Top k Optimal Sequenced Route Query with POI Preferences

Huaijie Zhu^{1,2} · Wenbin Li¹ · Wei Liu¹ · Jian Yin¹ · Jianliang Xu³

Received: 3 June 2021 / Revised: 17 September 2021 / Accepted: 13 January 2022 / Published online: 3 February 2022
© The Author(s) 2022

Abstract

The optimal sequenced route (OSR) query, as a popular problem in route planning for smart cities, searches for a minimum-distance route passing through several POIs in a specific order from a starting position. In reality, POIs are usually rated, which helps users in making decisions. Existing OSR queries neglect the fact that the POIs in the same category could have different scores, which may affect users' route choices. In this paper, we study a novel variant of OSR query, namely *Rating Constrained Optimal Sequenced Route query (RCOSR)*, in which the rating score of each POI in the optimal sequenced route should exceed the query threshold. To efficiently process RCOSR queries, we first extend the existing TD-OSR algorithm to propose a baseline method, called *MTDOSR*. To tackle the shortcomings of MTDOSR, we try to design a new RCOSR algorithm, namely *Optimal Subroute Expansion (OSE) Algorithm*. To enhance the OSE algorithm, we propose a *Reference Node Inverted Index (RNII)* to accelerate the distance computation of POI pairs in OSE and quickly retrieve the POIs of each category. To make full use of the OSE and RNII, we further propose a new efficient RCOSR algorithm, called *Recurrent Optimal Subroute Expansion (ROSE)*, which recurrently utilizes OSE to compute the current optimal route as the guiding path and update the distance of POI pairs to guide the expansion. Then, we extend our techniques to handle a variation of RCOSR query, namely *RCkOSR* query. The experimental results demonstrate that the proposed algorithm significantly outperforms the existing approaches.

Keywords Optimal sequenced route · Route query · Point of interest (POI) · Preference

1 Introduction

With the ever-growing popularity of smartphones and other location-based services, various route queries have been studied to cater to users' different needs. Among these route queries, the optimal sequenced route (OSR) query has received significant research momentum in recent years [22]. It is designed to find the optimal route passing through a sequence of points of interest (POIs) of specific categories (e.g., gas stations, restaurants, and shopping malls) in a particular order. An example of the OSR query in a road network is shown in Fig. 1.

The example shows four POI categories, where $p_j^S (j = 1, 2, 3, 4)$ denotes POIs of supermarkets (represented by squares), $p_j^R (j = 1, 2, 3)$ denotes POIs of restaurants (represented by rhombus), and $p_j^G (j = 1, 2, 3)$ denotes POIs of gas stations (represented by triangle), $p_j^H (j = 1, 2)$ denotes the hotels (represented by pentagon). There is a number denoting its corresponding rating score, for example, $\langle p_j^S, 70 \rangle$ means the rating score of p_j^S is 70. The black circles denote the nodes in the road network. Given a user u_1 located at v_1 , she wants to pass through a sequence of POIs (e.g., restaurant, supermarket) to arrive at the destination node v_{16} ; this OSR query returns a route $\{v_1, p_1^S, p_1^R, v_{16}\}$ ¹ with the cost of 55.

The OSR query is first studied by Sharifzadeh et al. [21, 22], followed up by a number of variants [4, 5, 14, 16, 21, 26]. However, these prior works assume that the POIs in the same category have the same preference (i.e., they are rated with the same score). In the works [3, 7], the weight of POIs is considered as a factor in route cost functions. But in

¹ Between two POIs, the shortest path is used in default, e.g., $\{p_1^S, v_3, p_1^R\}$ is for p_1^S, p_1^R .

✉ Huaijie Zhu
zhuhuaijie@mail.sysu.edu.cn
Jianliang Xu
xujl@comp.hkbu.edu.hk

¹ Sun Yat-Sen University, Guangzhou, China

² Laboratory of Big Data Analysis and Processing, Guangzhou, China

³ Hong Kong Baptist University, Kowloon Tong, China

Fig. 1 A road network with rated POIs

real applications, the rating scores of POIs in the same category could be different, which affects users' route choices. That is, users usually prefer to visit the POIs with a high rating score, while those having a rating score lower than their expectations will not be taken into consideration. For different categories of POIs, the expected rating scores are also different. Take restaurants and gas stations for example. For most customers, they expect a higher rating score for a restaurant than a gas station, but the exact score threshold depends on different individuals.

Motivated by this, we propose a new OSR query, namely *Rating Constrained Optimal Sequenced Route* (RCOSR) query, where for each category of POIs, there is a threshold representing the minimum rating score acceptable to the query user, in our initial study [15]. Given a starting node and a destination node, and a set of sequenced POI categories with the corresponding rating score thresholds, the RCOSR query finds the route with the minimum distance that visits only one POI of each category in order and the rating score of each POI satisfies the user specified threshold. Revisit the example of the RCOSR query illustrated in Fig. 1. Given a starting node $v_s = v_1$, and a destination node $v_d = v_{16}$, an RCOSR query $Q(v_1, v_{16}, \langle S, 70 \rangle, \langle R, 90 \rangle, \langle G, 50 \rangle, \langle H, 90 \rangle)$ returns an optimal route in which one of the POI in each category is visited and the rating scores of S, R, G, H are greater than 70, 90, 50, 90, respectively.

To answer the RCOSR query, one may adopt a greedy search to find the nearest POI satisfying the threshold in each step and generate the route result. Consider the example Fig. 1. While this method can quickly find a feasible result $\overline{R}_g = \{v_1, p_2^S, p_2^R, p_2^G, p_2^H, v_{16}\}$ with a total cost of 114 (i.e., the blue route), but it is not optimal. The optimal solution of this RCOSR query is actually

$\overline{R}_o = \{v_1, p_1^S, p_1^R, p_2^G, p_2^H, v_{16}\}$, which costs 85 only (i.e., the red route). It is obvious that the greedy search may not ensure the optimality for this problem. An idea is to filter the POIs that do not satisfy the rating score threshold to avoid unnecessary expanding exploration. However, the latest optimal sequenced route algorithm TD-OSR [6] cannot be directly applied to this problem, which does not consider the constrained rating score and is not efficient for our problem.

