**GeoInformatica manuscript No.**
(will be inserted by the editor)

# The Partial Sequenced Route Query with Traveling Rules in Road Networks

**Haiquan Chen · Wei-Shinn Ku · Min-Te Sun · Roger Zimmermann**

Received: date / Accepted: date

**Abstract** In modern geographic information systems, route search represents an important class of queries. In route search related applications, users may want to define a number of traveling rules (traveling preferences) when they plan their trips. However, these traveling rules are not considered in most existing techniques. In this paper, we propose a novel spatial query type, the multi-rule partial sequenced route (MRPSR) query, which enables efficient trip planning with user defined traveling rules. The MRPSR query provides a unified framework that subsumes the well-known trip planning query (TPQ) and the optimal sequenced route (OSR) query. The difficulty in answering MRPSR queries lies in how to integrate multiple choices of points-of-interest (POI) with traveling rules when searching for satisfying routes. We prove that MRPSR query is *NP*-hard and then provide three algorithms by mapping traveling rules to an activity on vertex network. Afterwards, we extend all the proposed algorithms to road networks. By utilizing both real and synthetic POI datasets, we investigate the performance of our algorithms. The results of extensive simulations show that our algorithms are able to answer MRPSR queries effectively and efficiently with underlying road networks. Compared to the Light Optimal Route Discoverer (LORD) based brute-force solution, the response time of our algorithms is significantly reduced while the distances of the computed routes are only slightly longer than the shortest route.

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

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

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

Roger Zimmermann
Dept. of Computer Science
National University of Singapore, Singapore 117590
E-mail: rogerz@comp.nus.edu.sg

**Keywords** Advanced traveler information systems · path search · query processing · location-based services.

## 1 Introduction

In Geographic Information Systems (GIS) related research [4,8,24,27,30], significant efforts have been spent on nearest neighbor (NN) queries, range queries as well as their variants [16, 18,31,35]. While these query types are building blocks for many existing applications, more advanced spatial query types must be studied for future GIS systems. Route queries [5, 14, 18, 27, 32, 33] are an important class of spatial queries for users to request an efficient path by specifying a source and a destination. As an essential component, route queries are widely supported by many of today's popular online map service providers (e.g., Google Maps[1], MapQuest [2], Yahoo! Maps[3], Bing Maps[4]). By issuing a route query to a map service provider, users will obtain a recommended route on the map with an estimated mileage and turn-by-turn driving instructions. Li et al. [18] proposed solutions for Trip Planning Queries (TPQ). With TPQ, the user specifies a set of Point of Interest (POI) types and asks for the optimal route (with minimum distance) from her starting location to a specified destination which passes through exactly one POI in each POI type. On the other hand, Sharifzadeh et al. [27,29] presented OSR queries where the user asks for an optimal route from her starting location and passing through a number of POIs (each with a different type) in a particular order (sequence) imposed on all the types of POIs to be visited. However, both TPQ and OSR queries fail to consider the sub-sequences of POI types which occur naturally in many GIS applications. To remedy this, in this study, we propose a novel route query type, *Multi-Rule Partial Sequenced Route* (MRPSR) query. Our objectives are to assist users to plan trips that involve multiple POIs which belong to different POI categories (types) and satisfy a number of user defined traveling rules in road networks with a short response time. Our MRPSR query aims at unifying the well-known TPQ and OSR queries.

### 1.1 Motivation

As a motivating application, consider the scenario as shown in Figure 1. Alice is planning a trip that starts from her home and involves visiting the following POI categories: a bank, a restaurant, a gas station, and a movie theater. In addition, Alice also makes the following traveling rules on her trip:

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 fulfill the two traveling rules, the returned trip must contain 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 in a gas station. 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 satisfying routes in a road network with different travel distances.

[1] http://maps.google.com/
[2] http://www.mapquest.com/
[3] http://maps.yahoo.com/
[4] http://maps.bing.com/

User defined traveling rules can be formulated as sub-sequences of POI categories in MRPSR queries. Such sub-sequences (or partial sequence) exist inherently in many GIS applications or can be specified by users as external constraints. Therefore, MRPSR queries are useful in numerous fields such as automotive navigation systems, transportation planning, supply chain management, online Web mapping services, etc.

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. On the other hand, 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 gas station visit, then one may choose any one of the available gas stations.

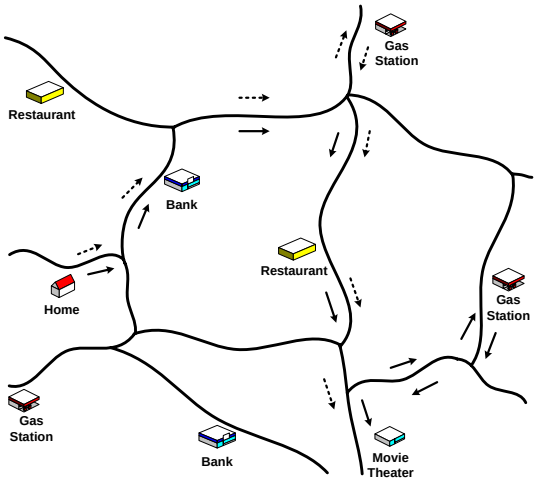


**Fig. 1** Two possible routes (solid and dashed arrows) of a MRPSR query.

## 1.2 Contribution

In this study we present the MRPSR query and provide three fast approximation algorithms which are designed to efficiently compute satisfying routes with the near-optimal travel distance in road networks. This paper is based on our earlier paper [5], in which all the proposed solutions and experiments are based on Euclidean distance in a vector space. In order to serve real-world GIS applications effectively, we extend all the route query algorithms to road networks with an extensive set of simulations in this paper. The contributions of our work are as follows:

- We formally define the Multi-rule Partial Sequenced Route (MRPSR) query and prove the MRPSR problem to be a member of the *NP*-complete class.
- By casting traveling rules into an activity-on-vertex network, we utilize topological sorting [13] to integrate traveling rules with multiple choices of POIs and study the solvability of MRPSR queries.

- We propose the Nearest Neighbor-based Partial Sequence Route query (NNPSR) algorithm. The NNPSR algorithm uses activity-on-vertex networks to guide the search to retrieve a near-optimal route satisfying all the traveling rules in road networks.
- We integrate NNPSR with the Light Optimal Route Discoverer (LORD) algorithm [27] to create NNPSR-LORD that further reduces the trip distance based on the NNPSR algorithm.
- We also design an Advanced A* Search-based Partial Sequence Route query (AASPSR($k$)) algorithm. AASPSR($k$) takes advantage of the location of the destination as well as traveling rules to generate an efficient trip plan in road networks.
- We compare the performance of NNPSR, AASPSR($k$) and NNPSR-LORD analytically.
- By using real and synthetic POI datasets, we compare experimentally the performance of NNPSR, AASPSR($k$), NNPSR-LORD and the LORD-based brute-force solution in the road network of California.

### 1.3 Paper Organization

The rest of the paper is organized as follows. The research problem is formally defined in Section 2. In Section 3 we introduce AOV networks. We elaborate on NNPSR, NNPSR-LORD, and AASPSR($k$) algorithms in Section 4. The experimental validation of our design is presented in Section 5. Section 6 surveys the related work. We conclude the paper with a discussion of future work in Section 7.

