear ordering of its vertices in which each vertex comes before all vertices to which it has outbound edges. An algorithm that sorts the tasks into topological order is straightforward. The first step is to list out a vertex in the network that has no predecessor. Then the second step is to delete this vertex and all edges leading out from it from the AOV. These two steps are repeated until either all the vertices have been listed or all remaining vertices have predecessors and hence none of them can be removed. In the latter case, the AOV has a cycle and the trip is infeasible, i.e., the partial sequence rules are not compatible. If a topological order has the property that

all pairs of consecutive vertices in it are connected by AOV edges, then these edges form a directed Hamiltonian path in the AOV [9]. If a Hamilton path exists, the topological sort order is unique and no other order respects the edges of the path. On the contrary, if a topological order does not form a Hamiltonian path, the AOV will have two or more valid topological orderings, for in this case it is always possible to form a second valid ordering by swapping two consecutive vertices that are not connected by an AOV edge to each other. To support both cases, we keep a counter of the number of immediate predecessors for each vertex and represent the network by its adjacency lists. We can then carry out the deletion of all incident edges of a vertex v by decreasing the predecessor count of all vertices on its adjacency list. Whenever the count of a vertex drops to zero (in-degree $= 0$), we place the vertex on a list (L_{zero}) of vertices with a zero count. As mentioned in Section 1, the traveling rules (the AOV network) may not cover all the user selected POI categories. With the goal of creating a complete trip plan (i.e., the plan covers all requested categories), we add all the requested POI types which are not included in the AOV into the list L_{zero}. The complexity of topological sort is $O(e + n)$, where n is the number of vertices and e is the total edge number. The sort can be finished in linear time.

We can start to compute a complete trip plan after having the corresponding topological order in hand. We devise a Nearest Neighbor-based Partial Sequence Route query algorithm by utilizing both the L_{zero} list and any well-known nearest neighbor query algorithm to generate an efficient route. With NNPSR, we first search for the nearest POI whose category is included in L_{zero} from the query point Q (as the starting point). The retrieved nearest POI p_x will be stored in a route list L_{route} and the category of p_x ($p_x.C$) will be removed from L_{zero}. Next, we update the adjacency list and new zero count vertices may be added to L_{zero}. In addition, the query point Q is also updated to the location of p_x ($p_x.L$). The process will repeat until all the selected categories are contained in the route. The complete algorithm of NNPSR is formalized in Algorithm 1.

Algorithm 1 Nearest Neighbor-based Partial Sequenced Route query($\mathbb{C}$, $\mathbb{R}$, S)

```
1: Set L_route = ∅ and Q = S
2: Integrate all elements in ℝ into an AOV adjacency list 𝔸 and
   put all vertices with zero count in L_zero
3: if The AOV network is a DAG then
4:    Add all elements of ℂ \ 𝔸 into L_zero
5:    while L_zero ≠ ∅ do
6:       ℙ = ∅
7:       for each C_i ∈ L_zero do
8:          ℙ = C_i.ℙ ∪ ℙ
9:       end for
10:      p_NN = NN(Q, ℙ)
11:      L_route = L_route ∪ p_NN
12:      Remove p_NN.C from L_zero
13:      Update 𝔸 and L_zero
14:      Q = p_NN.L
15:   end while
16:   return L_route
17: else
18:   Report cycles in ℝ
19: end if
```

4.2 Performance Improvement with the Light Optimal Route Discoverer Algorithm

As described in Section 2, if we have a given sequence of POI categories, the Light Optimal Route Discoverer (LORD) algorithm [20] can retrieve a route of minimum length. Since we can obtain a complete POI sequence after executing the NNPSR algorithm, we can further improve the efficiency of the trip plan by combining NNPSR with LORD. LORD is a threshold-based algorithm and requires less memory space compared with Dijkstra's shortest path solution. The first step in LORD is to issue consecutive nearest neighbor queries to find the greedy route that follows the given POI category sequence from the starting point. Then, the length of the greedy route becomes a constant threshold value T_c. In addition, LORD also keeps a variable threshold value T_v whose value reduces after each iteration and LORD discards all the POIs whose distances to the starting point are more than T_v. Afterward, LORD iteratively builds and maintains a set of partial sequenced routes in the reverse sequence (i.e., from the end points toward the starting point). During each iteration of LORD, POIs from the following category are added to the head of each of these partial sequence routes to make them closer to the starting point. The two thresholds are utilized to prune non-promising routes for reducing the search space.