To tackle the RCOSR query problem, we first revise the TD-OSR algorithm as our baseline, named as *MTDOSR*. The TD-OSR algorithm is originally designed for OSR queries in time-dependent road networks, but it can be also extended to address the traditional OSR problem. For our problem, we modify TD-OSR to apply on static road networks and solve RCOSR queries. The main idea of MTDOSR is to use the A* search scheme equipped with an admissible heuristic function. To find the next node to expand the current subroutes during the network expansion, MTDOSR checks the POI to see whether it satisfies the query threshold before inserting it into the path. However, such an expansion scheme fails to exploit all the query categories and generates a large number of candidate subroutes, which consumes a lot of memory to store them in the heap. Furthermore, the top entry (i.e., subroute) may not be the globally optimal choice to be used to expand. To overcome these two shortcomings, we try to propose a new RCOSR algorithm, *Optimal Subroute Expansion* (OSE), which iteratively finds the optimal subroute for $Q(v_s, p_j^{c_i}, \langle c_1, \dots, c_{i-1} \rangle)$ and $p_j^{c_i}$ is one specific POI in category c_i until the optimal route ending at destination node is obtained, which is the query result. However, OSE is very time-consuming to compute the distance of many POI pairs for obtaining the cost of each optimal subroute. To enhance the OSE algorithm, we propose an index, called *Reference Node Inverted Index* (RNII), to accelerate the distance computation and quickly retrieve POIs of each category for POI filtering. To determine the appropriate reference nodes for RNII, we develop a *Greedy Merge* (GM) strategy to determine the reference nodes by maximizing the number of POI pairs they can cover. To effectively utilize RNII and OSE, we propose a new efficient RCOSR algorithm, called *Recurrent Optimal Subroute Expansion* (ROSE), which iteratively searches the optimal route using OSE. By continuously updating the lower bound distance between POIs computed by RNII with the exact shortest distance, the guiding path that obtained from OSE gets closer to the optimal solution. ROSE terminates when the cost of guiding path is equal to its shortest length. Finally, we extend our ROSE to tackle the top k RCOSR query. This article extends the initial study, by (1) enriching the related work and proving the optimality of ROSE to answer the RCOSR query; (2) formalizing and tackling top

k RCOSR search and proposing corresponding methods for fast query processing; (3) conducting a more comprehensive performance evaluation which evaluates the top k RCOSR query.

The contributions of this article are summarized as follows:

- This paper presents and tackles the rating constrained optimal sequenced route (RCOSR) query problem in which the POIs in the sequenced route should satisfy category rating thresholds.
- We propose the MTDOSR algorithm as our baseline to answer the RCOSR query and explain its inefficiency.
- We propose a new OSE algorithm, which expands the optimal subroutes in dynamic programming scheme. To accelerate the distance computation of POI pairs in OSE, we propose a new index (RNII).
- Based on the OSE and RNII, we further develop a new algorithm (ROSE), which iteratively computes the optimal rating constrained sequenced route as the guiding path to guide the exploration. In addition, we prove the optimality of ROSE to answer the RCOSR query.
- We extend ROSE to propose KROSE, to tackle top k RCOSR query.
- We conduct extensive experiments on synthetic and real road networks to evaluate the proposed algorithms. The experimental results show that ROSE significantly outperforms the baseline by 92.25% in query time and 79.94% in expanded nodes on average.

2 Related Work

This work is relevant to two lines of research, including *optimal route queries*, and *indexes for the road networks*.

2.1 Optimal Route Queries

Li et al. [13] first propose the trip planning query (TPQ) in spatial databases. After that, a variant of the TPQ query, namely the optimal sequenced route query (OSRQ), is studied by Sharifzadeh et al. [22]. In OSRQ, the POI sequence in the optimal route is specified by the user. Three corresponding algorithms (i.e., LORD, R-LORD, and PNE) are developed for both Euclidean and general graphs. Moreover, Sharifzadeh and Shahabi [21] study the OSR query processing algorithm using Voronoi diagram. Recently, Liu et al. [16] study the top- k optimal sequenced route query, which mainly designs the efficient algorithms for finding the k optimal routes. Yawalkar and Ranu [25] solve the personalized route preference problem with skyline route queries. Chen et al. [4] study the *multi-rule Partial Sequenced Route* (MRPSR) query, which finds the optimal route via a number

of POIs in a partial visiting order defined by the user. Costa et al. [6] study optimal sequenced route queries in time-dependent road networks and propose an effective algorithm called TD-OSR, which is based on the A* scheme with an admissible heuristic function. In this paper, we first modify the TD-OSR to answer our RCOSR query as our baseline.

Moreover, Chen et al. [4] study the *multi-rule Partial Sequenced Route* (MRPSR) query, which finds the optimal route via a number of POIs in a partial visiting order defined by the user. Obviously, the MRPSR query is more general and can be converted to TPQ and OSR queries. In addition, Ohsawa et al. [17] study the OSR query in Euclidean space and develop the EOSR algorithm based on *Incremental Euclidean Restriction* (IER) [2]. Furthermore, some works [1, 5, 9–11, 19] study the optimal sequenced route query among a group of users.

As for weight constraints on POIs, some research attention has been paid to this area. Dai et al. [7] consider not only road length but also other factors such as POI rating and propose the personalized and sequenced route planning (PSR) query. In PSR, a score is computed for each route for a query, which is determined by both route length and POI rating. Sasaki et al. [20] propose the skyline sequenced route query which searches for all preferred sequenced routes to users by extending the shortest route search with the semantic similarity of POIs in the route. Yao et al. [24] study the multi-approximate-keyword routing query, which returns the shortest route that passes through at least one matching POI for each query keyword. Later in [3], the authors consider weighted POIs in optimal sequenced group trip planning query, where the weight of POIs is computed as utility and the cost of a trip consists of a distance value and utility value.

2.2 Indexes for Road Networks

To accelerate distance computation and nearest neighbor searching, a number of researches on road network indexes have been investigated. R-tree and its variants [8] as the most popular spatial indices have been used in recent years. Zhong et al. [27] propose G-tree for shortest distance computation and nearest neighbor query on large road networks, which splits the road network into multiple sub-networks and then constructs a balanced tree. Thus each node in G-tree corresponds to a sub-network. One of its advantages is that the space complexity is relatively low; thus, it can easily scale up to large datasets. Another effective index is Voronoi diagram [21], which partitions the space into regions named cells, but it uses Euclidean distance of two nodes to split and thus works mostly for Euclidean graph.

Kriegel et al. [12] propose *graph network embedding* to speed up the range and k -nearest neighbor queries. This work is the most related one to our work. Although the idea

of graph embedding utilizing reference nodes has been studied to build the filter-refinement architecture, our work is the first attempt to apply the idea of graph embedding in the RCOSR problem. In this paper, we propose a new reference node distribution strategy to guide the determination of the reference nodes.

3 Problem Formalization

In this section, we first introduce some fundamental notations and then formalize RCOSR problem. Generally, a road network is typically represented as a graph.

Definition 1 (Graph) A graph $G(V, E, P)$ consists of a node set V and an undirected weighted edge set $E \subseteq V \times V$. P is a set of POIs located on the edges, where each POI belongs to one category, denoted by c_i . The total number of nodes (not including the POIs) is denoted as $|V|$, and the weight on the edge indicates the length of the edge. In addition, $dist(v_i, v_j)$ denotes the shortest distance between nodes v_i and v_j . We assume that the distance satisfies the triangle inequality, and specially $dist(v_i, v_i) = 0$. Besides, each POI is associated with a rating score, which ranges from 0 to 100.