## 2 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 type. The definitions of the multi-rule partial sequenced route query and the partial sequence rules are introduced in Section 2.1. The properties of the multi-rule partial sequenced route query are discussed in Section 2.2. Section 2.3 presents the definition of the percentage of the constrained categories.

### 2.1 Problem Formulation

**Definition 1** Given $n$ disjoint sets of POI category $\{C_1, C_2, \ldots, C_n\}$, each containing a number of POIs in $R^2$, the MRPSR query is to search 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 route queries [18, 27], 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 partial sequence rule, defined as follows.

**Definition 2** 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 at a bank before going for grocery shopping and dinner. This constraint can be converted to the following two partial sequence rules:

1. $C_{Bank} \rightarrow C_{Supermarket}$
2. $C_{Bank} \rightarrow C_{Restaurant}$.

These two rules enforce that a bank should be visited before a supermarket and a restaurant on the trip, 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 [21]. The human natural language can be ambiguous and non-grammatical. The automatic translation requires to create 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 compatibility of a set of partial sequence rules can be defined as follows.

**Definition 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. When all the travel constraints are represented as a set of partial sequence rules, the original definition of the MRPSR query can be formulated as follows.

**Definition 4** Given a set of POI categories and a set of partial sequence rules, a MRPSR query is defined to return the route with the minimal total traveling distance that satisfies the order specified in each of the partial sequence rules.

## 2.2 Properties of The MRPSR Query

The following theorem shows that MRPSR query provides a unified framework that subsumes the well-known trip planning techniques, including the trip planning queries (TPQ) [18] and the optimal sequenced route (OSR) queries [27].

**Theorem 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 [18], 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 [27], 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 1, we obtain the following important property for the MRPSR query.

**Corollary 1** *The problem of the multi-rule partial sequence route query is NP-hard.*

*Proof* According to [18], the problem of the trip planning query is NP-hard. To show our problem of the multi-rule partial sequence route query is also NP-hard, we need to construct a polynomial transformation $f$ from the problem of the trip planning query to ours. Given an instance of the trip planning query, i.e., the optimal route between a pair of source and destination with a given set of POI types, we can easily transform it to a MRPSR query, which asks for the optimal route between exactly the same source/destination and the same set of POI types with no partial order between the types. This transformation is obviously polynomial. According to [7], it follows immediately that the problem of the multi-rule partial sequenced route query is NP-hard.

Corollary 1 implies that when the search space is large, it is advisable 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 1 and Corollary 1, if the set is empty, the search space will be large and the MRPSR query is NP-hard. However, if the rule specifies the total order of the categories, the MRPSR problem can be solved in polynomial time [27]. 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 Theorem 2 to see if a given set of rules will possibly lead to a solution. Theorem 2 shows the relationship between the solvability of a MRPSR query and the compatibility of a given set of rules.

**Theorem 2** *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 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 2 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 2 (i.e., the compatible set of rules implies the solvability of the corresponding MRPSR query) will also be true. According to Definition 3, if the partial sequence rules are compatible, then there must exist at least one total order of categories $< C_i >$ that satisfies the order specified in each of the rules. Let one of such orders be $\{C_1, C_2, \ldots, C_n\}$. Now since each category is not empty, we can arbitrarily pick one POI $p_x$ from each category $C_i$ to compose a route $\{p_1, p_2, \ldots, p_n\}$ which traverses through exactly one POI in each category and conforms with the given traveling rules. According to Definition 1, if there is only one such route, we have our answer. If not, the one with the minimal traveling distance will be what we want to retrieve. In Section 3, we will elaborate how to verify if a set of partial sequence rules is compatible.

### 2.3 Percentage of the Constrained Categories

**Definition 5** Given a MRPSR query, the *Percentage of the Constrained Categories (PCC)* is defined as the percentage of the number of categories included in the set of traveling rules over the total number of categories to be visited in the query.

PCC is used to measure the extent that a MRPSR query is constrained by traveling rules. According to the definition of PCC, the trip planning query (TPQ) [18] can be considered as a MRPSR query with a PCC of 0% while the optimal sequenced route (OSR) query [27] can be treated as a MRPSR query with a PCC of 100%.

## 3 Activity-on-Vertex Networks

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 Activity-On-Vertex (AOV) network [11]. The following theorem provides the relationship of an AOV network and the compatibility of the traveling rules.

**Theorem 3** *The partial sequence rules are compatible if and only if the corresponding AOV network is a directed acyclic graph.*

*Proof* Definition 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 [11], 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 1 lists the POI categories and partial sequence rules specified by an example MRPSR query $\mathbb{Q}$. The corresponding AOV network for $\mathbb{Q}$ 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 1** POI categories and partial sequence rules in an example MRPSR query $\mathbb{Q}$.

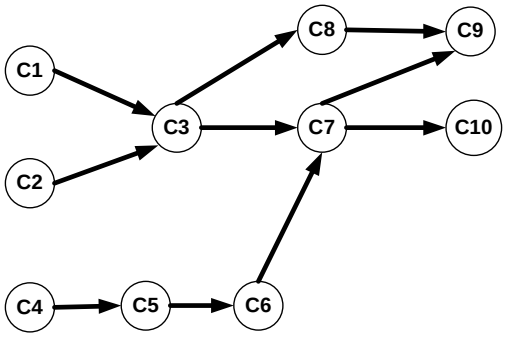


**Fig. 2** The AOV network of $\mathbb{Q}$ represents POI categories as vertices and prerequisites as edges.

After we represent all the partial sequence rules in a MRPSR query as an AOV network, providing that the AOV network is directed and acyclic, *Topological Order* (or *Topological Sorting*) can be used to generate a feasible complete ordering of POI categories which is compatible with every partial sequence rule in the MRPSQ query. 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. Each DAG has at least one *topological order*. The algorithm to find a *topological order* is as follows. 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. By repeating these two steps 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 [13]. 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. For supporting both cases, we keep a counter of the number of immediate predecessors for each vertex and represent the network by its adjacency lists. Then we can 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.