After executing the NNPSR algorithm, we can acquire a sequence of POIs. Since each POI belongs to an individual POI category, we can also obtain a POI category sequence as the input of LORD. For most cases, the compound NNPSR-LORD solution outperforms the original NNPSR algorithm. More detailed performance evaluations are demonstrated in Section 5.

4.3 A* Search-based Partial Sequenced Route Algorithm

Although the NNPSR and NNPSR-LORD algorithms can fulfill the traveling rules and reduce the travel distance of a trip, they do not consider the location of the destination when greedily generating the route sequence. Consider the example shown in Figure 3. By running NNPSR, we will find that the created trip plan T_1 = $\{S, p_x, p_y, p_z, D\}$ (the dashed route) is much longer than another feasible trip plan T_2 = $\{S, p_a, p_b, p_c, D\}$ (the solid route) which considers the destination location. A more advanced approach is to limit the trip planning within a range defined by S and D (e.g., an ellipse and S and D are the two focal points). Consequently, we design an A* Search-based Partial Sequenced Route (ASPSR) query algorithm which takes the location of the destination into account.

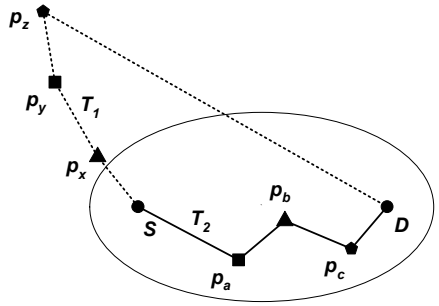

Figure 3: Two trip plans generated by NNPSR (the dashed route) and ASPSR (the solid route) respectively.

Similar to the *admissible heuristic* of the A* algorithm [17], we can use the sum of costs Dist(S, p_x) + Dist(p_x, D) to retrieve the POI which has the minimum total cost among all POIs belonging

to categories included in L_{zero} from the starting point. Afterward the lowest cost POI p_x will be inserted into the route list L_{route} and the category of p_x will be withdrawn from L_{zero}. Then both $\mathbb{A}$ and L_{zero} will be updated and the location of p_x is set as the new query point. The process will repeat until all the user selected categories are covered in the trip plan.

4.4 Advanced A* Search-based Partial Sequenced Route Algorithm

The A* search based solution in Section 4.3 considers the location of the destination in its heuristic function. However, there could be roundabout ways when we have skewed POI sets. Consider the example of Figure 4. Because the POIs which are closer to the major axis of an ellipse (S and D are the two focal points) have a lower cost, p_x will be the first picked POI from S. Assume that the other two POI sets are skewed. A user has to take a route which is far away from the destination to visit p_y and p_z before traveling to D. Consequently, we need to improve the A* search based solution to solve the aforementioned problem.

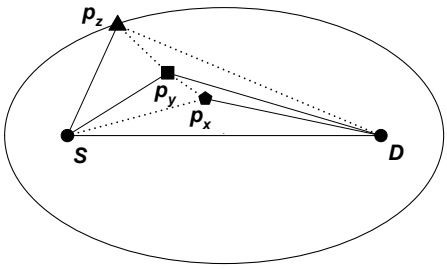

Figure 4: A roundabout way route (the dashed route) generated by the ASPSR algorithm.

The improved version of the A* search based solution is as follows. The Advanced A* Search-based Partial Sequenced Route query (AASPSR) algorithm computes a set of POIs $\mathbb{P}$*, one per category in $\mathbb{C}$, such that every element in $\mathbb{P}$* has the minimum traveling distance from S to D. After $\mathbb{P}$* has been generated, AASPSR creates a trip from S to D by utilizing both the AOV adjacency list and $\mathbb{P}$*. Starting with S, we first search for the nearest POI p_x in $\mathbb{P}$* and the category of p_x must be in L_{zero}. Afterward, p_x is inserted into L_{route} and the location of p_x is used as the new NN query point. Then, we remove the category of p_x from L_{zero} and recompute the adjacency list. The whole process will repeat until L_{zero} becomes empty. The complete algorithm of AASPSR is illustrated in Algorithm 2.