Definition 2 (Route) We define a route $\vec{R} = \{v_1, v_2, \dots, v_m\}$ where v_i ($1 \leq i \leq m$) is a node/POI in a road network. The cost of a route $\vec{R}$, denoted by $cost(\vec{R})$, is $\sum_{i=1}^m dist(v_{i-1}, v_i)$.

Before defining the RCOSR query, we first define the *feasible rating constrained route* and *optimal rating constrained sequenced route*.

Definition 3 (Feasible rating constrained route) Given a source-destination pair (v_s, v_d) and a category sequence $C = \{c_1, c_2, \dots, c_n\}$ as well as the corresponding rating threshold set $T = \{t_1, t_2, \dots, t_n\}$ (for simplicity, we use $CT = \{\langle c_1, t_1 \rangle, \langle c_2, t_2 \rangle, \dots, \langle c_n, t_n \rangle\}$ and $|CT| = n$ to represent the category-threshold pair sequence), a feasible rating constrained route $\vec{R}$ satisfies the following constraints: (i) $\vec{R}$ starts from starting node and ends at the destination node. (ii) $\vec{R}$ passes through at least one POI for each category in C and follows the sequence order in C . (iii) The rating score of the POIs in $\vec{R}$ should be equal or larger than the corresponding threshold in T .

According to the definition of feasible rating constrained route, we now define the optimal rating constrained sequenced route below.

Definition 4 (Optimal rating constrained sequenced route) Given a source-destination pair (v_s, v_d) , a category-threshold

pair sequence CT , the feasible route $\vec{R}_o$ is the optimal rating constrained sequenced route with threshold constraint if for any feasible rating constrained route $\vec{R}'$, such that $cost(\vec{R}_o) \leq cost(\vec{R}')$.

Based on the above definitions, we now formally define the RCOSR query.

Definition 5 (RCOSR query) Given a graph G , the RCOSR query is defined as $Q = (v_s, v_d, CT)$, where v_s and v_d are the starting node and destination node, and CT is the category-threshold pair sequence. In particular, if the rating score threshold of the category is not specified by the user, it is set as the average rating score of POIs in that category. The query returns the optimal rating constrained sequenced route, as defined in Definition 4.

When the number of categories $|CT|$ is greater than 1, it can be reduced from TSP [13] that the RCOSR problem is also NP-Hard.

4 Baseline for RCOSR

In this section, we present our baseline algorithm, namely *MTDOSR*, which extends the TD-OSR algorithm [6] to address the RCOSR problem. The main idea of MTDOSR is to utilize A* scheme with an admissible heuristic function to guide the network expansion. To find the most potential node to expand, MTDOSR uses a function $f(v) = g(v) + h(v)$, where $g(v)$ is the distance from starting node to the current node v through the corresponding route, and the heuristic function $h(v)$ is computed as $h(v) = \max(dist(v, v_d), dist(v, p_{nn}^c))$, where v_d is the destination node, and p_{nn}^c is the nearest POI of node v in category c . To accelerate calculating the heuristic function, before the query comes, an improved TD-NE-A* [6] algorithm is executed to calculate the distance of each node to its nearest POI in each category, i.e., $dist(v, p_{nn}^c)$. When computing $f(v)$, the POIs are checked whether it satisfies the query threshold.

During the expansion, a min-heap H is used to store the intermediate subroutes (i.e., entries). The form of the entry is $(v, g(v), f(v), visitedPOIset)$, where v is the current node, $visitedPOIset$ is the POI sequence visited in this subroute. In the following, we explain the running process for the running example in Fig. 1 with the query $Q = (v_1, v_{16}, \langle c_s, 70 \rangle, \langle c_R, 90 \rangle, \langle c_G, 50 \rangle, \langle c_H, 90 \rangle)$. MTDOSR begins to examine the starting node v_1 and computes function $f(v_1)$. Since $g(v_1)$ is 0, $f(v_1)$ is 85 (the shortest distance from v_1 to destination node v_{16}), and the POI sequence in this subroute is empty, then the entry $(v_1, 0, 85, \{\})$ is pushed in H . In the second iteration, v_1 is popped from H and its adjacent

nodes v_3, v_{11}, v_2 are found and pushed in H . At the same time, POIs p_1^S (i.e., over the edge v_3v_1) and p_2^S (i.e., over the edge $v_{11}v_1$) are checked and inserted in the corresponding subroute, respectively. The above iterations continue until all the query categories are visited and the destination node v_{16} of the query is found, the algorithm returns the result $R_o = \{p_1^S, p_1^R, p_2^G, p_2^H\}$. During this process, this expansion of subroutes guided by $f(v)$ produces many invalid subroutes, e.g., when expanding on node v_4 , node v_5 is de-heap from H and visited, which is not in the optimal route. The main reason is that the heuristic function of v used in MTDOSR only utilizes either the farthest POI or the destination node, which fails to make full use of all the query category sequences in the query; thus, MTDOSR is not efficient.

5 Recurrent Optimal Subroute Expansion

As analyzed in the above section, if we can fully make use of the information with respect to all the query categories, the route expansion can be more effective. Inspired by this, we try to design a new RCOSR algorithm, *Optimal Subroute Expansion* (OSE), to search for the optimal solution.

5.1 Optimal Subroute Expansion Algorithm

The main idea of optimal subroute expansion is to iteratively find the optimal rating constrained sequenced route (ORCR) for $Q = (v_s, p_j^{c_i}, \langle c_1, \dots, c_{i-1} \rangle)$ where $p_j^{c_i}$ is one POI in category c_i , until the *optimal rating constrained route* ending at destination node is obtained, which is the query result. The above idea is naturally implemented using dynamic programming shown as follows.

Definition 6 (*Dynamic programming formulation*) Given a query $Q = (v_s, v_d, CT)$, we construct a dynamic programming matrix OS with $k + 1$ rows and $\max(|c_i|)$ columns, where $i = (0, 1, \dots, k)$. The value of $OS[i, j]$ represents the cost of ORCR ending at $p_j^{c_i}$. In particular, the $(k + 1)$ -th row represents the ORCR ending at node v_d . The dynamic programming formulation is as follows:

$$OS[i, j] = \begin{cases} 0 & \text{if } i = 0 \\ \min_{1 \leq l \leq |c_{i-1}|} \{OS[i-1, l] + \text{dist}(p_l^{c_{i-1}}, p_j^{c_i})\} & \text{if } i > 0 \end{cases} \quad (1)$$

Consider a query $Q = (v_s, v_d, CT)$. The OSE accepts the query Q and a currently feasible route cost cost_{curr} , which is initialized with the cost of the greedy route (there is a greedy search after the query comes which generates a greedy route) to be used as a pruning threshold. For each query category c_i , we first retrieve the POIs of c_i and check if the

POI satisfies the rating constraint and those unqualified POIs are filtered. Then we construct the optimal subroute of each POI in dynamic programming and check whether the cost of the optimal subroute is greater than the current threshold cost_{curr} . If the cost of the current subroute exceeds cost_{curr} , it is considered invalid and should be pruned.

