Hindawi
Complexity
Volume 2019, Article ID 4047894, 15 pages
https://doi.org/10.1155/2019/4047894

Research Article

Semantic-Aware Top-k Multirequest Optimal Route

Shuang Wang, Yingchun Xu, Yinzhe Wang, Hezhi Liu, Qiaoqiao Zhang, Tiemin Ma, Shengnan Liu, Siyuan Zhang, and Anliang Li

Software College, Northeastern University, Shenyang 110004, China

Correspondence should be addressed to Shuang Wang; wangsh@mail.neu.edu.cn

Received 25 January 2019; Accepted 3 April 2019; Published 15 May 2019

Guest Editor: Jianxin Li

In recent years, research on location-based services has received a lot of interest, in both industry and academic aspects, due to a wide range of potential applications. Among them, one of the active topic areas is the route planning on a point-of-interest (POI) network. We study the top-k optimal routes querying on large, general graphs where the edge weights may not satisfy the triangle inequality. The query strives to find the top-k optimal routes from a given source, which must visit a number of vertices with all the services that the user needs. Existing POI query methods mainly focus on the textual similarities and ignore the semantic understanding of keywords in spatial objects and queries. To address this problem, this paper studies the semantic similarity of POI keyword searching in the route. Another problem is that most of the previous studies consider that a POI belongs to a category, and they do not consider that a POI may provide various kinds of services even in the same category. So, we propose a novel top-k optimal route planning algorithm based on semantic perception (KOR-SP). In KOR-SP, we define a dominance relationship between two partially explored routes which leads to a smaller searching space and consider the semantic similarity of keywords and the number of single POI's services. We use an efficient label indexing technique for the shortest path queries to further improve efficiency. Finally, we perform an extensive experimental evaluation on multiple real-world graphs to demonstrate that the proposed methods deliver excellent performance.

1. Introduction

In recent years, the rapid advancements of wireless communication techniques, Global Positioning System (GPS), and smart mobile devices have enabled a lot of Location-based Services (LBS). Among them, one of the popular issues is the path/route planning in a point-of-interest (POI) network [1, 2]. The users of the LBS often want to find short routes that pass through multiple POIs; consequently, developing trip planning queries that can find the shortest routes that passed through user-specified categories has attracted considerable attention [3, 4]. While the problem of computing the optimal route has been extensively studied and many efficient techniques have been developed over the past several decades, most of the past studies on route planning focused on origin-destination route planning and did not consider the user's specific requirements.

Recently, there are some approaches that find the route by using the user's queries. However, the approaches may find a longer route than the one that meets the user's actual demands, because the query keywords only show the meaning of user's query rather than requiring the conformance in shape. A major problem with the existing approaches is that they only output routes that perfectly match the given categories [5–8]. Take Figure 1 as an example: each object can be viewed as a POI that has a spatial location and additional keywords. Considering a user who wants to watch a film and she issues a keyword query q with her current location and keyword *film*, if we apply the traditional spatial keyword query method, o_1 is returned as it contains the query keyword. However, we can find that o_6 also meets the user's requirement as we know that the user only wants to watch a film. To overcome this problem, we introduce flexible semantic matching based on POI keywords to find shorter routes in a flexible manner. In addition, each POI contains a lot of keywords and provides multiple services. A POI may cover multiple keywords in query; otherwise, each POI only corresponds to one of the keywords in query. For example, a POI, called *WanDa Plaza*, has a lot of keywords as *cinema, popcorn, food*, etc. If a user is looking for POIs where she

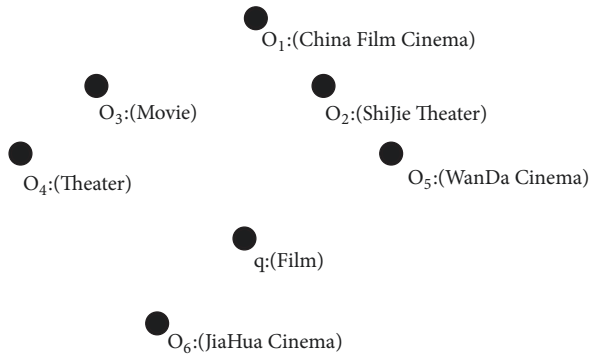


Figure 1: An example of keyword query.

can eat something and see a movie, she issues a query q with keywords *movie* and *food*. We can know that this POI may meet the all requirements of the user; otherwise, she must visit a restaurant and a cinema, respectively. So, we can cut the length of route down by reducing the number of POIs in the route.

Existing approaches find the shortest route that is an optimal sequenced route, but these approaches result in a lack of flexibility in route planning and leave user without possibility of choice. We are proposing to the top-k algorithm in order to provide more choices and satisfy users to the maximum. Besides, compared with keyword ordered query, keyword unordered query is more flexible, and we only need to consider the distance between POI and the current point under the premise of meeting user requirements. At the same time, unordered query can avoid the distant POI becoming the nearest neighbor because the query order of the keywords is no longer considered during the query process.

In this paper, solving the top-k route search problem faces three challenges. The first challenge is the larger search space of the query. Because we consider the semantic relation of the queries and POIs, while not the string matching, then the number of candidate objects is larger than the existing approaches. It calls for effective methods to filter some candidates for avoiding exhaustive search. The second challenge is the strategies to extend the route. Many existing methods only consider the nearest neighbor of the current point, but the route generated by extending from their neighbor perhaps does not become the final optimal result. In this paper, we not only consider the distance between the current object and the neighbors, but also need to take the neighbor as the current object and consider its neighbor for which the distance is the smallest. It shows that the extending route by this method has higher probability to be the final optimal route. Here, we consider the semantic distance and spatial distance simultaneously. In order to efficiently compute the distance cost, we propose a method to use the 2-hop labeling technique [9–12]. The third challenge is route refinement mechanism. The POIs in the final route found by our method may be redundant; that is to say, perhaps more than two POIs provide the same services in one route, since our algorithm is greedy approach. So, we need to propose a refinement mechanism to further enhance the route quality.

The main contributions of this paper can be summarized as follows:

(1) We introduce a semantic similarity to the route search query, which allows us to search for routes flexibly

(2) We propose the *top-k optimal route based on semantic perception* (KOR-SP), which finds all preferred routes related to keyword with semantic perception

(3) We propose a method to find the x-th nearest neighbor based on semantic perception

(4) We use real-world POI datasets to test and prove the superiority of the algorithm.