Algorithm 2 Advanced A* Search-based Partial Sequenced Route query($\mathbb{C}$, $\mathbb{R}$, S, D)

```
1: Set L_route = ∅ and Q = S
2: Integrate all elements in ℝ into an AOV adjacency list 𝔸 and
   put all vertices with zero count in L_zero
3: if The AOV network is a DAG then
4:    Add all elements of ℂ \ 𝔸 into L_zero
5:    for i = 0; i < |ℂ|; i++ do
6:       Min = ∞
7:       for j = 0; j < |C_i|; j++ do
8:          Cost = Dist(S, p_j) + Dist(p_j, D)
            {p_j ⊂ C_i}
9:          if Cost < Min then
10:            Min = Cost
11:            p_min = p_j
12:         end if
13:      end for
14:      ℙ* = ℙ* ∪ p_min
15:   end for
16:   while L_zero ≠ ∅ do
17:      ℙ = ∅
18:      for each C_i ∈ L_zero do
19:         Retrieve the corresponding p_x in ℙ* and
20:         ℙ = ℙ ∪ p_x
21:      end for
22:      p_NN = NN(Q, ℙ)
23:      L_route = L_route ∪ p_NN
24:      Remove p_NN.C from L_zero
25:      Update 𝔸 and L_zero
26:      Q = p_NN.L
27:   end while
28:   return L_route
29: else
30:   Report cycles in ℝ
31: end if
```

5. EXPERIMENTAL VALIDATION

We implemented the NNPSR, NNPSR-LORD, and AASPSR algorithms to evaluate their performances with respect to the distance of the returned route and the response time to generate the route. The experimental results are reported in this section. To highlight the benefits of our approaches, we use the LORD-based brute-force solution, which applies LORD [20] to reduce the search space of each possible permutation of the category sequence, to get the *shortest route distance* and record the *corresponding response time*. We varied the following parameters to obtain their effects on the route distance and response time: the percentage of the constrained categories (PCC), the average category cardinality (ACC), and the number of total categories (NTC). For a given MRPSR query, PCC is defined as the percentage of the number of categories involved in the traveling rules with respect to the total number of categories and ACC is defined as the average number of POIs of all categories. We used both the real dataset from the state of California and synthetic datasets. The California dataset has 63 different categories and the categories used in this research are shown in Table 5. The synthetic datasets consist of randomly generated POIs with a uniform distribution in the longitude range from 124W to 114W and latitude range from 32N to 42N. In Sections 5.1 and 5.2, we assume the category number is 6. In the real dataset, rules are generated that involve either the categories of Church and Hospital or Locale and Park according to the different level of PCC. In the synthetic datasets, rules are generated that involve either the second and the third category or the forth and the fifth category according to the different level of PCC. For each investigated algorithm, 100 MRPSR queries were generated with a randomly generated starting point and a destination in the simulated area for a given configuration. All the experiments were conducted on a Linux machine with an Intel Pentium 4 2.4GHz CPU.

5.1 Effect of the Percentage of the Constrained Categories

In this subsection, we varied the percentage of the constrained categories to investigate the performance of NNPSR, AASPSR, and NNPSR-LORD on route distance and response time and to compare them with the LORD-based brute-force solutions. Since our proposed MRPSR query subsumes the TPQ and OSR queries, the MRPSR query exhibits the characteristics of a TPQ query when