When constructing the optimal subroute of each POI in dynamic programming, with the optimal subroutes set S_{ps} , for each subroute $R_{os}^{p^{c_{i-1}}}$ ending at $p^{c_{i-1}}$, we extract the POI $p^{c_{i-1}}$ from the route and compute each new subroute cost cost_{curr} by adding up the distance from p^{c_i} to $p^{c_{i-1}}$ and the cost of $R_{os}^{p^{c_{i-1}}}$, and then compare the result with the current minimum cost minCost to find the optimal one.

Notice that in OSE, to calculate the cost of the optimal subroutes, the computation of the exact shortest distance between each two POIs is required; thus, it is too expensive to simply use the OSE algorithm to answer our RCOSR query. Therefore, our idea is to adopt the estimated distance (i.e., a lower bound) for each two POIs instead of the exact shortest distance to accelerate the computation. For efficient distance estimation between each two POIs, we next propose a new *Reference Node Inverted Index* (called RNII).

5.2 Reference Node Inverted Index

In this section, we develop our *Reference Node Inverted Index* (RNII), which is used for distance estimation and POI filtering.

5.2.1 Graph Embedding

As the basis of our RNII, we first recall the idea of graph embedding, which is proposed by Krieger et al. [12]. Graph embedding is used to perform distance approximation in large datasets. The main idea of graph embedding is to find a set of reference nodes on the graph and computes the shortest distances from each object (i.e., nodes, POIs) to these reference nodes on the graph, then transforming these distances into vectors. When estimating the distance between any two objects, a vector operation is executed to obtain the upper or lower bounds of the road network distance. Compared with regular distance approximation methods such as Euclidean distance estimation, utilizing graph embedding can significantly accelerate the computation, and the accuracy is relatively higher [12].

Given a reference nodes set $\mathcal{RN} = (r_1, r_2, \dots, r_t)$, for any POI p_i , the distance vectors are $V_{p_i} = (\text{dist}(p_i, r_1), \dots, \text{dist}(p_i, r_t))^T$.

As shown in Fig. 2, assuming two reference nodes are v_5 and v_6 , the distance vector of POI p_1 is computed as $V_{p_1} = (\text{dist}(p_1, v_5), \text{dist}(p_1, v_6))^T = (3, 6)^T$, and for p_2 it is $V_{p_2} = (\text{dist}(p_2, v_5), \text{dist}(p_2, v_6))^T = (6, 3)^T$. Once the distance

Fig. 2 A road network with two POIs

vectors are built, we then perform distance approximation by vector operation. Accordingly, we find the maximum value among each dimension of the vector from the difference (denoted as $LB_{p_i, p_j} = \max(|3 - 6|, |6 - 3|) = 3$). Formally, we give the equations of LB_{p_i, p_j} .

$$LB_{p_i, p_j} = \max_{l=1, \dots, t} (V_{p_i}[l] - V_{p_j}[l]) \quad (2)$$

where $V_{p_i}[l]$ indicates the l -th element of vector V_{p_i} . According to the triangle inequality, we can infer that $LB_{p_i, p_j} \leq \text{dist}(p_i, p_j)$, which represents the lower bound of the distance approximation.

Selecting appropriate reference nodes is important to the accuracy. For example, in the road network in Fig. 2, the lower bound distance between p_1 and p_2 is to be estimated. It is better to select v_7 as the reference node than v_1 . Such an ideal reference node is desired to obtain the exact or tighter lower bound distance. Furthermore, the accuracy of distance approximation using the reference nodes relies on the number and distribution of the selecting reference nodes. According to these issues above, we discuss the strategy of how to determine the reference nodes below.

5.2.2 Determining the Reference Nodes

The strategy of determining reference nodes significantly influences the accuracy of distance approximation. Notice that in the rating constrained optimal subroute expansion, we just exploit the lower bound to estimate the approximate distance of POIs. Thus, our goal is to derive a tight lower bound of distance approximation instead of the upper bound when designing the strategy of determining reference nodes. A natural and simple idea is to randomly pick up nodes (e.g., $v_1, v_2, \dots, v_n$)² from the road network. Since this strategy is with high uncertainty, it cannot guarantee the efficiency of

RNII. Accordingly, we propose a new *Greedy Merge* (GM) strategy to guide how to determine the reference nodes.

Consider a reference node r and a pair of POIs $\langle p_i, p_j \rangle$ in the road network. If the LB_{p_i, p_j} computed according to r is exactly equal to $\text{dist}(p_i, p_j)$, we say r covers the pair $\langle p_i, p_j \rangle$, the more pairs it covers, the better lower bound estimation it provides. The main idea of the GM strategy is to find a set of nodes as the reference nodes that maximize the number of POI pairs they can cover. To achieve this goal, we first calculate how many pairs of POI that each node covers, and greedily choose the node which can cover the largest number of POI pairs into the reference node set iteratively. This process continues until the number of reference nodes reaches the preset value or the total number of POI pairs covered does not increase anymore. Take the road network in Fig. 1 as an example. Assume the number of reference nodes is 3, we choose nodes $\{v_7, v_{16}, v_0\}$ as the reference nodes using the GM strategy. This reference node set can cover 132 pairs of POIs, while the total number of POI pairs in this road network is 144.

5.2.3 RNII Data Structure

Inverted index is the mapping from keywords to documents and is widely used in the search engine, large-scale database index, document retrieval, multimedia retrieval/information retrieval [23]. In this work, we utilize an inverted index to retrieve the POIs by their categories. Each inverted item in RNII consists of two parts: category ID and corresponding POI entities. All the POI entities of category c_i can be retrieved quickly by querying the records of the inverted index with category c_i . A POI entity p is represented as $\langle PID, \text{score}(p), V_p \rangle$, where PID is the id of p , $\text{score}(p)$ is the rating score of p , and V_p is the corresponding distance vector (i.e., $V_p = \text{dist}(p, r_1), \text{dist}(p, r_2), \dots, \text{dist}(p, r_t)$), where $\text{dist}(p, r_t)$ is the shortest distance from p to the t -th reference node.

5.3 The ROSE Algorithm

Based on the OSE and RNII, we further design a new ROSE algorithm. The ROSE initially adopts the approximate distance (which is computed quickly using RNII) for each two POI and utilizes the OSE algorithm to find the optimal route under the approximate distance. Notice that the route computed by OSE using the approximate distance is not the optimal solution. To find the optimal one, the route is served as the *guiding path* and then we calculate the exact shortest length of this guiding path using the shortest distance algorithm. While calculating the exact shortest length, the exact shortest distances between POIs in the guiding path are obtained and we

² Theoretically, the reference node can be any point in the road network, but for simplicity we select the nodes as the reference nodes.