The remainder of this paper is organized as follows. In Section 2, we briefly review the related work. In Section 3, we formally state the problem. In Section 4, we first introduce the KOR-SP algorithm and how to find the x-th nearest neighbor. The empirical performance study is presented in Section 5. Conclusion and future work are presented in Section 6.

2. Related Work

We review the related works in this section. Route planning is one of the hot topics on LBSs [13, 14]. The algorithms on *destination-oriented route planning* have been split into *single-destination route planning* and *multidestination route planning*. Among them, a number of algorithms belonging to *single-destination route planning*, such as *Dijkstra* [15] and A^* [16], have been proposed to find the shortest route between two locations. Besides, an increasing number of approaches on *multidestination route planning* have been proposed [17–19], such as *Traveling Salesman Problem* (TSP) [19] and *TSP with Neighborhood* (TSPN) [17]. All of the above are *destination-oriented route planning,* but *requirement-oriented route planning* is another kind of routing problem. Li [18] et al. proposed *Trip Planning Query* (TPQ) and proposed *Nearest Neighbor Algorithm* (A_{NN}) and *Minimum Distance Algorithm* (A_{MD}). A_{NN} visits the nearest POI that belongs to the last visited POI and A_{MD} finds each "good" POI that belongs to each unvisited category and traverses these POIs in a nearest neighbor order. However, the planned results of these algorithms are not good since the routes found by these algorithms may be tortuous which means that they are full of twists, turns, or bends. Based on TPQ, Ahmadi and Nascimento [20] studied *Sequenced Group Trip Planning Queries* (SGTPQs) to find a sequence of POIs belonging to the specified categories and minimize the total distance travelled by all groups of users. Sharifzadeh [7] et al. proposed a related query problem named *Optimal Sequenced Route* (OSR) to retrieve the shortest route from a given source via several locations with different categories in a particular order. Based on OSR, Liu [21] et al. proposed top-k optimal sequenced route (KOSR) to find the top-k optimal routes from a given source to a given destination, which must visit a number of POIs with specific categories in a particular order. However, the above works considered that a POI only belongs to a category. In an urban area, a POI may not only belong to a category but also provide various services. A better routing approach should consider whether the provided services on the route satisfy user's requests rather than the categories.

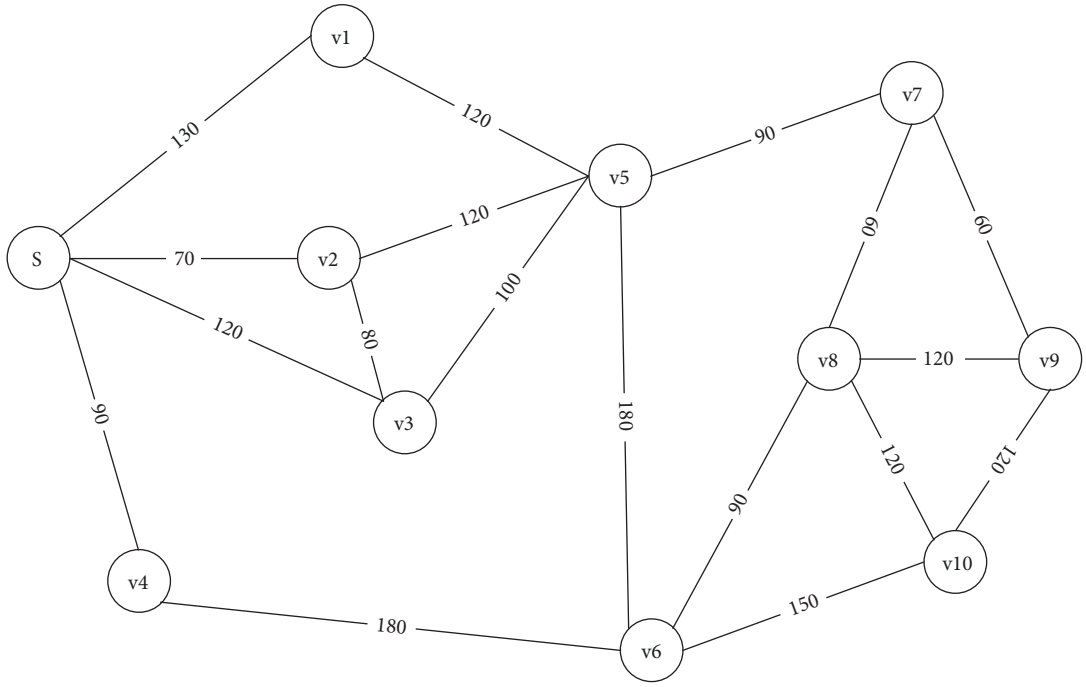


Figure 2: A route network.

In addition, there is also a lot of research on POI. POI recommendation is one of hot topics and it can provide better POIs for route planning. Lei Tang [22] et al. proposed a personal POI recommendation method based on destination prediction. And Jianxin Li [23] proposed *Personalized Influential Topic Search,* or more succinctly PIT-Search. The goal of PIT-Search is to find how important topics and influential users might be better leverage to meet a specific user's information need.

Route planning has attracted a lot of attention. So far, people still focus on user preferences or POI's categories to extend their work. But in reality, the POI's categories are not able to sufficiently represent services provided by POI. So, in order to better meet the needs of the user, it is necessary to consider the services provided by POI when designing the algorithm. So, in this paper, we designed a multirequest route planning algorithm considering the POI's service.

3. Problem Statement

We formalize the KOR-SP problem in this section. Frequent notations are summarized in Table 2.

3.1. Some Definitions. In this section, we first define some terms used in this paper and then specify our research problem.

Definition 1 (graph). An undirected weighted graph $G(V, E, W)$ consists of a set of edge weights that represent the distance between two POIs including a vertex set V and an edge set $E \subseteq V \times V$. Weight function $W : E \longrightarrow R^+$ takes an edge (u, v) as input and returns a nonnegative cost of the edge $W((u, v))$. For example, in Figure 2, we have $W((s, v_4)) = 90$. Note that the edge weights can be arbitrary and may not satisfy the triangle inequality. At the same time, it is also applicable to directed weighted graph and the edge weights can represent distance, time and so on.