## 4 Algorithm Design

After having the AOV networks in hand, we can start to compute a trip plan satisfying all the traveling rules. In this section, we propose three approximate algorithms to answer a MRPSR query: the Nearest Neighbor-based Partial Sequenced Route (NNPSR) algorithm,

the Nearest Neighbor-based Partial Sequenced Route with Light Optimal Route Discoverer [27] (NNPSR-LORD) algorithm, and the Advanced A* Search-based Partial Sequenced Route (AASPSR($k$)) algorithm. NNPSR applies AOV networks to capture traveling rules and launches successive nearest neighbor queries to answer a given MRPSR query. NNPSR-LORD utilizes the Light Optimal Route Discoverer [27] to optimize the route obtained by NNPSR. In AASPSR($k$), as a hybrid scheme of NNPSR and ASPSR, distance heuristic functions are integrated with NNPSR to answer a MRPSR query. All of the proposed algorithms aim to find the near-optimal route which follows all of the traveling rules. Table 2 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 |
| $Q$ | The priority queue |
| $S$ | The starting point of a MRPSR query |
| $D$ | The destination of a MRPSR query |
| $q$ | The query point of a nearest neighbor query |
| $C_i$ | A POI category |
| $C_i.\mathbb{P}$ | All the POIs of a category |
| $L_{zero}$ | A list of AOV vertices with a zero count |
| $L_{route}$ | A list of the POI sequence of a trip plan |
| $P_{NN}$ | The query result of a nearest neighbor query |
| $Dist_E(x, y)$ | The Euclidean distance between points $x$ and $y$ |
| $Dist_N(x, y)$ | The network distance between points $x$ and $y$ |

**Table 2** Symbolic notations.

### 4.1 Nearest Neighbor Search in Road Networks

In practice, users usually move only in the underlying road networks rather than traveling freely through obstacles (e.g., buildings, rivers, etc.). Network distance computations and nearest neighbor queries in road networks have been well studied [12, 15, 20]. As the basic building block of our proposed algorithms, in this subsection, we briefly review how to answer a nearest neighbor query in road networks by the incremental network expansion approach [20].

Figure 3 demonstrates the nearest neighbor search by applying the incremental network expansion technique [20]. In Figure 3, the black point, $q$, stands for the query point, the white points, $A$, $B$, $C$, $D$, $E$, and $F$, denote road network conjunctions, the triangles, $P_1$, $P_2$, and $P_3$, represent POIs (which are in ascending order of their Euclidean distance to $q$), and the numbers symbolize the distance between two points. Incremental network expansion performs network expansion from $q$ and examines POIs in the order which they are encountered. To be specific, first, the road segment $CD$ that covers $q$ is found and all POIs on $CD$ are retrieved. Then a priority queue, $Q =< (C, 5), (D, 7) >$, is initiated. Since no POI is covered by $CD$, the node $C$ which is closest to $q$ is de-queued and its adjacent nodes, $A$ and $E$, are inserted into $Q$ with their accumulated distance from $q$, i.e., $Q =< (D, 7), (A, 9), (E, 10) >$. No POI is discovered on $CA$ and $CE$. Next, $D$ whose dis-

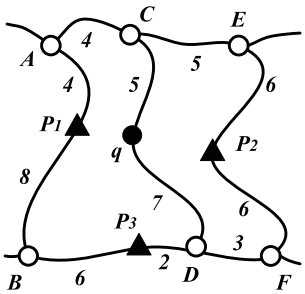


**Fig. 3** NN search by the incremental network expansion algorithm [20].

tance is closest to $q$ in current $Q$ is expanded and its adjacent nodes, $B$ and $F$, are en-queued. Then, we have $Q =< (A, 9), (E, 10), (F, 10), (B, 15) >$. Afterward, $P_3$ can be discovered on $DB$ with a distance of 9 while no POI is found on $DF$. This distance offers an upper bound to restrict the search space. Because the next node to expand is $A$ and the distance from $q$ to $A$ is already 9, which is no less than the upper bound, the algorithm terminates and returns $P_3$ as the nearest neighbor to $q$ with a network distance of 9.

### 4.2 Nearest Neighbor-based Partial Sequenced Route Algorithm

Here we devise a Nearest Neighbor-based Partial Sequence Route (NNPSR) query algorithm by utilizing both the $L_{zero}$ list and the nearest neighbor query (i.e., the incremental network expansion [20] based implementation) to generate a trip satisfying all the traveling rules. With NNPSR, we first search for the nearest POI to the query point $q$ (as the starting point) whose category is included in $L_{zero}$. The retrieved nearest POI $P_{NN}$ will be stored in a route list $L_{route}$ and the category of $P_{NN}$ (i.e., $P_{NN}.C$) 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 $P_{NN}$. The process will repeat until all the selected categories are contained in the route. The complete algorithm of NNPSR is formalized in Algorithm 1.

### 4.3 NNPSR with Light Optimal Route Discoverer Algorithm

Suppose a complete POI sequence to be visited is given, the Light Optimal Route Discoverer (LORD) algorithm [27] can guarantee to retrieve a route of minimum distance. Since we can obtain a complete POI sequence after each execution of the NNPSR algorithm, we can further optimize the trip by applying LORD on the POI sequence found by NNPSR. 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