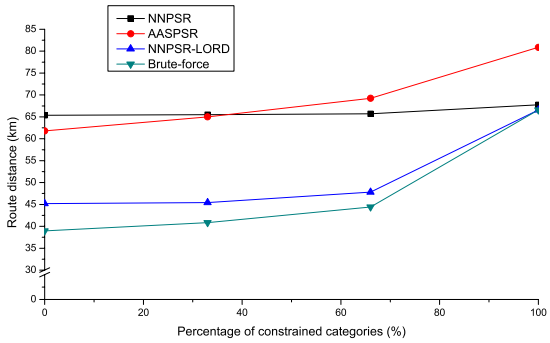

Fig. 5(a) California dataset

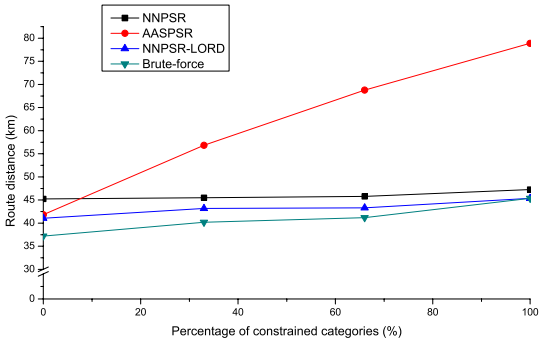

Fig. 5(b) Synthetic dataset

Figure 5: Route distance of NNPSR, AASPSR, NNPSR-LORD, and LORD-based brute-force as a function of PCC.

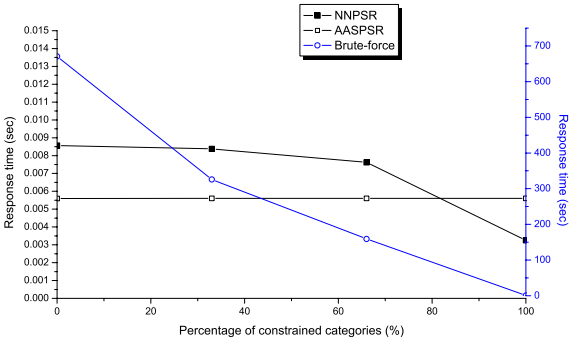

Fig. 6(a) NNPSR, AASPSR, and LORD-based brute-force (real)

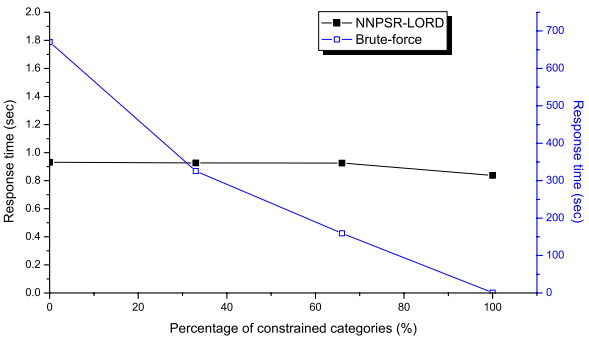

Fig. 6(b) NNPSR-LORD and LORD-based brute-force (real)

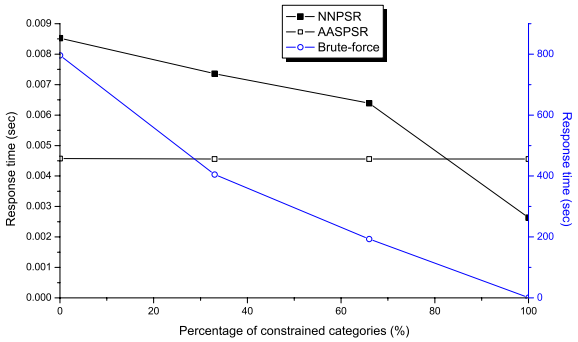

Fig. 6(c) NNPSR, AASPSR, and LORD-based brute-force (synthetic)

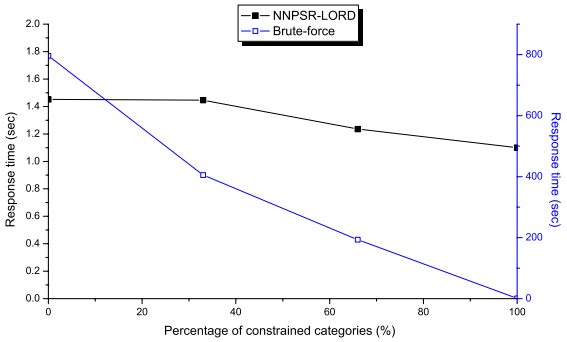

Fig. 6(d) NNPSR-LORD and LORD-based brute-force (synthetic)

Figure 6: Response time of NNSPR, AASPSR, NNPSR-LORD, and LORD-based brute-force as a function of PCC.

PCC decreases and the characteristics of an OSR query when PCC increases. Figure 5 illustrates the relationship between route distance and PCC for NNPSR, AASPSR, NNPSR-LORD, and LORD-based brute-force. Our observations are as follows:

1. Route distance increases with the increase of PCC for all the algorithms. This is because with higher PCC there will be more restrictions on the order of the categories, which leads to a longer route.