Definition 2 (request). Request means a thing, a need, a requirement, or a service that the user wants. $Q = \{q_1, q_2, \ldots, q_{|R|}\}$ denotes the collection of requests.

Definition 3 (POI). POI, the abbreviation of point of interest, represents the specific location in the map. It includes two aspects information: spatial information and key words. It is defined as follows:

$$v = (v.\lambda, v.\kappa), \tag{1}$$

where v is the POI, $v.\lambda$ is the POI's spatial information, usually in the form of latitude and longitude of POI, and $v.\kappa$ is the set of keywords.

It is worth mentioning that each POI may contain multiple keywords, and through these keywords it can get the basic information of the POI. For example, there is a POI named *WanDa* that contains keywords as *movie, food,* and so on. These keywords can describe the basic characteristics of the POI. And these keywords of POI can be defined as follows:

$$v.\kappa = \left\{\kappa_1, \kappa_2, \ldots, \kappa_i, \ldots, \kappa_{|v.\kappa|}\right\}, \tag{2}$$

where k_i is the i-th keyword of POI and $k_{|v|}$ is the total number of POI's keywords.

In addition, we need to consider the semantics of the keyword when querying the POI. For each keyword, we acquire its topics through the *Latent Dirichlet Allocation* (LDA) [24] and set up a collection to hold these topics and every topic's probability. This can be defined as follows:

$$\kappa = \{T, \eta\}, \tag{3}$$

where T and η are the set of topics and topic's probability, respectively.

It is worth mentioning that we use the probabilistic topic model to transform the textual description into their semantic representations, and then we can use them to quantify the semantic correlation between textual descriptions. By applying a popular probabilistic topic model called LDA, we can obtain a topic distribution of each object to describe the semantic correlation between the object keywords and a limited set of potential topics. Given a query and an object, it is possible to measure their semantic similarity based on their topic distributions.

Definition 4 (keyword similarity). Given a query keyword k and a POI's keyword q, the similarity $sim(k, q) \in [0, 1]$ is calculated by an arbitrary function such as the *Wu and Palmer similarity* or length [7, 15] . We assume the relations in the similarity as follows:

$$sim(k, q) = \begin{cases} \beta & , \beta > 0.5 \\ 0 & , else, \end{cases} \tag{4}$$

where k is the query keyword and q is the POI's keyword. If k is relevant to q and corresponding probability $\beta > 0.5$, we set $sim(k, q) = \beta$; otherwise, $sim(k, q) = 0$. And when we calculate the value β as in the following, k and q are topic probability distribution vectors representing the query keyword and the POI's keyword, respectively:

$$\beta = \cos(k, q) = \frac{k \bullet q}{\|k\| * \|q\|} = \frac{\sum_{i=1}^{n} k_i \times q_i}{\sqrt{\sum_{i=1}^{n} k_i^2} \times \sqrt{\sum_{i=1}^{n} q_i^2}}. \tag{5}$$

For example, each tuple in Table 3 is a topic distribution over five topics. Considering a user who wants to watch a movie and she issues a keyword query q with her current location and keyword *cinema*, we can get that $\beta_{cinema,theater} = \cos(cinema, theater) = 0.995$ and the value is larger than 0.5, so the semantic similarity is $sim(cinema, theater) = 0.995$.

Definition 5 (PRQ and QRP). $PRQ(q) = \{v \mid v \in V\}$ provides the set of POIs that is the services with which it can meet user's querying keyword q and $QRP(v) = \{k \mid k \in v.\kappa\}$ provides the set of point's services. Given a $q \in Q$ and a $v \in V$, we can get that $PRQ(q) \subseteq V$ and $QRP(v) \subseteq Q$. Take Table 1 as an example: $PRQ(q_1) = \{v_1, v_3, v_4\}$ and $QRP(v_1) = \{q_1, q_2\}$.

Definition 6 (route). Route refers to a collection containing several POIs. POI in the collection has a certain order to form a route, so the route is defined as follows:

$$R = \{v_1, v_2, \ldots, v_j, \ldots, v_{|R|}\}, \tag{6}$$

Table 1: POI information.

POI	Provided Services
v_1	q_1, q_2
v_2	q_3, q_9
v_3	q_1, q_4, q_5
v_4	q_1, q_2
v_5	q_6
v_6	q_6, q_7, q_9
v_7	q_3, q_9
v_8	q_2, q_7
v_9	q_5, q_7, q_8
v_{10}	q_3

where v_i is the i-th POI in the route and |R| is the number of POIs in the route.

And each route also has its keywords, because the route contains several POIs, so the route also contains the keywords of all POIs, which can be expressed as follows:

$$R.K = \bigcup_{v \in R} v.\kappa. \tag{7}$$

As you can see, the keyword of the route is a collection of all the POI's keywords in the route.

Definition 7 (route cost). Given a set of user's requests $Q = \{q_1, q_2, \ldots, q_n\}$ and a route $R = \{s, v_1, \ldots, v_m\}$, the cost of a route is the spatial distance through all the POIs in the entire route from a given source s. We can denote the cost of route R as follows:

$$\cos t(R) = dist(s, v_1) + \sum_{i=1}^{|R|-2} dist(v_i, v_{i+1}), \tag{8}$$

where $cost(R)$ is the spatial distance of the route, $dist(s, v_1)$ is the distance between the starting point s and the first POI in route, and $dist(v_i, v_{i+1})$ is the distance of the i-th POI to the $(i + 1)$-th POI in the route.

Definition 8 (route average cost). Because one POI may include multiple services and requests, we should consider the number of services in route when extending a partial route. So, the best method is that, calculating the average cost that is the route cost divided by the number of services, we consider the partial route with the least average cost to extend. Given a route $R = \langle s, v_1, v_2, \ldots, v_i, \ldots, v_{|R|} \rangle$ and a set of user's requests $Q = \{q_1, q_2, \ldots, q_n\}$, we denote the average cost as follows:

$$ave(R) = \frac{\cos t(R)}{\left| \bigcup_{i=1}^{|R|} QRP(v_i) \cap Q \right|}. \tag{9}$$

3.2. Two-Hop Labeling Technique. The POI map data is stored on disk. To answer user queries rapidly with low I/O access and speed-up distant cost computation, we build index HI stored on disk.