**Algorithm 1** Nearest Neighbor-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:    while L_zero ≠ ∅ do
6:       ℙ = ∅
7:       for each C_i ∈ L_zero do
8:          ℙ = C_i.ℙ ∪ℙ
9:       end for
10:      Identify the road segment n_i n_j covering q.
11:      Find all the POIs in ℙ on n_i n_j.
12:      if If there exists at least one POI in ℙ on n_i n_j then
13:         Update p_NN with the POI P_k with the smallest Dist_N(q, P_k).
14:      else
15:         Q =< (n_i, Dist_N(q, n_i)), (n_j, Dist_N(q, n_j)) >
16:         De-queue the node n in Q with the smallest Dist_N(q, n)
17:         while Dist_N(q, n) < Threshold do
18:            for each non-visited adjacent node n_k of n do
19:               Find all the POIs in ℙ on the road segment n n_k.
20:               Update P_NN from the POI p′ in ℙ with the smallest network distance found so far
21:               Update Threshold with Dist_N(q, p′)
22:               En-queue (n_k, Dist_N(q, n_k)) in Q
23:            end for
24:            De-queue the node n in the updated Q with the smallest Dist_N(q, n)
25:         end while
26:      end if
27:      q = P_NN
28:      L_route = L_route ∪ P_NN
29:      Remove P_NN.C from L_zero
30:      Update 𝔸 and L_zero
31:   end while
32:   return L_route
33: else
34:   Report cycles in ℝ
35: end if
```

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 NNPSR-LORD solution outperforms the original NNPSR algorithm in terms of route distance. More detailed performance evaluations are presented in Section 5.

### 4.4 Advanced 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 4. In Figure 4, $S$ and $D$ denote the start point and destination of the trip. Suppose we have a traveling rule which can be denoted as **Bank** $\rightarrow$ **Restaurant**. We will find that the dashed route returned by NNPSR is much longer than another feasible trip (the solid route) which considers the

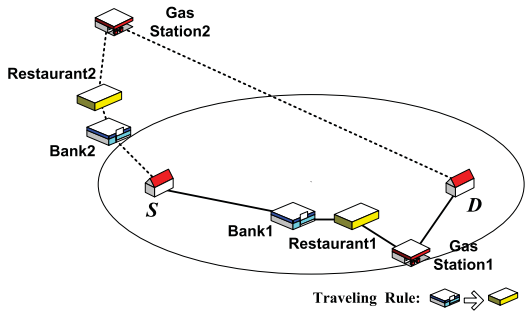


**Fig. 4** Two trips generated by NNPSR (the dashed route) and ASPSR (the solid route) with the traveling rule **Bank** → **Restaurant**.

location of the destination. Therefore, another approach is to limit the trip planning within a range defined by $S$ and $D$ (e.g., an ellipse whose two focal points are $S$ and $D$).

The A* search based Partial Sequenced Route (ASPSR) algorithm considers the location of the destination in its heuristic function. Similar to the *admissible heuristic* of the A* algorithm [23], in ASPSR, we retrieve the POI $p$ with the minimum cost of $Dist_E(S, p) + Dist_E(p, D)$ in each category included in $L_{zero}$. Afterward the POI $p$ with the lowest cost will be added into the route list $L_{route}$ and the category of $p$ will be withdrawn from $L_{zero}$. Then both $\mathbb{A}$ and $L_{zero}$ will be updated and the location of $p$ is set as the new query point. The process will reiterate until all the user selected categories are covered.

However, there could be roundabout ways when we plan a trip by ASPSR. Consider the example as shown in Figure 5. Because the POIs which are closer to the major axis of the ellipse ($S$ and $D$ are the two focal points) have a lower distance cost, *Bank1*, *Restaurant1* and *Gas Station1* will be picked sequentially in ascending order of their costs. Consequently, a detour will occur where the user has to travel far away from from $D$ to visit *Gas Station1* and *Restaurant1* before reaching $D$ at last. Therefore, we need to improve ASPSR to solve the aforementioned problem.

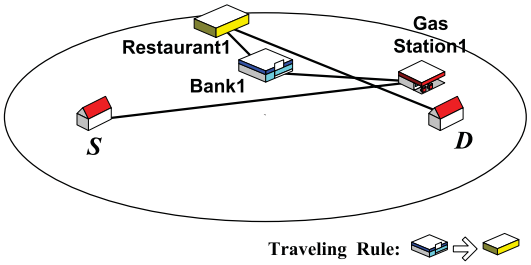


**Fig. 5** An illustration of trip search by ASPSR with the traveling rule **Bank** → **Restaurant**.

The improved version of ASPSR is named as the Advanced A* Search-based Partial Sequenced Route query (AASPSR) algorithm. In the following sections of this paper, we use AASPSR($k$) to denote our AASPSR algorithm with parameter $k$ to specify the number of POIs we retain for each category. AASPSR($k$) can be considered as a hybrid scheme to combine ASPSR and NNPSR. To be specific, AASPSR($k$) first computes $C_i.\mathbb{P}$* for each

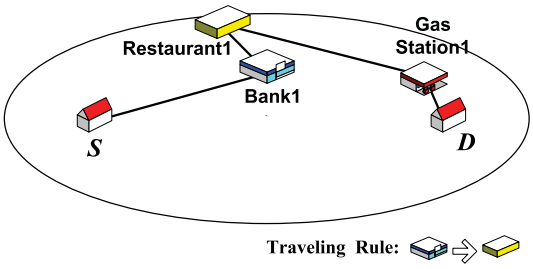


**Fig. 6** An illustration of trip search by AASPSR(1) with the traveling rule **Bank** → **Restaurant**.

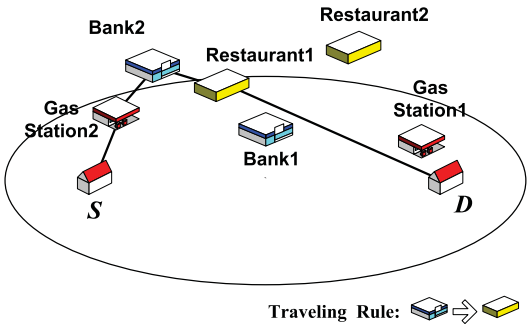


**Fig. 7** An illustration of trip search by AASPSR(2) with the traveling rule **Bank** → **Restaurant**.

category $C_i$ in $\mathbb{C}$ such that every POI in $C_i.\mathbb{P}$* is a POI with the top-$k$ minimum traveling distance sum from $S$ to $D$ in $C_i$. In particular, if $k = 1$, only one POI with the shortest traveling distance sum from $S$ to $D$ for $C_i$ is added to $C_i.\mathbb{P}$*. On the contrary, if $k = \infty$, then all the POIs on the underlying road network will be included in $C_i.\mathbb{P}$* for each $C_i$. After $C_i.\mathbb{P}$* has been generated for each category, we launch NNPSR to generate a route only on these selected POIs in each $C_i.\mathbb{P}$*. Starting with $S$, we search for the nearest POI $p$ in $C_i.\mathbb{P}$* ($C_i \in L_{zero}$). Afterward, $p$ is inserted into $L_{route}$ and the location of $p$ is used as the query point of the following NN query. Next we remove the category of $p$ from $L_{zero}$ and recompute the adjacency list. The whole process will repeat until $L_{zero}$ becomes empty. The complete algorithm of AASPSR($k$) is illustrated in Algorithm 2. For comparison purpose, the trips generated by AASPSR(1) and by AASPSR(2) are demonstrated in Figure 6 and Figure 7, respectively.

### 4.5 Comparison between NNPSR, NNPSR-LORD and AASPSR

In order to analyze the performance of the three aforementioned algorithms, we have the following two definitions:

**Definition 6** A MRPSR query is called to be a *strictly constrained query* if its PCC value is relatively high.

**Definition 7** A MRPSR query is called to be a *loosely constrained query* if its PCC value is relatively low.

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

Set $L_{route}$ = $\emptyset$ and $Q$ = $S$
Integrate all elements in $\mathbb{R}$ into an AOV adjacency list $\mathbb{A}$ and put all vertices with zero count in $L_{zero}$
**if** The AOV network is a DAG **then**
  Add all elements of $\mathbb{C} \setminus \mathbb{A}$ into $L_{zero}$
  **for** each category $C_i \in \mathbb{C}$ **do**
    **for** each POI $p_j \in C_i.\mathbb{P}$ **do**
      $Cost_j = Dist_E(S, p_j) + Dist_E(p_j, D)$
    **end for**