2. The route distance of AASPSR changes remarkably against PCC in contrast to NNPSR and NNPSR-LORD. The lower PCC is, the better AASPSR works compared with NNPSR. Hence, AASPSR is only suitable for trips with low PCC, for example, TPQ (PCC equals zero). With a higher PCC, the route distance of AASPSR increases dramatically. This is because AASPSR only picks a single POI in each category for the subsequent NN search. With a higher PCC, there are more restrictions on the category order, which generates a longer route. Therefore, the performance of AASPSR is not necessarily better than NNPSR in terms of route distance.

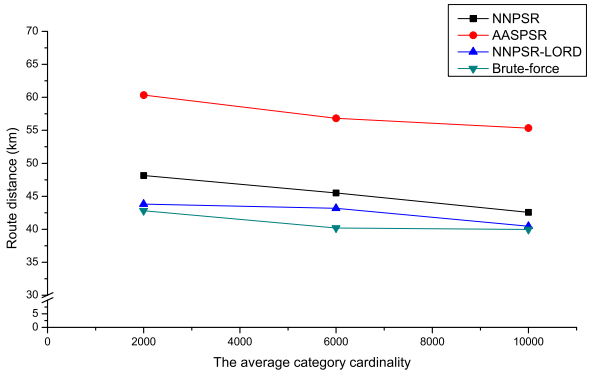

Fig. 8(a) Percentage of constrained categories = 33%

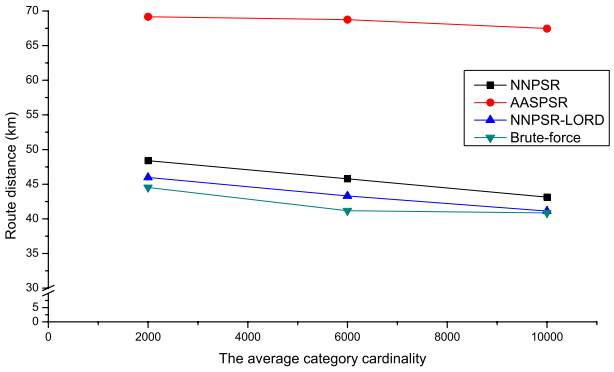

Fig. 8(b) Percentage of constrained categories = 66%

Figure 8: Route distance of NNSPR, AASPSR, NNPSR-LORD, and LORD-based brute-force as a function of the average category cardinality.

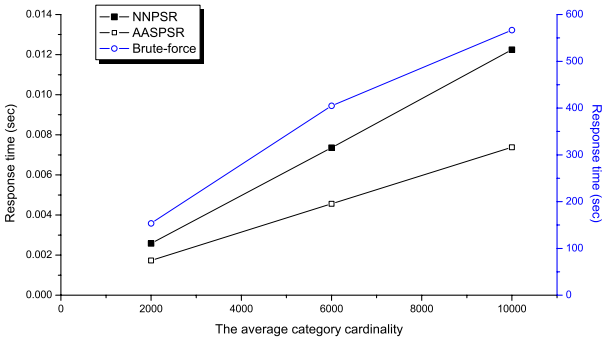

Fig. 9(a) NNPSR, AASPSR, and LORD-based brute-force (PCC = 33%)

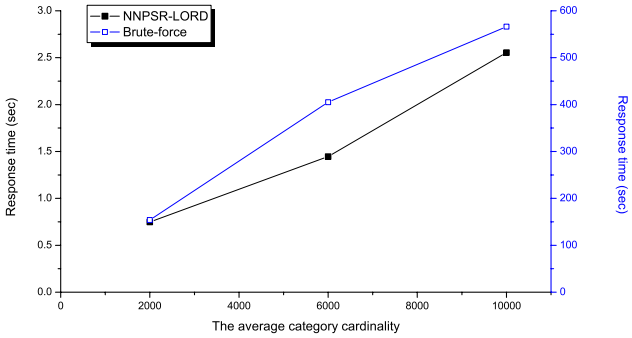

Fig. 9(b) NNPSR-LORD and LORD-based brute-force (PCC = 33%)