Table 2: Meaning of notations.

Notation	Meaning	Notation	Meaning
$R_{s,t}$	A route from s to t	K	Top k results are needed
Q	The requests of user	$D_{s,t}$	The distance from s to t
C	The collection of POIs	$v.\lambda$	POI's spatial information
$\|NR\|$	The number of services that meet user's request in route or witness R	$v.\kappa$	POI's keywords
$\|NQ\|$	The number of user's requests	v	POI point of interest

Table 3: Topic distributions of textual descriptions.

Keyword	Topics				
	Exercise	Movie	Drink	Shop	Food
market	0.09	0.09	0.09	0.64	0.09
fast food	0.04	0.04	0.16	0.04	0.72
cinema	0.07	0.72	0.07	0.07	0.07
Wal-Mart	0.07	0.07	0.07	0.72	0.07
theater	0.04	0.84	0.03	0.04	0.04
KFC	0.03	0.03	0.03	0.03	0.88

Table 4: 2-hop index HI.

Vertex	HI(v)
s	$(s,0),(v_2,70),(v_4,90),(v_3,120),(v_1,130)$
v_1	$(v_1,0),(v_5,120),(s,130)$
v_2	$(v_2,0),(v_3,80),(v_5,120)$
v_3	$(v_3,0),(v_5,100)$
v_4	$(v_4,0),(v_6,180)$
v_5	$(v_5,0),(v_7,90),(v_6,180)$
v_6	$(v_6,0),(v_{10},150)$
v_7	$(v_7,0),(v_8,60),(v_9,60)$
v_8	$(v_8,0),(v_9,120),(v_{10},120)$
v_9	$(v_9,0),(v_{10},120)$
v_{10}	$(v_{10},0)$

Table 4 shows the HI for the POI map in Figure 2; for each vertex $v \in V$, 2-hop labeling maintains a label HI(v). In particular, HI(v) consists of a set of label entries in the form of $(u, d_{u,v})$, where $u \in V$ is a vertex that is able to reach v and $d_{u,v} = dist(u, v)$.

We note that it is NP-hard to construct 2-hop tags at a minimum size while satisfying the coverage feature. Therefore, the existing methods [9–11, 25] are all heuristic and approximate the minimal 2-hop labeling index. Alternatively, we can use the full pair shortest path algorithm to generate the index. Although it works, it requires an index size of $O(|V|^2)$, which is unacceptable for large graphs.

4. Proposed Solutions

In this section, we provide the effective method of solving KOR-SP. We described a route planning method based on semantic perception that satisfies multiple requests of user. The method proposes a dominating relationship on candidate routes to filter the candidate routes and thus reduces the search space. In addition, by combining an optimization technique, we can effectively find the x-th nearest neighbor of the current vertex.

4.1. KOR-SP Algorithm. We introduce the domination relationship: the so-called domination relationship is in the same starting point and end point, and the route with larger average cost dominates the small one, as shown in Figure 2, when the user demands service for $\{q_1, q_3, q_6, q_7, q_8\}$ when considering $\langle s \longrightarrow v_1 \longrightarrow v_5\rangle$ (2,250) and $\langle s \longrightarrow v_3 \longrightarrow v_5\rangle$2,220). In the case of the same destination, the numbers of services of route $\langle s \longrightarrow v_1 \longrightarrow v_5\rangle$ and $\langle s \longrightarrow v_3 \longrightarrow v_5\rangle$ provided are equal, but route $\langle s \longrightarrow v_3 \longrightarrow v_5\rangle$ has smaller distance cost, so the $\langle s \longrightarrow v_3 \longrightarrow v_5\rangle$ belongs to the dominating path, and $\langle s \longrightarrow v_1 \longrightarrow v_5\rangle$ belongs to the dominated path.

Definition 9 (domination). Given a user's request $Q = \{q_1, \ldots, q_j\}$ and two partially explored candidate routes $R_1 = \langle s, v_1^1, \ldots, v_q^1\rangle$ and $R_2 = \langle s, v_1^2, \ldots, v_q^2\rangle$ $(1 \le q \le j)$, if $v_q^1 = v_q^2$ and $\mathrm{QRP}(R_1) \cap Q = \mathrm{QRP}(R_2) \cap Q$ and $\cos t(R_1) \le \cos t(R_2)$ holds, R_1 dominates R_2, denoted as $R_1 \prec_Q R_2$.

Lemma 10. *Given a KOR-SP query $(s, Q = \langle q_1, \ldots, q_j\rangle, k)$ and two partially explored routes R_1 and R_2, if $R_1 \prec_Q R_2$, then $ave({R_1}^*) \le ave({R_2}^*)$, where ${R_1}^*$ and ${R_2}^*$ are the optimal feasible routes that are extended from R_1 and R_2, respectively.*

Proof. Suppose $R_1 = \langle s, v_1^1, \ldots, v_q^1\rangle$, $R_2 = \langle s, v_1^2, \ldots, v_q^2\rangle$, and $R_1^* = \langle s, v_1^1, \ldots, v_q^1, v_{q+1}^1, \ldots, v_j\rangle$; since R_1^* is the optimal feasible route extended from R_1, $R = \langle v_1^q, v_{q+1}, \ldots, v_j\rangle$ must be the optimal route from v_1^q to v_j. Because $R_1 \prec_Q R_2$, we have $v_q^1 = v_q^2$, $\mathrm{cost(R_1)} < \mathrm{cost(R_2)}$ and the services provided by route are the same; thus, R_2^* can be represented by $\langle s, v_1^2, \ldots, v_q^2, v_{q+1}, \ldots, v_j\rangle$, and then $\mathrm{cost}(R_1^*) = \mathrm{cost}(R_1) + \mathrm{cost}(R)$ and $\mathrm{cost}(R_2^*) = \mathrm{cost}(R_2) + \mathrm{cost}(R)$, and since