Fig. 3 An example of illustrating ROSE

replace the approximate distance with the exact shortest distance for these POI pairs. After that, we again employ OSE to find a new guiding path with the updated distance information. By continuing the above steps, the cost of the guiding path gets closer to the optimal solution. The algorithm terminates when the cost of the guiding path is equal to the exact cost of the corresponding expanded route, which is the solution to RCOSR.

Algorithm 1 Recurrent Optimal Subroute Expansion (ROSE)

Require: Query $Q = \{v_s, v_d, CT\}$ ($|CT| = k$)
Ensure: $\vec{R}_{os}$: optimal rating constrained sequenced route for v_d
1: $cost_{guide} \leftarrow 0, R_{guide} \leftarrow \emptyset, cost_{exact} \leftarrow 0$;
2: obtain the exact cost $cost_{exact}$ by invoking the greedy search;
3: Initialize current optimal cost $cost_{tresh}$ with $cost_{exact}$
4: **while** $cost_{guide} \neq cost_{exact}$ **do**
5: $\langle R_{guide}, cost_{guide} \rangle \leftarrow OSE(Q, cost_{tresh})$;
6: $cost_{exact} \leftarrow computeRouteDistance(R_{guide})$;
7: **if** $cost_{exact} < cost_{tresh}$ **then**
8: $cost_{tresh} \leftarrow cost_{exact}$;
9: **end if**
10: **end while**
11: $\vec{R}_{os} \leftarrow \vec{R}_{guide}$;
12: **return** $\vec{R}_{os}$;

The pseudo-code of ROSE is illustrated in Algorithm 1. Given a query Q , we first find a current feasible route (which is updated later) by invoking the greedy search (Line 2). Next, the recurrent process goes as follows: we find the guiding path and obtain its exact shortest length as the threshold cost of the current feasible route (Lines 5–6). Note that when executing OSE, if the shortest distance of some POI pairs has been explored, we use the shortest distance stored in an unordered map *pairDistMap*; otherwise, we use the lower bound distance calculated by RNII. After that, we calculate the exact distance by invoking function *computeRouteDistance()* (Line 6). *computeRouteDistance()*

can be implemented by any shortest distance algorithms in road networks, such as A* with graph embedding as the heuristic function [12] and H2H-Index [18]. At the same time, the exact distance of the involved POI pairs is stored in a map *pairDistMap*, which is used in further distance estimation. We also update the cost of the current solution (Lines 7–8). This process continues until the cost of the expanded route is the same as the cost of its guiding path.

Example 1 Figure 3 illustrates an example of ROSE. In the initial state, the distance between each two POI is initialized with the lower bound distance, which is represented by the dashed line (Fig. 5a). The first iteration computes the optimal route in the current lower bound distance as the guiding path $\{v_s, p_1, p_3, v_d\}$ (i.e., the black bold line in Fig. 5b). Then ROSE calculates as well as records the shortest distance from v_s to p_1 , the shortest distance from p_1 to p_3 , the shortest distance from p_3 to v_d (i.e., the red line in Fig. 5c). In the second iteration, another guiding path (i.e., the black bold line in Fig. 5d) is found and the exact distance of the corresponding POI pairs is calculated (i.e., the red line in Fig. 5e). In the last iteration, the guiding path found by ROSE is $\{v_s, p_1, p_4, v_d\}$, whose cost is equal to the actual length of the corresponding path; thus, ROSE terminates and $\{v_s, p_1, p_4, v_d\}$ is returned as the result.

In addition, the optimality of ROSE is proved as follows.

Lemma 1 *ROSE is optimal for the RCOSR query.*

Proof Consider a feasible route $\{v_s, p_f^{c_1}, \dots, p_f^{c_i}, \dots, p_f^{c_n}, v_d\}$ of category $c_1, c_2, \dots, c_n$ as well as starting node v_s and ending node v_d in graph G . We assume the guiding path found by OSE, denoted as $\vec{R}_{guide} = \{v_s, p_o^{c_1}, \dots, p_o^{c_i}, \dots, p_o^{c_n}, v_d\}$, where $p_o^{c_i}$ is passed by R_{guide} for category c_i and the cost of guiding path is

$cost(\overrightarrow{R_{guide}}) = distEst(v_s, p_o^{c_1}) + distEst(p_o^{c_n}, v_d) + \sum_{i=1}^{k-1} distEst(p_o^{c_i}, p_o^{c_{i+1}})$. According to the definition of guiding path, $\overrightarrow{R_{guide}}$ is the optimal route among all the routes including route $\{v_s, p_f^{c_1}, \dots, p_f^{c_i}, \dots, p_f^{c_n}, v_d\}$, we have $cost(\overrightarrow{R_{guide}}) \leq distEst(v_s, p_{c_1f}) + \sum_{i=1}^{n-1} distEst(p_f^{c_i}, p_f^{c_{i+1}}) + distEst(p_f^{c_n}, v_d)$. Moreover, $distEst(v_i, v_j)$ is the exact shortest distance $dist(v_i, v_j)$ or lower bound distance LB_{v_i, v_j} mentioned above, and we can deduce that $cost(\overrightarrow{R_{guide}}) \leq dist(v_s, p_f^{c_1}) + \sum_{i=1}^{n-1} dist(p_f^{c_i}, p_f^{c_{i+1}}) + dist(p_f^{c_n}, v_d)$. Because the termination condition of ROSE is when the cost of guiding path equals to the exact cost of the expanded route corresponding to the guiding path, we have $cost(\overrightarrow{R_{guide}}) = dist(v_s, p_o^{c_1}) + \dots + dist(p_o^{c_n}, v_d)$. Due to Eq. (2), we deduce that $cost(\overrightarrow{R_{guide}}) = dist(v_s, p_o^{c_1}) + \sum_{i=1}^{n-1} dist(p_o^{c_i}, p_o^{c_{i+1}}) + dist(p_o^{c_n}, v_d) \leq dist(v_s, p_f^{c_1}) + \sum_{i=1}^{n-1} dist(p_f^{c_i}, p_f^{c_{i+1}}) + dist(p_f^{c_n}, v_d)$, which shows that the cost of $\overrightarrow{R_{guide}} = R_{os}$ is the minimum among any other feasible solutions. From the above analysis, the proof completes. $\square$

Compared to MTDOSR, ROSE has the following advantages: (1) ROSE utilizes the information of every query category. The guiding path is an optimal route under lower bound distance of the query, which consists of POI in every query category; thus, it can lead to more accurate guidance of the expansion. (2) ROSE uses an inverted index to manage and pre-filter the unqualified POIs, rather than filtering the POIs during route expansion. (3) ROSE updates and uses the cost threshold for pruning in OSE, while MTDOSR has no pruning strategies.