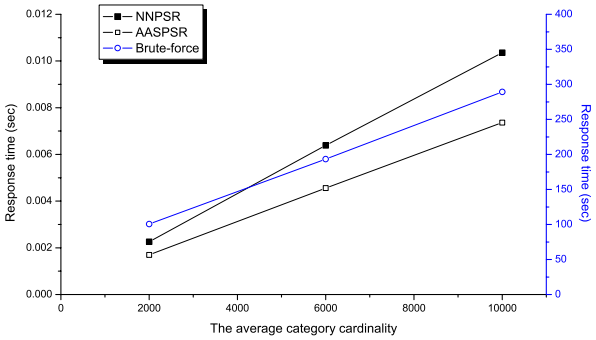

Fig. 9(c) NNPSR, AASPSR, and LORD-based brute-force (PCC = 66%)

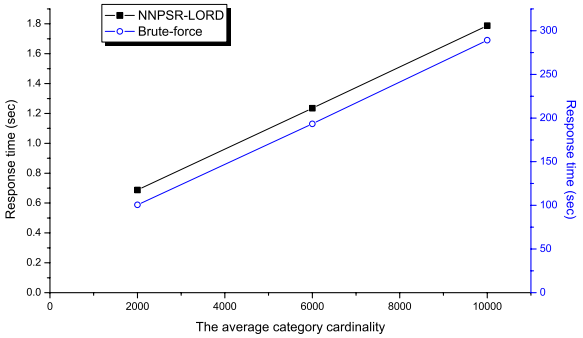

Fig. 9(d) NNPSR-LORD and LORD-based brute-force (PCC = 66%)

Figure 9: Response time of NNSPR, AASPSR, NNPSR-LORD, and LORD-based brute-force as a function of the average category cardinality.

For instance, in Figure 7, the triangle, rectangle, and pentagon each represent a different category of POIs, and S and D denote the start and destination of the trip. The NNPSR created trip plan $T_1 = \{S, p_{x1}, p_{y1}, p_{z1}, D\}$ (the solid route) is shorter than the AASPSR created trip plan $T_2 = \{S, p_{x2}, p_{y2}, p_{z2}, D\}$ (the dashed route). The partial sequence rule, **triangle** $\rightarrow$ **rectangle** $\rightarrow$ **pentagon**, leads to a roundabout route in AASPSR, which only chooses the POIs inside the ellipse. Consequently, AASPSR performs worse than NNPSR in terms of route distance in this scenario.

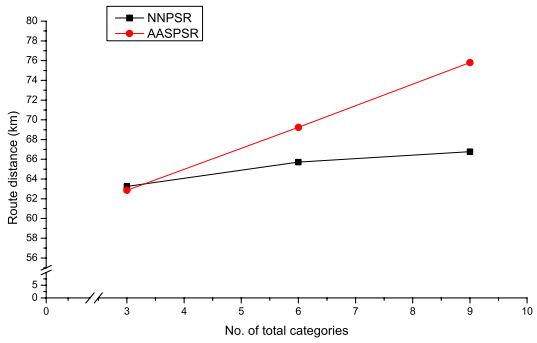

Figure 10: Route distance of NNSPR and AASPSR as a function of the number of total categories.

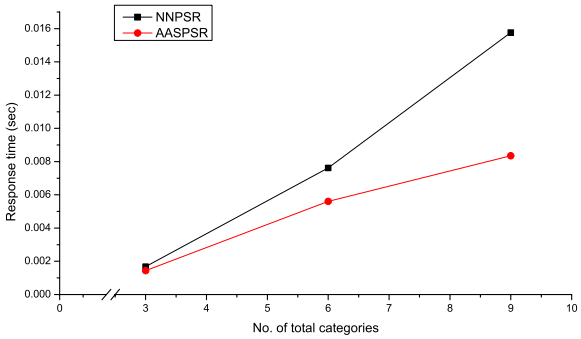

Figure 11: Response time of NNSPR and AASPSR as a function of the number of total categories.

Category	Size
Building	4110
Church	7680
Hospital	835
Locale	13481
Park	6728
School	11173
Populated place	6900
Summit	5594
Valley	7596

Table 3: The California dataset used in our experiments.