Input: Graph: G(V,E); Request: Q = $\langle r_1, r_2, \ldots, r_q \rangle$; number of routes: K;
Output: top-k routes
1 $\forall v \in V$, initialize v.$HT_{<Q}$ and v.$HT_{>Q}$;
2 $\Psi \longleftarrow \emptyset$;
3 priority queue R $\longleftarrow$ {($\langle s \rangle$, 1)};
4 while R is not empty and $|\Psi| < K$ do
5 P=($\langle v_0, v_1, \ldots, v_{q-1}, v_q \rangle$,x)$\longleftarrow$R.extractMin();
6 Q' $\longleftarrow \sum_{i=1}^{|q|} QRP(v_i) \cap Q$
7 if |Q'| = |NQ| then
8 $\Psi \longleftarrow \Psi U\{R\}$;
9 for each i=1,. . .,q-1 do
10 if QRP($\langle v_0, \ldots, v_i \rangle$)=QRP($v_i$.$HT_{<Q}$.getValue())
11 then
12 P'=($\langle v_0, v'_1, \ldots, v_i \rangle$,-) $\longleftarrow$ v_i.$HT_{>Q}$.getValue().extractMin();
13 R.insert(P');
14 v_i.$HT_{<Q}$.remove();
15 else
16 if QRP(p)=QRP (v_q.$HT_{<Q}$.Value) then
17 v_q.$HT_{<Q}$.add(|QRP(P)|, $\langle v_0, \ldots, v_q \rangle$, QRP(P));
18 $v_{q+1} \longleftarrow$ FindNN(v_q,QPR(Q-Q'),1);
19 R.insert(($\langle v_0, v_1, \ldots, v_q, v_{q+1} \rangle$,1));
20 else
21 v_q.$HT_{>Q}$.add(|QRP(P)|,P);
22 if q>0 then
23 $v'_q \longleftarrow$ FindNN(v_{q-1},PRQ(R-QRP($\langle v_0, \ldots, v_{q-1} \rangle$)),x+1);
24 R.insert(($\langle v_0, v_1, \ldots, v_{q-1}, v_q \rangle$,x+1));
25 return Ψ;

Algorithm 1: KOR-SP(G,s,Q,k).

cost(R_1) < cost(R_2), we have cost(R_1^*) < cost(R_2^*) and the services are the same. □

According to Lemma 10, before the optimal potential route expanded from their dominating route to be one of the top-k optimal routes, there is no need to extend the routes that are dominated. Based on dominating relationship, we put forward KOR-SP method (Algorithm 1).

To check relationship of domination and store dominated route, for each POI v, we recommend two hash tables in the shape of (*key*, *value*) pairs. The first is $HT_{<Q}$ for saving dominating route, where key is the number of services that meet the user's requests, provided by the partially dominating route that has been extended from current POI v and explored, and *value* is the route itself. Another one is $HT_{>Q}$ for saving dominated routes, where *key* is the number of services which meet the request of user, provided by the partially explored dominated route that has been extended from v, and *value* is the route itself, and the dominated routes are ordered according to their average costs in an ascending order. We also keep Ψ as a result set to save top-k optimal routes and a global priority queue R for partially explored routes sorted by their average costs in an ascending order. In addition, for each $P = \langle v_0, v_1, \ldots, v_{q-1}, v_q \rangle$, we introduce an additional attributes x to represent that v_q is the x-th nearest neighbor of v_{q-1} when generating P. Initially, only the source with $x = 1$ is added to the queue R. Then, we begin a loop until R is empty or top-k optimal sorting route has been found.

Pruning Dominated Routes. At each iteration, the algorithm chooses the route with the minimum average cost to be checked. If it has completed all of the user's requests, we will add it to Ψ and reconsider dominated routes (lines 5-14). Otherwise, we inspect if it is dominated or not. For a route $P = \langle v_0, v_1, \ldots, v_{q-1}, v_q \rangle$ to be examined, if P is the first route with QRP(P) that reaches vertex v_q, we add p to $HT_{<Q}$ of v_q and extend it via v_q's nearest neighbor v_{q+1} (lines 14-17). Otherwise, if its QRP(P) belongs to the $HT_{<Q}$ of v_q, it signifies that existing other route with QRP(P) and smaller average cost has been maintained and expanded to the v_q, so that P is dominated. According to Lemma 10, there is no need to extend P anymore; therefore, we add it into $HT_{>Q}$ of v_q rather than the priority queue R (lines 19). Then, we generate a new candidate route from P. Because the x-th nearest neighbor of v_{q-1} has generated in the previous iteration, we need to find the $(x + 1)$-th nearest neighbor of v_{q-1} by invoking algorithm FindNN and create candidate route $\langle v_0, v_1, \ldots, v_{q-1}, v_q' \rangle$ with incremental x and insert it into the priority queue (lines 22-24).

Table 5: Running example of Algorithm 1 for Figure 1.

Step	Routes(route(|NR|, ave(R)),x)
1	$(\langle s\rangle(0,0),1)$
2	$(\langle s, v_2\rangle(1,70),1)$
3	$(\langle s, v_2, v_3\rangle(2,75),1),(\langle s, v_4\rangle(1,90),2)$
4	$(\langle s, v_2, v_3, v_5\rangle(3,83.3),1),(\langle s, v_4\rangle(1,90),2),(\langle s, v_2, v_5\rangle(2,95),2)$
5	$(\langle s, v_2, v_3, v_5, v_9\rangle(5,80),1),(\langle s, v_4\rangle(1,90),2),(\langle s, v_2, v_5\rangle(2,95),2),(\langle s, v_2, v_3, v_6\rangle(4,107.5),2)$
6	$(\langle s,v_2,v_3,v_5,v_9\rangle(5,80),1),(\langle s, v_4\rangle(1,90),2),(\langle s, v_2, v_5\rangle(2,95),2),(\langle s, v_2, v_3, v_6\rangle(4,107.5),2)$
7	$(\langle s, v_4\rangle(1,90),2),(\langle s, v_2, v_5\rangle(2,95),2),(\langle s, v_2, v_3, v_6\rangle(4,107.5),2)$
8	$(\langle s, v_4, v_6\rangle(3,90),1),(\langle s, v_2, v_5\rangle(2,95),2),(\langle s, v_2, v_3, v_6\rangle(4,107.5),2),(\langle s, v_3\rangle(1,120),3)$
9	$(\langle s, v_2, v_5\rangle(2,95),2),(\langle s, v_4, v_6, v_{10}\rangle(4,105),1),(\langle s, v_2, v_3, v_6\rangle(4,107.5),2),(\langle s, v_3\rangle(1,120),3),(\langle s, v_4, v_{10}\rangle(2,210),2)$
10	$(\langle s, v_2, v_5, v_9\rangle(4,85),1),(\langle s, v_4, v_6, v_{10}\rangle(4,105),1),(\langle s, v_2, v_3, v_6\rangle(4,107.5),2),(\langle s, v_3\rangle(1,120),3),(\langle s, v_4, v_{10}\rangle(2,210),2)$