    Sort POI $p_j \in C_i$ in ascending order based on $Cost_j$ and add the top-$k$ $p_j \in C_i$ with the minimum $Cost_j$ into $C_i.\mathbb{P}$*
  **end for**
  {ASPSR search done, the subsequent NNPSR search starts}
  **while** $L_{zero} \neq \emptyset$ **do**
    $\mathbb{P}$ = $\emptyset$
    **for** each $C_i \in L_{zero}$ **do**
      $\mathbb{P} = \mathbb{P} \cup C_i.\mathbb{P}*$
    **end for**
    Identify the road segment $n_i n_j$ covering $q$.
    Find all the POIs in $\mathbb{P}$ on $n_i n_j$.
    **if** If there exists at least one POI in $\mathbb{P}$ on $n_i n_j$ **then**
      Update $P_{NN}$ with the POI $P_k$ with the smallest $Dist_N(q, P_k)$.
    **else**
      $Q =< (n_i, Dist_N(q, n_i)), (n_j, Dist_N(q, n_j)) >$
      De-queue the node $n$ in $Q$ with the smallest $Dist_N(q, n)$
      **while** $Dist_N(q, n) < Threshold$ **do**
        **for** each non-visited adjacent node $n_k$ of $n$ **do**
          Find all the POIs in $\mathbb{P}$ on the road segment $nn_k$.
          Update $P_{NN}$ from the POI $p'$ in $\mathbb{P}$ with the smallest network distance found so far
          Update $Threshold$ with $Dist_N(q, p')$
          En-queue $(n_k, Dist_N(q, n_k))$ in $Q$
        **end for**
        De-queue the node $n$ in the updated $Q$ with the smallest $Dist_N(q, n)$
      **end while**
    **end if**
    $q$ = $P_{NN}$
    $L_{route} = L_{route} \cup P_{NN}$
    Remove $P_{NN}.C$ from $L_{zero}$
    Update $\mathbb{A}$ and $L_{zero}$
  **end while**
  **return** $L_{route}$
**else**
  Report cycles in $\mathbb{R}$
**end if**

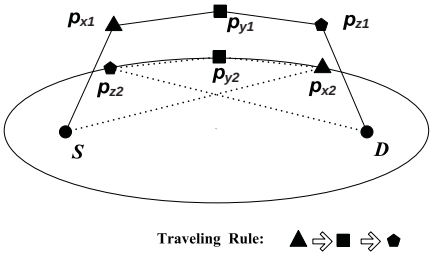


**Fig. 8** An example where AASPSR generates a longer route than NNPSR.

| Query Features | NNPSR | AASPSR($k$) | NNPSR-LORD |
|---|---|---|---|
| Strictly constrained MRPSR (e.g., OSR) | ✓ | | ✓ |
| Loosely constrained MRPSR (e.g., TPQ) | | ✓ | ✓ |
| Further optimized route | | | ✓ |
| Time sensitive applications | ✓ | ✓ | |

**Table 3** The feature comparison among proposed algorithms.

**Theorem 4** *Given a MRPSR query, AASPSR does not necessarily return a shorter route than NNPSR.*

*Proof* We can prove it by a counter-example as shown in Figure 8. In Figure 8, the triangle, rectangle, and pentagon each represents a different category of POIs, and $S$ and $D$ denote the start point and destination of the trip, respectively. Additionally, we have a traveling rule denoted as **triangle** → **rectangle** → **pentagon**. Consequently, 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 existence of traveling rules leads to a roundabout route in AASPSR, which only chooses the POIs inside the ellipse. Therefore, AASPSR performs worse than NNPSR in terms of route distance in this scenario.

In each category, AASPSR($k$) only chooses the $k$ POIs which have the minimum traveling distance sum from the start point to the destination for the subsequent NN search. Consequently, if there are too many traveling rules imposed, i.e., there are many restrictions on the category order, AASPSR($k$) will be very likely to generate a roundabout route. Therefore, for strictly constrained MRPSR queries, which have a higher PCC, NNPSR usually returns a shorter route than AASPSR($k$). On the other hand, for loosely constrained MRPSR queries, which hold a lower PCC, such as TPQ (PCC equals zero), AASPSR($k$) usually outperforms NNPSR in terms of route distance.

By taking advantage of the LORD algorithm [27], NNPSR-LORD can further shorten the length of the route retrieved by the NNPSR algorithm. Consequently, NNPSR-LORD can consistently outperform NNPSR in terms of route distance. However, as far as the response time is concerned, NNPSR-LORD, compared with NNPSR and AASPSR, needs much more computational time, especially in road networks. This is due to its extensive usage of network distance functions. Therefore, NNPSR-LORD is only applicable to non-time sensitive applications. A complete comparison among proposed algorithms is illustrated in Table 3.

## 5 Experimental Validation

### 5.1 Experimental Setup

The experimental results are reported in this section. We implemented the NNPSR, NNPSR-LORD, and AASPSR($k$) algorithms in road networks to evaluate their performances with respect to the returned *route distance* and the *response time* to generate the corresponding routes. Besides, to highlight the benefits of our three approximate approaches, we used the LORD-based brute-force solution as the baseline, which applies LORD [27] on each possible permutation of all categories to get the optimal sequenced route for each particular category sequence. For each run of the LORD-based brute-force solution, we compared

the distances of all the possible optimal sequenced routes and recorded the minimum route distance and overall response time.

As we discussed in Section 4, AASPSR($k$) will exhibit more characteristics of NNPSR when $k$ increases (in particular, AASPSR($k$) degrades to NNPSR if $k = \infty$) and show more characteristics of AASPSR(1) when $k$ decreases. Therefore, in this section we only focused on the performance of AASPSR(1) (the terms AASPSR and AASPSR(1) are used interchangeably in this section). 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, and the number of query categories. PCC describes the percentage of the number of categories involved in traveling rules over the total number of categories to be visited in a query. The average category cardinality is the average number of POIs over all categories while the number of query categories is the total number of categories to be visited in the query. For each result of the NNPSR, AASPSR and NNPSR-LORD algorithms, 100 MRPSR queries were launched with a starting point and a destination generated randomly on the road network, and then the results were averaged. All the experiments were conducted on a Linux machine with an Intel Core2 Quad CPU (Q9400 2.66GHz) and 4GB memory.

Fig.9(a) Road network of California.

Fig.9(b) California points of interest distribution.

**Fig. 9** Real Datasets from the state of California.

### *5.1.1 Road Network Dataset*

To investigate the performance of our proposed algorithms for road networks, first we obtained the road network dataset of the state of California from [1]. As shown in Figure 9(a), the road network of California contains 21,048 nodes and 22,830 edges. Each node is described with a tuple of $\langle Node\_ID, Longitude, Latitude\rangle$ and each edge is represented by a tuple of $\langle Edge\_ID, Start\_Node\_ID, End\_Node\_ID, L2\_Distance\rangle$.