5.4 Complexity Analysis

In this section, we analyze the time and space complexity of MTDOSR and ROSE. Let $|V|$ be the size of nodes in the graph, $|E|$ be the number of edges, $|C|$ be the total categories of POI, r be the number of reference nodes, and k be the size of query categories. In addition, we assume m is the average number of qualified POIs in each category. Because a Dijkstra shortest distance algorithm is executed before the main loop, the complexity of MTDOSR is $O(k|E|\log k|V| + |V||E|\log|V|)$ [6]. For each iteration in ROSE, we first invoke the OSE to find the guiding path. In OSE, it constructs the optimal subroute for each POI using dynamic programming, whose time complexity is $O(mr)$. Moreover, it requires exploring km POIs. Thus, OSE takes $O(krm^2)$ time for each iteration. After that, *computeRouteDistance* takes $O(k|V||E|\log|V|)$ to compute the exact distance of guiding path (using Dijkstra). The total iterations of ROSE depend on the accuracy of lower bound distance computed by RNII; in the worst case, the

iteration goes m^k times for each query. Note that in reality, due to the lower bound estimation provided by RNII, the iteration goes far less than m^k times. In theory, the overall time complexity of ROSE is $O(km^k(m^2r + |V||E|\log|V|))$. Besides, the space complexity of RNII is $O(|C|mr)$.

6 Top k RCOSR Query

In this section, we extend ROSE to answer the top k RCOSR query, denoted as *RCkOSR*. *RCkOSR* is to return the top k optimal rating constrained sequenced routes. As shown in ROSE, the main idea is to recurrently exploit OSE to find the optimal route. Thus, we first adjust the *Optimal Subroute Expansion* (OSE), to search for the top- k optimal solution.

6.1 Top k Optimal Subroute Expansion Algorithm

The main difference between OSE and top k OSE is to iteratively find the top k *optimal rating constrained sequenced routes* (ORCRs) for $Q = (v_s, p_j^{c_i}, \langle c_1, \dots, c_{i-1} \rangle)$ instead of the top 1 ORCR where $p_j^{c_i}$ is one POI in category c_i , until the top k ORCRs ending at destination node are obtained, which are the query results. Accordingly, the corresponding dynamic programming is shown as follows.

Definition 7 (*Dynamic programming formulation*) Given a query $Q = (v_s, v_d, CT)$, we construct a dynamic programming matrix OS with $j + 1$ rows and $\max(|c_i|)$ columns, where $i = (0, 1, \dots, n)$. The value of $OS[i, j, k]$ represents the cost of the k -th ORCR ending at $p_j^{c_i}$. Specially, the $(k + 1)$ -th row represents the ORCRs ending at node v_d . The dynamic programming formulation is as follows:

$$OS[i, j, k] = \begin{cases} dist(v_s, p_0^{c_i}) & \text{if } i = 0 \\ \min_{1 \leq l \leq |c_{i-1}|, 0 \leq h \leq k} \{OS[i-1, l, h] + dist(p_l^{c_{i-1}}, p_j^{c_i})\} & \text{if } i > 0 \end{cases} \quad (3)$$

Consider a query $Q = (v_s, v_d, CT)$. Different from OSE, top k OSE maintains the maximal cost $cost_k$ among k feasible routes, which initially can be obtained from a k times' greedy search after the query comes which generates k greedy routes to be used as a pruning threshold. For each POI, we construct the top k optimal subroutes in dynamic programming and check whether the cost of the k -th subroute is greater than the current threshold $cost_k$. If the cost of the current subroute exceeds $cost_k$, it is considered invalid and should be pruned.

An example of top k OSE is shown in Fig. 4, which is one part of Fig. 1 (left part of this figure). Consider a query $Q = (v_s, v_d, CT = \{S, G\})$ where $k = 3$. For category S , there are two POIs p_1^S and p_2^S , while for category G , there are three POIs p_1^G , p_2^G and p_3^G . For POI p_1^S of category S , we only have one route from s to p_1^S and the corresponding cost is 30.

Fig. 4 An example of top k OSE **Fig. 5** An example of illustrating the KROSE algorithm

Thus, for the value of $OS[1, 1, 3]$, we store $[30, 30, 30]$. Similarly, for p_2^S of category S , we also have one route from s to p_2^S and $OS[2, 1, 3]$ is $[20, 20, 20]$. The detail of $OS[i, j, n](n = 1, 2, \dots, k)$ is shown in right table.

6.2 The KROSE Algorithm

Based on the top k OSE and RNII, we design a top k ROSE algorithm, called *KROSE*. The ROSE initially adopts the approximate distance (which is computed quickly using RNII) for each two POI and utilizes the top k OSE algorithm to find the top k routes under the approximate distance. Different from ROSE, in top k ROSE, k routes are served as the *guiding paths* and then we calculate the exact shortest length of each guiding path using the shortest distance algorithm. While calculating the exact shortest length for the guiding path with the minimal cost, the exact shortest distances between POIs in the guiding path are obtained and we replace the approximate distance with the exact shortest distance for these POI pairs. After that, we again employ top k OSE to find k new guiding paths with the updated distance information. By continuing the above steps, the cost of the k -th guiding path gets closer to the k -th optimal route. The algorithm terminates when the cost of the k -th guiding path

is equal to the exact cost of the corresponding expanded route, which is the solution to RCOSR.

Example 2 Figure 5 illustrates an example of KROSE where $k = 2$. In the initial state, same to the example of ROSE, the distance between each two POI is initialized with the lower bound distance, which is represented by the dashed line (Fig. 5a). The first iteration finds two guiding path $\{v_s, p_1, p_3, v_d\}$ and $\{v_s, p_1, p_4, v_d\}$ (i.e., the black bold line in Fig. 5b) under current lower bound distance. Then top k ROSE calculates as well as records the shortest distance from v_s to p_1 , the shortest distance from p_1 to p_3 , the shortest distance from p_3 to v_d , and the shortest distance from p_1 to p_4 , the shortest distance from p_4 to v_d , (i.e., the red line in Fig. 5c). Currently, the top 2 routes are $\{v_s, p_1, p_4, v_d\}$ (cost 80) and $\{v_s, p_1, p_3, v_d\}$ (cost 95); thus, the cost threshold $cost_{thresh}$ is 95. In the second iteration, we also find two new guiding paths $\{v_s, p_2, p_4, v_d\}$ with approximate cost 85 and $\{v_s, p_2, p_5, v_d\}$ with approximate cost 87. Similarly, we compute the exact cost of these two guiding paths. We update the top 2-rd route as $\{v_s, p_2, p_4, v_d\}$ with cost 93, while the cost of $\{v_s, p_2, p_5, v_d\}$ is 105. Since the remaining paths are filtered by the current threshold 93, the query processing ends. Finally, $\{v_s, p_1, p_4, v_d\}$ and $\{v_s, p_2, p_4, v_d\}$ are returned as the result.