Rethink Dominated Route. After finding the optimal route P, we need to rethink the partial routes that has been explored and been dominated by subroutes of P, because these routes are more likely to be extended to be another optimal route now. Therefore, for every POI v_i in P, if $\langle v_0, \ldots v_i\rangle$ dominates the routes with QRP(p_i) in the $HT_{\succ Q}$ of v_i (line 10), we only consider the dominated route p' with the least average cost and at the end of v_i, because other routes at the end of v_i are dominated by P'. This also accounts for why we use a priority queue as *value* in hash table $HT_{\succ Q}$. Since p's $x + 1$ nearest neighbor has been computed after it is dominated, we set its x to "-" (which means it makes no sense generating candidate route) and add it to the priority queue (lines 10-13). Meanwhile, we remove $\langle v_0, \ldots v_i\rangle$ from the $HT_{\prec Q}$ of v_i; thus, the next candidate route can be extended (line 14).

Example 11 (consider Figure 2). Suppose the given query is $(s, \langle r_1, r_3, r_6, r_7, r_8\rangle, 2)$. Table 5 shows the routes in the priority queue R at each step. At step 1, route $\langle s\rangle$ is added to the queue, and then it is extended via v_2 (s's nearest neighbor in C) that is the collection of POIs with the unfinished services in Q, and no candidate route can be generated. At step 2, $\langle s, v_2\rangle$ is examined, it is extended via v_3 (v_2's nearest neighbor in C) that is the collection of POIs with the unfinished services and candidate route $\langle s, v_4\rangle$ is generated via s's 2^{nd} nearest neighbor in C that is the collection of POIs with the unfinished services. And so on until the exit condition is met.

Finding the x-th Nearest Neighbor. Next, we interpreted how to find the x-th nearest neighbor, the core operation in KOR-SP.

Definition 12 (neighbor distance). Because the POI keywords and the query keywords have a semantic difference, we are not able to choose the nearest neighbor according to the actual distance. And then, we calculate the neighbor distance by combining the semantic difference with the actual distance. And we choose the nearest neighbor according to the neighbor distance. We measure the neighbor distance by

$$sem\left(v, v'\right) = \sum_{1}^{m} \left\{\left(1 - sim\left(k, q\right)\right) \bullet dist\left(v, v'\right)\right\}, \quad (10)$$

where v is the current vertex, v' is the possible nearest neighbor, and m is the number of $sim(k, q)$ that is not equal to 0. Besides, k is keyword of v' and q is the query keyword. Now, given a route $R = \langle s, v_1, \ldots, v_i\rangle$, when we extend R to v_j, we can estimate the cost of v_j as follows:

$$\begin{aligned} T_{i,j} = {} & dist\left(v_i, v_j\right) + sem\left(v_i, v_j\right) \\ & + \min\left\{d \mid d \in \left(HI\left(v_j\right) - \left(v_i, d_{v_i, v_j}\right)\right)\right\}, \end{aligned} \quad (11)$$

where v_i is the current node and v_j represents one of the neighbors of v_i. For example, in Figure 2, we can know that $dist(v_2, v_3) = 80$, $sem(v_2, v_3) = 0$, and $\min(d) = 100$, so $T_{2,3} = 180$.

A straightforward way to find the x-th nearest neighbor of vertex v_i in collection of POIs with services in Q-$Q \cap QRP(\langle v_0, \ldots, v_i\rangle)$ is by using 2-hop labeling technique rather than *Dijkstra*'s search, since FindNN is frequently invoked. Frequent Dijkstra searches on large graphs are practically inefficient. When the number of unfinished services is greater than 1, we do the following steps (lines 3-7). We start from v_i and extend vertices via the equation (line 6) that calculates the average distance between two points, and each vertex's average distance with the current vertex is stored in the ascending sorting queue N (line 7). When the number of unfinished services is less than 1, we perform the following steps (lines 9-11). We directly consider the actual distance between the current point and the nearest possible neighbor as the average distance (line 10) and store it in the queue N (line 11). Finally, we output the corresponding vertex as needed (lines 13-14).

Input: Vertex v_i, collection of POIs with service in Q- $Q \cap QRP(\langle v_0, \ldots, v_i \rangle)$ C, integer x.
Output: The x-th nearest neighbor of v_i in C.

```
1  N ⟵ Ø
2  C ⟵ PRQ(Q- Q ∩ QRP(⟨v_0, ..., v_i⟩));
3  if |Q- Q ∩ QRP(⟨v_0, ..., v_i⟩)| > 1 then
4    for v' in C
5      v_j = min(HI(v').d_{u,v'}).get(u)
6      average = T_{v_i,v'} / |QRP(v') ∪ QRP(v_i) ∩ C|;
7      N.add(v',average);
8  else
9    for v' in C
10     average = dist(v_i, v')
11     N.add(v',average);
12   N.ascending(average);
13   if |N| ≥ x then
14     return N[x];
```

Algorithm 2: FindNN(v_i,C,x).

4.2. Approximate KOR-SP

4.2.1. Domination Conditions. This is different from KOR-SP. The dominating relationship changes such that it does not require the partial routes that have been queried to provide the same services. Next, we will introduce the novel dominating relationship in detail.