### *5.1.2 California Point of Interest Dataset*

We collected the points of interest of the state of California from [2] as shown in Figure 9(b). This California dataset has 63 different categories, including airports, hospitals, schools, populated places, etc., which correspond to more than 100,000 points of interest. Each category exhibits a distinct density and distribution. Each point of interest is represented as a tuple of $\langle Category\_Name, Longitude, Latitude\rangle$. The cardinalities of all the categories used in our research is shown in Table 4.

| Category | Size |
|---|---|
| Airport | 995 |
| Area | 287 |
| Bar | 278 |
| Building | 4110 |
| Church | 7680 |
| Hospital | 835 |
| Locale | 13481 |
| Park | 6728 |
| School | 11173 |
| Populated place | 6900 |
| Summit | 5594 |
| Valley | 7596 |

**Table 4** The category cardinalities used in our California dataset.

To merge the points of interest in the real dataset with the road network, we adopted the map format where each point of interest was at first mapped to a point on an edge and then represented as the distance of this point to the start node of that edge.

#### *5.1.3 Synthetic Point of Interest Datasets*

To control different cardinalities and distributions of categories, we also applied synthetic datasets in our experiments. We generated different numbers of points of interest for different datasets and uniformly distributed the points of interest on the edges of the California road networks.

#### *5.1.4 Traveling Rules*

Without loss of generality, for the real dataset, rules were generated between Building and Populated place, Church and Hospital, and Locale and Park, i.e., rules can be represented as follows: *Building* $\rightarrow$ *Populated place*, *Church* $\rightarrow$ *Hospital* and *Locale* $\rightarrow$ *Park*. For the synthetic datasets, rules were generated between any two arbitrary categories at random.

### 5.2 Effect of the Percentage of the Constrained Categories

In our first experiment, we varied the Percentage of the Constrained Categories (PCC) to investigate the performance of NNPSR, AASPSR, NNPSR-LORD and the LORD-based brute-force solution in terms of the route distance and response time. Since our proposed MRPSR query subsumes the TPQ and OSR queries, the MRPSR queries exhibit the characteristics of TPQ queries when PCC decreases and the characteristics of OSR queries when PCC increases. Our results are based on the California POI dataset and the synthetic POI datasets, respectively. In the synthetic POI dataset, the average category cardinality is 6000. Furthermore, we assumed that the number of query categories is 6. Figure 10 illustrates the relationship between route distance and PCC for NNPSR, AASPSR, NNPSR-LORD, and LORD-based brute-force algorithms.

In Figure 10 route distance increases with the increase of PCC for all the algorithms. This is because with a higher PCC, there will be more restrictions on the order of the categories, which leads to a longer route. Note that the route distance of AASPSR changes

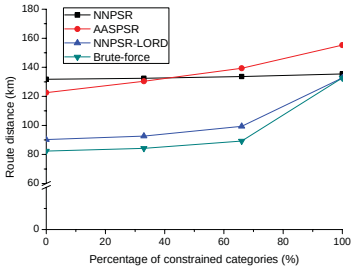


Fig.10(a) California dataset

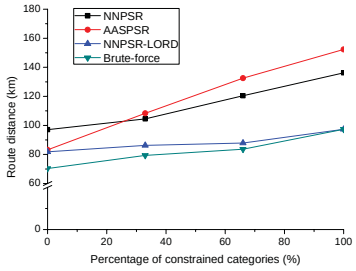


Fig.10(b) Synthetic dataset

**Fig. 10** Route distance of NNPSR, AASPSR, NNPSR-LORD, and LORD-based brute-force as a function of PCC.

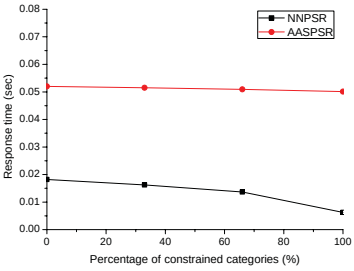


Fig. 11(a) NNPSR and AASPSR (California dataset)

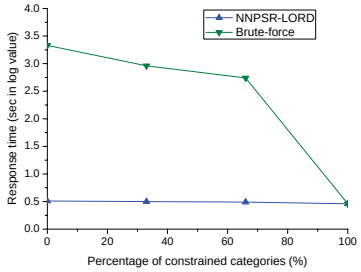


Fig. 11(b) NNPSR-LORD and LORD-based brute-force (California dataset)

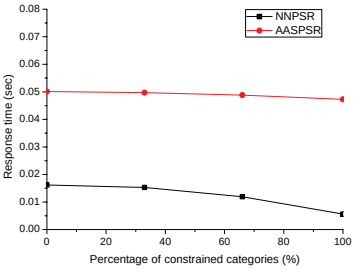


Fig. 11(c) NNPSR and AASPSR (synthetic dataset)

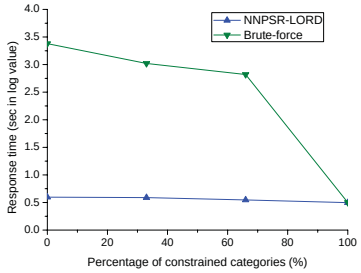


Fig. 11(d) NNPSR-LORD and LORD-based brute-force (synthetic dataset)

**Fig. 11** Response time of NNSPR, AASPSR, NNPSR-LORD, and LORD-based brute-force as a function of PCC

remarkably against PCC in contrast to NNPSR and NNPSR-LORD. The lower PCC is, the better AASPSR works compared with NNPSR. With a higher PCC, the route distance of AASPSR increases dramatically. In other words, AASPSR is only suitable for planning a trip with a low PCC, such as TPQ (PCC equals zero). This is because AASPSR only picks a single POI in each category for the subsequent NN search. Consequently, given a higher

PCC (there are more restrictions on the category order), a longer route may be needed to traverse all the POIs picked up in the first step. In addition, 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 11 plots the response time against PCC for NNPSR, AASPSR, NNPSR-LORD, and the LORD-based brute-force method. Notice that in Figure 11(b) and (d), we plotted the relationship by using the *log* values of the response time instead of the original values because the response times of NNPSR-LORD and LORD-based brute-force solutions are in different orders of magnitude. First, as shown in Figure 11, all our proposed algorithms significantly reduce the response time compared with the LORD-based brute-force solution. To be specific, NNPSR and AASPSR are more than 10000 times faster in some cases than the LORD-based brute-force method while NNPSR-LORD is more than 100 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.

## 5.3 Effect of the Average Category Cardinality

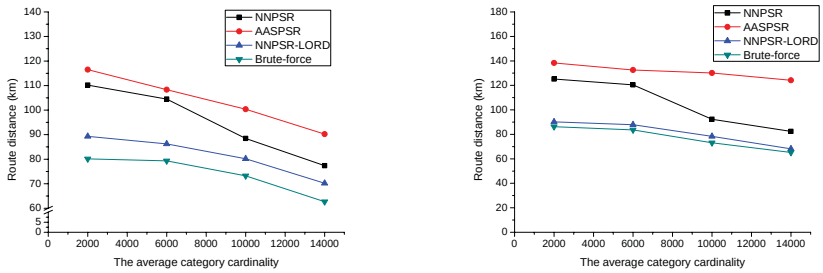


Fig.12(a) PCC = 33%

Fig.12(b) PCC = 66%

**Fig. 12** Route distance of NNSPR, AASPSR, NNPSR-LORD, and LORD-based brute-force as a function of the average category cardinality.

Next, we studied the effect of the average category cardinality by varying the cardinality from 2000 to 14000 using synthetic datasets. Here we assumed that the number of query categories is 6. Figure 12 shows the route distances of NNPSR, AASPSR, NNPSR-LORD, and the LORD-based brute-force method where PCC equals 33% and 66%, respectively. As Figure 12 shows, the route distance decreases for each algorithm with the increase of the average category cardinality. The reason is that a denser distribution of a category will lead to more POI choices, which result in a lower probability of detours. Notice that AASPSR has relatively poor performance with either a PCC of 33% or 66% because AASPSR outperforms NNPSR in terms of route distance only if PCC is very low. The PCC of 33% or 66% is large enough to deteriorate the performance of AASPSR. Figure 13 shows the response time for the above algorithms. As Figure 13 demonstrates, the response time of each algorithm increases with the enlargement of the average category cardinality. This is due to a higher density of each POI category which elongates the computational time.