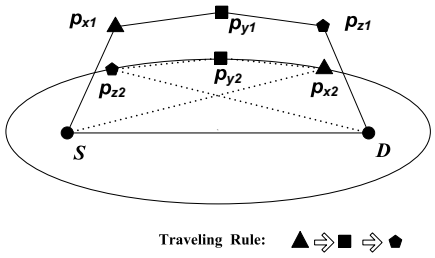

Figure 7: An example where NNPSR generates a shorter route than AASPSR.

3. NNPSR-LORD outperforms NNPSR and AASPSR in terms of route distance given any PCC. The reason is that NNPSR-LORD employs LORD to obtain the shortest route under the specific order of categories in the route found by NNPSR.

Figure 6 plots the response time against PCC for NNPSR, AASPSR, NNPSR-LORD, and the LORD-based brute-force method. Notice that, because the response times of the shortest route are in a different order of magnitude, the right Y axis corresponds to the LORD-based brute-force case and the left Y axis corresponds to our proposed algorithms. First, as shown in Figure 6, all our proposed algorithms significantly reduce the response time compared with the LORD-based brute-force solution. To be specific, NNPSR and AASPSR are about 100,000 times faster than the LORD-based brute-force method while NNPSR-LORD is about 200 times faster. Second, the response times decrease with the increase of PCC. This is because a higher PCC will decrease the search space of POIs. Third, PCC has little impact on the response time of AASPSR. The reason is because AASPSR only selects from each category the POI with the shortest distance sum from the starting point and the destination and thus the cost of computation is relatively constant. In particular, AASPSR responds faster than NNPSR with a lower PCC.

5.2 Effect of the Average Category Cardinality

The cardinality of each category has an impact on the route distance and response time. In this subsection, we studied the effect of the average category cardinality by varying the cardinality from 2,000 to 10,000 using synthetic datasets. Figure 8 shows the route distances of NNPSR, AASPSR, NNPSR-LORD, and the LORD-based brute-force method where PCC equals 33% and 66%, respectively. Figure 9 shows the response time for the above algorithms. Our observations are as follows:

1. The route distance reduces for each algorithm with the increase of the average cardinality. The reason is that a denser distribution of a category will lead to more POI choices, which results in a lower probability of detours.

2. The response time of each algorithm increases with the enlargement of the average cardinality. This is a result of the expanded POI search space which each algorithm uses to answer a query.

3. With respect to route distance, AASPSR has poor performance under either 33% or 66% of PCC because, as mentioned in Section 5.1, AASPSR outperforms NNPSR in terms of route distance only if PCC is low. Both 33% and 66% of PCC are large enough to deteriorate the performance of AASPSR. On the other hand, AASPSR has the shortest respond time among all the proposed algorithms under either 33% or 66% of PCC because the computation cost of AASPSR comes mainly from the selection of the POIs within a small ellipse whose foci are the starting point and the destination as discussed in Section 4.4. To sum up, whether AASPSR can outperform NNPSR depends largely on the level of PCC.

5.3 Effect of the Total Number of Categories

In this subsection, we changed the total number of categories to 3, 6, and 9 to investigate the impact of the category number on the performance of NNPSR and AASPSR. We assume that PCC equals 66% and use the California dataset. Figure 10 and Figure 11 illustrate the simulation results. Our observations are as follows:

1. When the category number increases, the route distance of both NNPSR and AASPSR extends. This is because with an increasing number of categories there will be more POIs to be traversed.

2. When the category number increases, the response time prolongs accordingly. The reason is that both NNPSR and AASPSR need to compute more categories to answer a MRPSR query.

3. The number of categories has a significant impact on whether AASPSR has a better performance than NNPSR. For 3 categories, AASPSR outperforms NNPSR in terms of both the route distance and response time. On the other hand, in case of either 6 or 9 categories, AASPSR responds faster but returns a longer route than NNPSR. The reason is that fewer categories will lead to a lower probability for a detour to occur when AASPSR traverses the POIs in a smaller ellipse. Therefore, we conclude that AASPSR outperforms NNPSR in terms of both the route distance and response time on the condition that a relatively small number of categories is requested in a MRPSR query.