We reconsider the dominating relationship. The original requirements are too strict. First, some partial explored routes should have the same end; second, they should provide the same number of the services required by user. For example, in Figure 2, $\langle s \longrightarrow v_1 \longrightarrow v_5 \rangle$ (2,250) and $\langle s \longrightarrow v_2 \longrightarrow v_3 \longrightarrow v_5 \rangle$(3,250) have the same destination, and the numbers of services of route $\langle s \longrightarrow v_1 \longrightarrow v_5 \rangle$ and $\langle s \longrightarrow v_2 \longrightarrow v_3 \longrightarrow v_5 \rangle$ provided are not equal; according to the original definition, they do not satisfy the dominating relationship. Now, we relax this restriction. The number of services of $\langle s \longrightarrow v_1 \longrightarrow v_5 \rangle$ is 2, which is one less than $\langle s \longrightarrow v_2 \longrightarrow v_3 \longrightarrow v_5 \rangle$. If $\langle s \longrightarrow v_1 \longrightarrow v_5 \rangle$ reaches the same number of services of $\langle s \longrightarrow v_2 \longrightarrow v_3 \longrightarrow v_5 \rangle$, the number is 3, and it should add an edge. We assume the cost of the new added edge is ave_weight, that is the average edge weight of all the edges in the graph; as shown in the following, n is the number of edges and $weght_i$ is the i-th edge's weight:

$$ave_{weight} = \frac{\sum_{i=1}^{n} weight_i}{n}. \tag{12}$$

Now, after adding, we can find that route $\langle s \longrightarrow v_2 \longrightarrow v_3 \longrightarrow v_5 \rangle$ has smaller distance cost, so the $\langle s \longrightarrow v_2 \longrightarrow v_3 \longrightarrow v_5 \rangle$ is the dominating route, and $\langle s \longrightarrow v_1 \longrightarrow v_5 \rangle$ is dominated.

Definition 13 (optimize domination). Given a user's request $Q = \{q_1, \ldots, q_j\}$ and two partially explored candidate routes $R_1 = \langle s, v_1^1, \ldots, v_q^1 \rangle$ and $R_2 = \langle s, v_1^2, \ldots, v_q^2 \rangle$ $(1 \le q \le j)$, if $v_q^1 = v_q^2$ and the service number of R_1 is less than or equal to R_2, and $\text{cost}(R_1) + l * \text{ave\_weight} \le \text{cost}(R_2)$ holds, R_1 dominates R_2, denoted as $R_1 \prec_{QN} R_2$, where l is the difference value of the number of services of two routes.

For example, if R_2 have two services and R_1 have four services, in order to achieve the same number of services of R_1, we will add two average edge weights to R_2 as the route's estimated cost. After that, we decide the dominating relationship.

Lemma 14. *Given a query $(s, Q = \langle q_1, \ldots, q_j \rangle, k)$ and two partially explored routes R_1 and R_2, if $R_1 \prec_{QN} R_2$, then $cost(R_1^*) \le cost(R_2^*)$, where R_1^* and R_2^* are the optimal feasible routes that are extended from R_1 and R_2, respectively.*

Proof. Suppose $R_1 = \langle s, v_1^1, \ldots, v_q^1 \rangle$, $R_2 = \langle s, v_1^2, \ldots, v_q^2 \rangle$, $R_1^* = \langle s, v_1^1, \ldots, v_q^1, v_{q+1}, \ldots, v_j \rangle$, and $R_2^* = \langle s, v_1^2, \ldots, v_q^2, v_{q+1}, \ldots, v_i \rangle$. $R_1' = \langle v_q^1, v_{q+1}, v_{q+2} \ldots, v_j \rangle$ and $R_2' = \langle v_q^2, v_{q+2}', \ldots, v_i \rangle$ are the optimal route from R_1 and R_2, respectively, where v_{q+2} and v_{q+2}' have the same service and $v_q^1 = v_q^2$. According to Algorithm 2, $\cos t(v_q^2, v_{q+2}') \le \cos t(v_q^1, v_{q+1}, v_{q+2})$ and $\cos t(v_q^1, v_{q+1}, v_{q+2}) \le \cos t(v_q^2, v_{q+2}') + ave\_weight$, so $\cos t(R_1') \le \cos t(R_2') + l * ave\_weight$. Because of $\cos t(R_1^*) = \cos t(R_1) + \cos t(R_1')$ and $\cos t(R_2^*) = \cos t(R_2) + \cos t(R_2')$, we can know that $\cos t(R_1^*) \le \cos t(R_2^*)$. □

4.2.2. Priority Query. We introduce the priority of query in this section. By analyzing the travel routes of most users, we find that some services inherently have a higher priority than others. Next, we keep these services and their priorities in the dictionary D. For example, a large amount of data shows that users first go to the bank to withdraw money and then spend money, so the priority of withdrawing money is higher than that of consumption. If the services do not have the priority relationship, they are considered the same. By classifying the priority, the services with higher priority are queried firstly,

```
Input: Request: Q = ⟨r1, r2, . . . , rq⟩; N = |Q|: the number of services;
Priority dictionary: D(keyword, priority); initialize S=Q;
Output: priority set
1   n=1, HQ ⟵ Ø, QH ⟵ Ø, t=0, set ⟵ Ø, pre_set ⟵ Ø;
2   if |S|==N then
3      for rn in Q do
4         if (Q- rn)∩rn.D ≠ Ø then
5            HQ= (Q- rn)∩rn.D
6            t=max(HQ.priority)
7            if t < rn.priority then
8               rn.priority=1
9            else
10              rn.priority=0
11        else
12           rn.priority=1
13           QH.add(rn,rn.priority)
14     for rn in QH do
15        if rn.priority==1 then
16           set.add(rn)
17     pre_set⟵set;
18     S ⟵ Ø;
19  else
20     r'=pre_set-S;
21     HQ=(Q- r')∩r'.D;
22     if HQ ≠ Ø then
23        t=max(HQ.priority)
24        for r in HQ do
25           if r.priority==t then
26              S.add(r)
27     pre_set⟵S;
28     set⟵pre_set;
29  return set;
```

Algorithm 3: Priority(Q,D,S).

and the services with the same priority are queried randomly. We call this kind of query partial ordered query based on priority (POQP). In order to facilitate the classification of user request priority, we first plan the priority of services with obvious priority relationship in offline work.