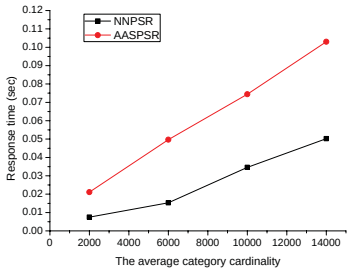


Fig. 13(a) NNPSR and AASPSR (PCC = 33%)

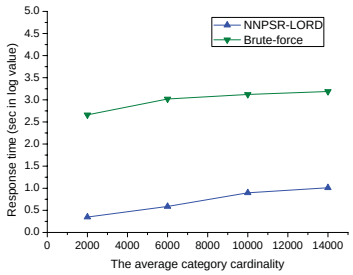


Fig. 13(b) NNPSR-LORD and LORD-based brute-force (PCC = 33%)

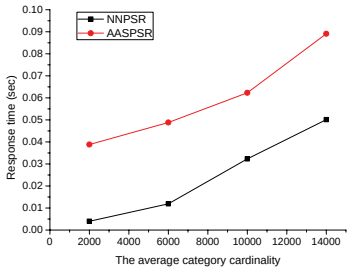


Fig. 13(c) NNPSR and AASPSR (PCC = 66%)

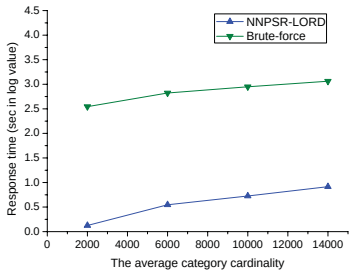


Fig. 13(d) NNPSR-LORD and LORD-based brute-force (PCC = 66%)

**Fig. 13** Response time of NNSPR, AASPSR, NNPSR-LORD, and LORD-based brute-force as a function of the average category cardinality.

## 5.4 Effect of the Number of Query Categories

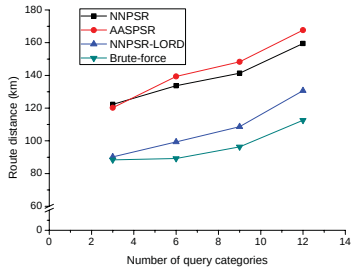


Fig.14(a) California dataset

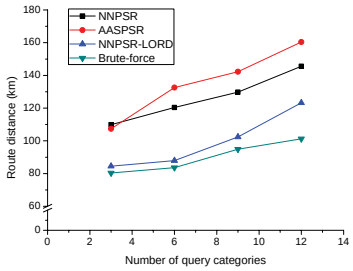


Fig.14(b) Synthetic dataset

**Fig. 14** Route distance of NNPSR, AASPSR, NNPSR-LORD, and LORD-based brute-force as a function of the number of query categories.

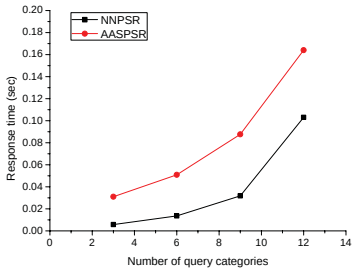


Fig. 15(a) NNPSR and AASPSR (California dataset)

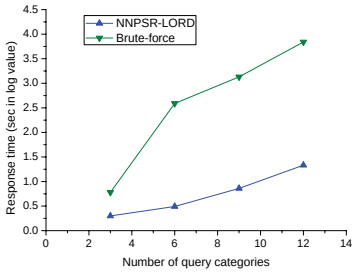


Fig. 15(b) NNPSR-LORD and LORD-based brute-force (California dataset)

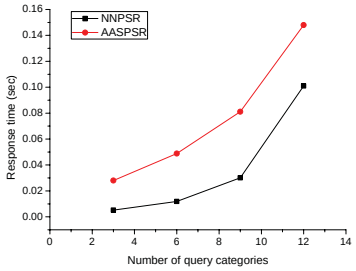


Fig. 15(c) NNPSR and AASPSR (synthetic dataset)

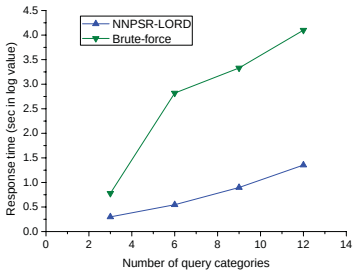


Fig. 15(d) NNPSR-LORD and LORD-based brute-force (synthetic dataset)

**Fig. 15** Response time of NNSPR, AASPSR, NNPSR-LORD, and LORD-based brute-force as a function of the number of query categories.

In this subsection, we changed the number of query categories to 3, 6, 9 and 12 to investigate the impact of the number of query categories on the performance of NNPSR, AASPSR, NNPSR-LORD, and the LORD-based brute-force method. Our experiments are based on the California POI and the synthetic POI datasets, respectively. In the synthetic POI dataset, the average category cardinality is assumed to be 6000. In addition, we assume that PCC equals 66%. Figures 14 and 15 illustrate the experimental results. As shown in Figure 14, when the number of query categories increases, the route distance of each algorithm extends dramatically. This is because with an increasing number of categories to be visited, there will be more POIs to be traversed in a trip. Notice that the number of query categories has a significant impact on whether AASPSR outperforms NNPSR in terms of route distance. For three-category cases, AASPSR returns a shorter route distance than NNPSR. On the contrary, for the 6, 9, or 12 category cases, AASPSR reports a longer route than NNPSR. The reason is that fewer categories will lead to a lower probability for a detour to occur when AASPSR tries to traverse all the POIs selected in its first step. On the other hand, as Figure 15 shows, when the number of query categories increases, the response time prolongs accordingly. The reason is that all the algorithms need more time to compute more categories to answer a MRPSR query. In particular, AAPSR consistently needs more response time than NNPSR, irrespective of the number of query categories.

## 6 Related Work

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

### 6.1 Nearest Neighbor Query

The nearest neighbor query is a very important query type for supporting GIS applications. With the R-tree family [3, 9, 26] of spatial indices, depth first search (DFS) [22] and best first search (BFS) [10] 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. [12] proposed data models and graph representations for NN queries in road networks and designed corresponding solutions. Papadias et al. [20] 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 [15]. Sharifzadeh et al. [28] extended the Voronoi diagram based approach for spatial data streams by using approximate Voronoi cell computation. On the contrary, in order to speed up the NN search, Samet et al. [25] proposed a solution to explore the entire spatial network by pre-computing the shortest paths between all the vertices in the network and using a shortest path quadtree to capture spatial coherence. By employing their approach, the shortest paths between various vertices can be computed only once to answer different NN queries on a given spatial network. However, the above pre-computation based approaches suffer from high overhead and adapt poorly to network updates. To overcome this shortcoming, Lee et al. [17] presented an efficient and flexible query framework, *ROAD*, based on *search space pruning* by using *shortcuts* for accelerating network traversals and *object abstracts* for guiding traversals.

### 6.2 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. [27]. 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 (sequence) imposed on all the POI types. In [29], a pre-computation approach was provided to answer OSR query by taking advantage of a family of AW-Voronoi diagrams for different POI types. However,

because the searches in [29] are based on NN queries, the authors need to investigate the performance of their method in terms of the traveling distance besides the response time. A multi-type nearest neighbor (MTNN) query solution was proposed in [19] 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. MTNN can be treated as an extended solution of OSR by exploiting a page-level upper bound. On the contrary, Li et al. [18] designed solutions for another new query type – Trip Planning Queries (TPQ). With TPQ, the user specifies a set of POI types and asks for the optimal route from her starting location to a specified destination which passes through exactly one POI in each POI type. Notice that compared to a OSR query, there is no order imposed on the types of POIs to be visited in a TPQ query. Terrovitis et al. [34] 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. Tian et al. [32] proposed skyline path queries in road networks based on multiple route search criteria (i.e., shortest traveling distance and shortest traveling time). By taking into account the probability that each POI type satisfies the user's particular need, an interactive approach was proposed in [14]. In order to capture the characteristics of ever-changeling road networks, Tian et al. [33] proposed the continuous min-cost path query and presented a system, *PathMon*, to monitor min-cost routes in dynamic road networks. However, all the aforementioned solutions cannot support MRPSR queries.