Table 1 Parameter ranges and default values

Parameters	Query size	Network size (×1000)	POI numbers	Average degree of nodes	Number of reference nodes	k values
Ranges	1, 3, 5, 7, 9	10, 40, 70, 100	100, 500, 1000	1.5, 2.5 , 3.5	20, 40, 80 , 160, 320	1, 2, 3, 4, 5

7 Experiments

In this section, we conduct a number of experiments to evaluate the performance of the proposed algorithms and RNII using both real datasets and synthetic datasets. All the algorithms are implemented using C++, and the experiments are conducted on a server with an Intel Core i7-9700 CPU of 3.0 GHz and 16 GB RAM. We use the following real datasets:³

- *CA* The *CA* dataset is the real road network of California (Footnote 3). It contains 21,048 nodes, 21,693 road edges, and 87,635 POIs belong to 64 categories.
- *OL* The *OL* dataset is the real road network of Oldenburg city (Footnote 3). It contains 6105 nodes, 7035 road edges, and 2,404 POIs belong to 26 categories.

In addition, we generate a synthetic dataset that contains 100,000 nodes, 125,000 road edges, and 10,000 POIs belong to 50 categories and we randomly set the rating score ranging from 0 to 100 for each POI. We generate the synthetic dataset as follows. We first generate an m cross tree. Then we randomly choose to connect some nodes by the edges. While for generating the POIs, we randomly decide whether there is POIs on each edge and how many edges we have on one edge. The maximal number of POIs on one edge is 10.

We conduct the performance evaluation in two aspects: (1) evaluating the efficiency of RCOSR algorithms. We compare the *query time* and the *number of visited nodes* (i.e., the number of times the nodes gets visited) under various parameters, including the *query size* (i.e., the number of query categories), the *network size* (i.e., $|V|$), the *POI numbers*, shown in Table 1; and (2) evaluating the efficiency of RNII. We compare the query time and cover rate using RNII concerning the reference nodes strategies. The bold number represents the default values. In each experimentation, we vary one parameter at a time and fix the other parameters as the default value, and generate 100 random queries. The reported experimental results are obtained by averaging the query time as well as the number of visited nodes.

7.1 Efficiency of RCOSR Algorithms

In this section, we evaluate the efficiency of MTDOSR and ROSE. While implementing the *computeRouteDistance()* function in ROSE, we use A* algorithm to calculate the shortest distance between POIs and use the lower bound distance as the heuristic function. Moreover, we construct the RNII by adopting the GM strategy with the number of reference nodes as 80 in ROSE.

Algorithm performance in real road network The query size refers to the number of POI categories in the query (i.e., the $|CT|$). In this experiment, we compare the efficiency of ROSE using the random strategy (noted as ROSE_RD), ROSE using greedy merge strategy (noted as ROSE_GM), and MTDOSR using the two real datasets. From the results shown in Fig. 6, we can see that the query time and the number of visited nodes for all algorithms increase when the query size increases. ROSE_GM outperforms MTDOSR with regard to the query time (with 85.07% in CA and 98.88% in OL improvement for average query size, respectively) and the number of visited nodes (with 66.8% in CA and 93.08% in OL improvement for average query size, respectively), and ROSE_GM outperforms ROSE_RD with regard to the query time (with 42.96% in CA and 93.8% in OL improvement for average query size, respectively) and the number of visited nodes (with 37.3% in CA and 46.2% in OL improvement for average query size, respectively). For the OL dataset, the superiority of ROSE is more obvious compared to that in the CA dataset. It can be explained that the number of POIs in the OL dataset is smaller than that of CA. That is because ROSE computes the guiding path using all the POI categories information, the ROSE runs faster in a sparser road network. Figure 6c, f shows the number of ROSE iterations before the optimal route is found. For the CA dataset, the average numbers of iterations using random and greedy merge strategy are 60.89 and 34.11, respectively, while in the OL dataset the ROSE only needs 6.84 and 5.16 iterations on average to find the optimal route with respect to random and greedy merge strategies.

Effect of the network size Figure 7 shows the query time and the number of visited nodes of ROSE and MTDOSR with respect to different network sizes in the synthetic dataset. As illustrated in Fig. 7a, the query time of the two algorithms increases by increasing the network sizes, while ROSE outperforms the MTDOSR significantly. Figure 7b also describes the results of the number of visited nodes by varying the network sizes. When the network size increases,

³ <http://www.cs.utah.edu/~lifeifei/SpatialDataset.htm>.

Fig. 6 Performance w.r.t query size in real road network

Fig. 7 Time w.r.t. network size

Fig. 9 Time w.r.t. average degree of nodes

Fig. 8 Time w.r.t. POI numbers

finding the optimal rating constrained sequenced route requires to visit more nodes.

Effect of POI Numbers As expected, the query time and the number of visited nodes of ROSE increase when increasing the POI percentages in Fig. 8. When the POI number increases, there are more guiding paths that are close to the optimal rating constrained sequenced route; thus, ROSE requires to spend more iterations to find the optimal one incurring more computations of the exact distance between

the POIs when constructing optimal subroutes. ROSE outperforms MTDOSR with respect to the effect of the POI number, especially when the POI number is small. As the number of POI numbers grows, the query time of both algorithms increases, and ROSE is more sensitive to the number of POI numbers.

Effect of the average degree of nodes Figure 9 compares the algorithm performance with respect to the average degree of nodes. For the two algorithms, the query time and the number of visited nodes increase slightly when increasing the average degree of nodes, because the network becomes more complex and there is more possibility to search the routes. In addition, we can observe from Fig. 8 that MTDOSR takes more time to visit the same amount of nodes compared to ROSE. This can be explained by the computation cost of the heap that MTDOSR maintains. The heap used in MTDOSR stores the nodes of the entire path, which makes the number of nodes increase as the iteration continues. Moreover, it is very time-consuming to arrange and search the heap,

Fig. 10 Query time w.r.t k value Fig. 11 Visited nodes w.r.t k value

while ROSE uses A* algorithm to compute the distance of two POIs, which only needs to maintain a small number of nodes in the heap.

7.2 Efficiency of RCKOSR Algorithms

Effect of k values In addition, we compare the performance of KROSE and KMTDOSR when increasing k values, as shown in Figs. 10 and 11. Notice that KMTDOSR is the variant of MTDOSR, which is used to search the top k optimal routes. The main idea of KMTDOSR is to expand the routes until k routes are found. The results on query time by varying k values can be seen from Fig. 10. It shows that KROSE outperforms KMTDOSR a lot when increasing k values. Moreover, the query time of KROSE keeps more stable than the query time of KMTDOSR. We also test algorithms on the number of visited nodes by KMTDOSR in Fig. 11. In consistent with the results on query time, KROSE visits less nodes than KMTDOSR to find the top k optimal routes.

Effect of network size and POI Numbers Figure 12 shows the results of RCKOSR algorithms (setting $k = 3$) by varying the network size and POI numbers. Regarding the results by varying the networksize (see Fig. 12a), KROSE outperforms KMTDOSR a lot. Similar to the results by varying the networksize, KROSE is also faster than KMTDOSR when varying the POI numbers.