6. CONCLUSIONS

Geographic information systems are getting increasingly sophisticated and trip planning with traveling rules represent a significant class of queries. Existing solutions work on trips with a predefined complete POI category sequence or no sequence at all. However, GIS users usually set traveling preferences when they plan their trips. We have introduced three multi-rule partial sequenced route query algorithms which can fulfill all the traveling rules and effectively decrease the traveling distance. We have shown through simulation results that our techniques generate trip plans which are very close to the shortest routes with noticeably short response time.

For future work, we plan to extend our algorithms to support multi-user MRPSR queries, in which users travel cooperatively to visit a set of POIs for a specific task. Examples include how the members of a rescue team work together to quickly inspect all of the suspicious locations for possible survivors after a catastrophe.

7. ACKNOWLEDGMENTS

This research has been funded in part by the US National Science Foundation (NSF) Grant CNS-0831502 (Cyber Trust) and NUS AcRF grant WBS R-252-050-280-101/133. We also acknowledge the support of the NUS Interactive and Digital Media Institute (IDMI).

8. REFERENCES

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD Conference*, pages 322–331, 1990.

[2] C. Beeri, Y. Kanza, E. Safra, and Y. Sagiv. Object Fusion in Geographic Information Systems. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB)*, pages 816–827, 2004.

[3] L. F. Escudero. An Inexact Algorithm for the Sequential Ordering Problem. *European Journal of Operational Research*, 37(2):236–249, 1988.

[4] B. George, S. Kim, and S. Shekhar. Spatio-temporal Network Databases and Routing Algorithms: A Summary of Results. In *Proceedings of the 10th International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, pages 460–477, 2007.

[5] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD'84, Proceedings of Annual Meeting*, pages 47–57, 1984.

[6] G. R. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. *ACM Trans. Database Syst.*, 24(2):265–318, 1999.

[7] E. Horowitz, S. Sahni, and S. Anderson-Freed. *Fundamentals of Data Strucures in C*. W. H. Freeman, 1993.

[8] C. S. Jensen, J. Kolárvr, T. B. Pedersen, and I. Timko. Nearest Neighbor Queries in Road Networks. In *Proceedings of the 11^{th} ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS)*, pages 1–8, 2003.

[9] A. B. Kahn. Topological Sorting of Large Networks. *Commun. ACM*, 5(11):558–562, 1962.

[10] M. R. Kolahdouzan and C. Shahabi. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 840–851, 2004.

[11] W.-S. Ku, R. Zimmermann, H. Wang, and C.-N. Wan. Adaptive Nearest Neighbor Queries in Travel Time Networks. In *Proceedings of the 13^{th} ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS)*, pages 210–219, 2005.

[12] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng. On Trip Planning Queries in Spatial Databases. In *Proceedings of the 9th International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, pages 273–290, 2005.

[13] X. Ma, S. Shekhar, H. Xiong, and P. Zhang. Exploiting a Page-Level Upper Bound for Multi-Type Nearest Neighbor Queries. In *Proceedings of the 14^{th} ACM International Symposium on Geographic Information Systems (ACM-GIS)*, pages 179–186, 2006.

[14] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query Processing in Spatial Network Databases. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pages 802–813, 2003.

[15] R. Reddy. To dream the possible dream. *Commun. ACM*, 39(5):105–112, 1996.

[16] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest Neighbor Queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79, 1995.

[17] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[18] H. Samet. Issues, Developments, and Challenges in Spatial Databases and Geographic Information Systems (gis). In *Proceedings of the 9^{th} ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS)*, page 1, 2001.

[19] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of 13th International Conference on Very Large Data Bases (VLDB)*, pages 507–518, 1987.

[20] M. Sharifzadeh, M. R. Kolahdouzan, and C. Shahabi. The Optimal Sequenced Route Query. *The VLDB Journal*, 17(4):765–787, 2008.

[21] S. Shekhar, M. Coyle, B. Goyal, D.-R. Liu, and S. Sarkar. Data Models in Geographic Information Systems. *Commun. ACM*, 40(4):103–111, 1997.

[22] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB)*, pages 287–298, 2002.

[23] M. Terrovitis, S. Bakiras, D. Papadias, and K. Mouratidis. Constrained Shortest Path Computation. In *Proceedings of the 9th International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, pages 181–199, 2005.

[24] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based Spatial Queries. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 443–454, 2003.