Another related problem to MRPSR is the sequential ordering problem (SOP) [6] 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 considered by our solution.

## 7 Conclusions

Geographic information systems are getting increasingly sophisticated and route queries with traveling rules represent a significant class of spatial queries. Existing solutions only focus on trips with a complete POI category sequence or without any sequence. However, GIS users usually want to set a number of traveling preferences when they plan their trips. In this paper we propose the MRPSR query and design three fast approximation algorithms to efficiently compute routes which can fulfill all the traveling rules with a near-optimal travel distance based on the underlying road networks. With extensive simulations, we show that our techniques can generate satisfying trips which are very close to the shortest routes with remarkable short response time. For future work, we plan to extend our algorithms to support dynamic road networks in which traffic information (e.g., travel time, traffic congestion, etc.) is incrementally becoming available as a data stream.

**Acknowledgements** This research has been funded in part by the National Science Foundation grants CNS-0831502 (CT), CNS-0855251 (CRI).

## References


1. Digital Chart of the World Server. http://www.maproom.psu.edu/dcw/.
2. U.S. Geological Survey. http://www.usgs.gov/.
3. Beckmann N, Kriegel HP, Schneider R and Seeger B (1990) The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In: Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, pp 322–331
4. Beeri C, Kanza Y, Safra E, Sagiv Y (2004) Object Fusion in Geographic Information Systems. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB), pp 816–827
5. Chen H, Ku WS, Sun MT, Zimmermann R (2008) The Multi-rule Partial Sequenced Route Query, In: Proceedings of 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp 10
6. Escudero LF (1988) An Inexact Algorithm for the Sequential Ordering Problem. European Journal of Operational Research 37(2):236–249
7. Garey MR, Johnson DS (1990) Computers and Intractability A Guide to the Theory of NP-Completeness. W. H. Freeman
8. George B, Kim S, Shekhar S (2007) 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), pp 460–477
9. Guttman A (1984) R-Trees: A Dynamic Index Structure for Spatial Searching. In: SIGMOD'84, Proceedings of Annual Meeting, pp 47–57
10. Hjaltason GR, Samet H (1999) Distance Browsing in Spatial Databases. ACM Trans. Database Syst 24(2):265–318
11. Horowitz E, Sahni S, Anderson-Freed S (1993) Fundamentals of Data Strucures in C. W. H. Freeman
12. Jensen CS, Kolárvr J, Pedersen TB, Timko I (2003) Nearest Neighbor Queries in Road Networks. In: Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS), pp 1–8
13. Kahn AB (1962) Topological Sorting of Large Networks. Commun ACM 5(11):558–562
14. Kanza Y, Levin R, Safra E, Sagiv Y (2009) An Interactive Approach to Route Search. In: Proceedings of 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp 408-411
15. Kolahdouzan MR, Shahabi C (2004) Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In: Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), pp 840–851
16. Ku WS, Zimmermann R, Wang H, Wan CN (2005) Adaptive Nearest Neighbor Queries in Travel Time Networks. In: Proceedings of the 13th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS), pp 210–219
17. Lee CK, Lee WC, Zheng B (2009) Fast Object Search on Road Networks. In: Proceeding of 12th International Conference on Extending Database Technology, pp 1018-1029
18. Li F, Cheng D, Hadjieleftheriou M, Kollios G, Teng SH (2005) On Trip Planning Queries in Spatial Databases. In: Proceedings of the 9th International Symposium on Advances in Spatial and Temporal Databases (SSTD), pp 273–290
19. Ma X, Shekhar S, Xiong H, Zhang P (2006) Exploiting a Page-Level Upper Bound for Multi-Type Nearest Neighbor Queries. In: Proceedings of the 14th ACM International Symposium on Geographic Information Systems (ACM-GIS), pp 179–186
20. Papadias D, Zhang J, Mamoulis N, Tao Y (2003) Query Processing in Spatial Network Databases. In: Proceedings of the 29th International Conference on Very Large Data Bases (VLDB), pp 802–813
21. Reddy R (1996) To Dream the Possible Dream. Commun ACM 39(5):105–112
22. Roussopoulos N, Kelley S, Vincent F (1995) Nearest Neighbor Queries. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, pp 71–79
23. Russell SJ, Norvig P (2002) Artificial Intelligence: A Modern Approach. Prentice Hall
24. Samet H (2001) Issues, Developments, and Challenges in Spatial Databases and Geographic Information Systems (gis). In: Proceedings of the Ninth ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS), p 1
25. Samet H, Sankaranarayanan J, Alborzi H (2008) Scalable Network Distance Browsing in Spatial Databases. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp 43-54
26. Sellis TK, Roussopoulos N, Faloutsos C (1987) The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In: Proceedings of 13th International Conference on Very Large Data Bases (VLDB), pp 507–518
27. Sharifzadeh M, Kolahdouzan MR, Shahabi C (2008) The Optimal Sequenced Route Query. The VLDB Journal 17(4), pp 765 –787

28. Sharifzade M, Shahabi C (2009) Approximate Voronoi Cell Computation on Spatial Data Streams. VLDB Journal 18(1) pp 57-75
29. Sharifzadeh M, Shahabi C (2008) Processing Optimal Sequenced Route Queries Using Voronoi Diagrams. GeoInformatica (12)4, pp 411-433
30. Shekhar S, Coyle M, Goyal B, Liu DR, Sarkar S (1997) Data Models in Geographic Information Systems. Commun ACM 40(4):103–111
31. Tao Y, Papadias D, Shen Q (2002) Continuous Nearest Neighbor Search. In: Proceedings of 28th International Conference on Very Large Data Bases (VLDB), pp 287–298
32. Tian Y, Lee CK, Lee WC (2009) Finding Skyline Paths in Road Networks. In: Proceedings of 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp 444-447
33. Tian Y, Lee CK, Lee WC (2009) Monitoring Minimum Cost Paths on Road Networks. In: Proceedings of 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp 217-226
34. Terrovitis M, Bakiras S, Papadias D, Mouratidis K (2005) Constrained Shortest Path Computation. In: Proceedings of the 9th International Symposium on Advances in Spatial and Temporal Databases (SSTD), pp 181–199
35. Zhang J, Zhu M, Papadias D, Tao Y, Lee DL (2003) Location-based Spatial Queries. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp 443–454