Fig. 12 Time ($k = 3$) w.r.t. network size and POI numbers

Fig. 13 RNII performance w.r.t. number of reference nodes

7.3 Efficiency of RNII Index

As discussed in Sect. 5.2, determining the reference nodes is an essential issue in the construction of RNII. To achieve the desired RNII, we propose the GM strategy. In this section, we evaluate the performance of the GM strategy, in comparison with the Random strategy (i.e., this strategy randomly chooses the reference nodes) under a various number of reference nodes. We use the synthetic network of 10,000 nodes and 1000 POIs, which belong to 10 categories.

Effect of strategies and number of reference nodes It is shown in Fig. 13 that GM outperforms Random significantly under a various number of reference nodes. This also indicates that the reference nodes decided by the GM strategy ensure higher accuracy in estimating the lower bound distance than the reference nodes decided by Random. From Fig. 8, we can observe that the more reference nodes we decide, the more POI pairs we can cover, which illustrates that the more reference nodes we use, the more accuracy we gain.

8 Conclusion

In this paper, we formalize and study the *Rating Constrained Optimal Sequenced Route* (RCOSR) query, which constrains the rating score of all POIs in the result and the optimal route should satisfy the user thresholds. To answer the query, we adapt the TD-OSR algorithm as *MTDOSR* to serve as a

baseline. Next, we try to propose a new *Optimal Subroute Expansion* (OSE) algorithm to solve the problem. Moreover, we propose a *Reference Node Inverted Index* (RNII) to accelerate the distance computation in OSE. Based on OSE and RNII, we propose a new *Recurrent Optimal Subroute Expansion* (ROSE) algorithm. Moreover, we formalize and study a variation of RCOSR query, namely RCkOSR query. At last, a comprehensive performance evaluation is conducted to validate the proposed ideas and demonstrate the efficiency and effectiveness of the proposed index and algorithms.

Acknowledgments This work is supported by the National Natural Science Foundation of China (U1711262, U1711261, U1811264, U1811261, U1911203, U2001211), Guangdong Basic and Applied Basic Research Foundation (2019B1515130001), Key R&D Program of Guangdong Province (2018B010107005).

Funding National Natural Science Foundation of China (61902438, 61902439, U1811264, U19112031), Natural Science Foundation of Guangdong Province under Grant (2019A1515011704, 2019A1515011159), Guangdong Basic and Applied Basic Research Foundation (2019B1515130001), and Hong Kong RGC Grant 12200817.

Availability of data and materials Yes, available.

Declarations

Conflict of interest The authors declare that there is no conflict of interest.

Consent for publication Yes.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Ahmadi E, Nascimento MA (2015) A mixed breadth-depth first search strategy for sequenced group trip planning queries. In: MDM, pp 24–33
- Al-Khalidi H, Abbas Z, Safar M (2013) Approximate range query processing in spatial network databases. Multimed Syst 19(2):151–161
- Barua S, Jahan R, Ahmed T (2017) Weighted optimal sequenced group trip planning queries. In: MDM, pp 222–227
- Chen H, Ku W, Sun M, Zimmermann R (2008) The multi-rule partial sequenced route query. In: SIGSPATIAL, pp 1–10
- Chen B, Zhu H, Liu W, Yin J, Xu J (2021) Querying optimal routes for group meetup. Data Sci Eng (11)
- Costa CF, Nascimento MA, Macêdo JA, Theodoridis Y, Pelekis N, Machado J (2015) Optimal time-dependent sequenced route queries in road networks. In: SIGSPATIAL, pp 1–4
- Dai J, Liu C, Xu J, Ding Z (2016) On personalized and sequenced route planning. WWWJ 19(4):679–705
- Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: SIGMOD, pp 47–57
- Hashem T, Hashem T, Ali ME, Kulik L (2013) Group trip planning queries in spatial databases. In: SSTD, pp 259–276
- Hashem T, Barua S, Ali ME, Kulik L, Tanin E (2015) Efficient computation of trips with friends and families. In: CIKM, pp 931–940
- Jahan R, Hashem T, Barua S (2017) Group trip scheduling (GTS) queries in spatial databases. In: EDBT, pp 390–401
- Kriegel H, Kröger P, Kunath P, Renz M, Schmidt T (2007) Proximity queries in large traffic networks. In: GIS, pp 1–8
- Li F, Cheng D, Hadjieleftheriou M, Kollios G, Teng S (2005) On trip planning queries in spatial databases. In: SSTD, pp 273–290
- Li J, Yang YD, Mamoulis N (2013) Optimal route queries with arbitrary order constraints. TKDE 25(5):1097–1110
- Li W, Zhu H, Liu W, Yin J, Xu J (2021) Optimal Sequenced Route Query with POI Preferences. In: Database systems for advanced applications, 26th international conference, DASFAA 2021, Taipei, Taiwan, April 11–14, 2021, Proceedings, Part I
- Liu H, Jin C, Yang B, Zhou A (2018) Finding top-*k* optimal sequenced routes. In: 2018 IEEE 34th international conference on data engineering (ICDE)
- Ohsawa Y, Htoo H, Sonehara N, Sakauchi M (2012) Sequenced route query in road network distance based on incremental euclidean restriction. In: DEXA, pp 484–491
- Ouyang D, Qin L, Chang L, Lin X, Zhang Y, Zhu Q (2018) When hierarchy meets 2-hop-labeling: efficient shortest distance queries on road networks. In: ICMD, pp 709–724
- Samrose S, Hashem T, Barua S, Ali ME, Uddin MH, Mahmud MI (2015) Efficient computation of group optimal sequenced routes in road networks. In: MDM, pp 122–127
- Sasaki Y, Ishikawa Y, Fujiwara Y, Onizuka M (2018) Sequenced route query with semantic hierarchy. EDBT 2018:37–48
- Sharifzadeh M, Shahabi C (2008) Processing optimal sequenced route queries using voronoi diagrams. GeoInformatica 12(4):411–433
- Sharifzadeh M, Kolahdouzan MR, Shahabi C (2008) The optimal sequenced route query. VLDB J 17(4):765–787
- Singhal A (2001) Modern information retrieval: a brief overview. IEEE DEB 24(4):35–43
- Yao B, Tang M, Li F (2011) Multi-approximate-keyword routing in gis data. In: SIGSPATIAL, pp 201–210
- Yawalkar P, Ranu S (2019) Route recommendations on road networks for arbitrary user preference functions. 04:602–613
- Zheng B, Su H, Hua W, Zheng K, Zhou X, Li G (2017) Efficient clue-based route search on road networks. TKDE 12(4):1846–1859
- Zhong R, Li G, Tan K, Zhou L (2013) G-tree: an efficient index for KNN search on road networks. In: CIKM, pp 39–48