At each iteration, the algorithm classifies the priority of user's requests and queries the high-priority requests firstly. Algorithm 3 assigns priority for query keywords and stores the result in a specific set named *pre_set* (lines 1). Initially, |S| is equal to N and we check every keyword in the Q. Some keywords have a prior relationship. If a keyword's priority level is the highest in these keywords, we define that the keyword's priority level is 1; otherwise, the priority is 0 (lines 2-10). For other keywords, they have no prior relationship, so they are independent. These keywords have no effect on other keywords, so we also define that their priority level is 1 (line 12). After that, we add the keyword to the result set and assign the result set to *pre_set* (lines 14-17). After the initial step, the algorithm finds the service with lower priority through the last finished service and stores the service into the result set (lines 19-28). Finally, the algorithm outputs the result set (line 29). Algorithm 3 is used in Algorithm 1. When the algorithm queries the nearest neighbor, Algorithm 3 can reduce the number of the candidate POIs.

By classifying the queries according to their priority, we can avoid the possibility that the route formed is inconsistent with the actual situation, and at the same time we can reduce the number of candidate POIs.

4.3. Route Replacement. We proposed a routing optimization mechanism, namely, route replacement, to check whether the routing cost can be shorter.

After obtaining the final route $R = \langle s, v_2, v_3, v_5, v_9 \rangle$ produced by Algorithm 1, we propose a postprocessing mechanism named route replacement to refine it. As shown in Figure 2, the route from v_5 to v_9 must go through v_7, and we can find that the services provided by v_2 are the same as those provided by v_7, and we can know that $dist(s, v_3)$ is smaller than the sum of $dist(s, v_2)$ and $dist(v_2, v_3)$. So, we can use v_7 to replace v_2. Finally, the shortest route is $R' = \langle s, v_3, v_5, v_7, v_9 \rangle$. Obviously, the routing cost of R' is smaller than that of R. When we refine the route, we should store these points which satisfies the following requirements: (1) those that have not been searched, (2) those that must be passed by the refined route, and (3) those that provide the same services as the existing points in the refined route. In the refining process, we justify whether to use these points to replace the existing

```
Input: route: R=⟨s, v1, v2, ..., v|R|⟩; query: Q
Output: route: R'
1 v0 ⟵ s, V ⟵ ∅
2 for vi in R and i < |R| do
3     V.add(between(vi,vi+1));
4 for v in V do
5   if QRP(v)∩Q ≠ Ø then
6     for vi in R do
7       if QRP(v)∩Q=QRP(vi)∩Q and cost(vi−1, vi+1)< cost(vi−1, vi)+cost(vi, vi+1) then
8         R'=R.del(vi);
9         R'=R'.add(v);
10      else continue;
11    else continue;
12 return R';
```

Algorithm 4: R2(Q,R).

point, which can reduce the route cost. We should choose the replacing point with more services that satisfies the user's requests, so it can not only reduce the cost but also make the route more concise by reducing the point in the route at the same time. We can understand the process of route replacement in detail through Algorithm 4.

As shown in Algorithm 4, the pseudocode has described the process of route replacement. Firstly, we define the notion that V is for storing POIs (line 1). These POIs are going to be used to replace the POI in the route. For each POI v_i in the route R, we look for the POIs between v_i and v_{i+1}. Then, we add these POIs to the V (lines 2-3). Next, we check each POI in the V and seek out the POI v that is able to meet the user's query (line 4-5). We look for a POI v_i in the route. If v_i and v are able to provide the same services, we compare the cost between $cost(v_{i-1}, v_{i+1})$ and $cost(v_{i-1}, v_i) + cost(v_i, v_{i+1})$. If $\text{cost}(v_{i-1}, v_{i+1})$ is less than $\text{cost}(v_{i-1}, v_i) + \text{cost}(v_i, v_{i+1})$, v_i is able to be replaced by v (lines 6-10).

5. Experimental Evaluation

5.1. Experimental Setup

5.1.1. Datasets. We use two real-world datasets from Zeng [26]. *Singapore* represents Foursquare check-in data collected in Singapore, and *Austin* represents Gowalla check-in data collected in Austin. *Singapore* has 189,306 check-in points, 5,412 locations, and 2,321 users. *Austin* has 201,525 check-ins, 6,176 locations, and 4,630 users. The same as suggested [26, 27], we built an edge between two locations if they were visited on the same date by the same user. The locations not connected by edges were ignored. We filled in the edge costs $t_{i,j}$ by querying the traveling time in minute using Google Maps API under driving mode. The statistic information of the dataset is shown in Table 6.

Both datasets were used in [26], which also studied a route planning problem. The datasets are not small considering the scenario for a daily trip in a city where the user has a limited cost budget. Even with 150 POIs to choose from, the number of possible routes consisting of 5 POIs can reach 70 billion. Compared to our work, Jeffrey [28] evaluated its itinerary recommendation methods using theme park data, where each park contains only 20 to 30 attractions.

Table 6: Dataset statistics.

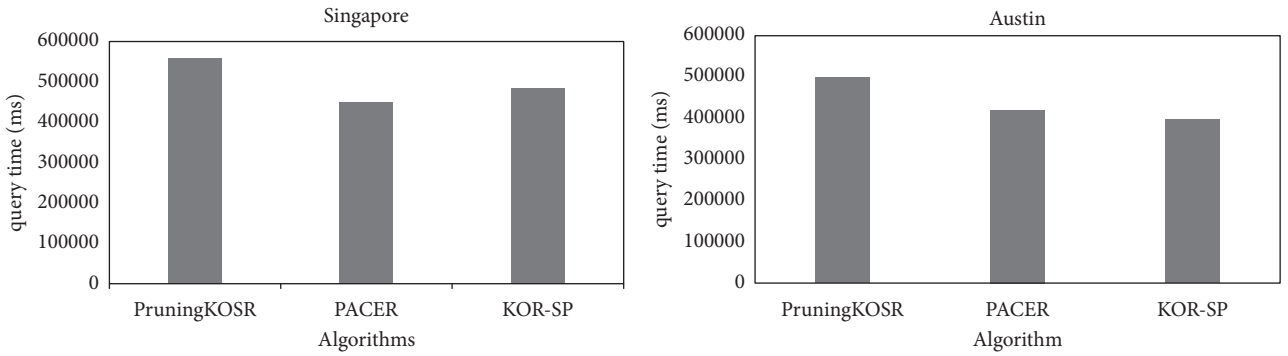
	#POI	#Edges
Singapore	1,625	24,969
Austin	2,609	34,340

5.1.2. Algorithms. We compared the following algorithms. *PACER* [8] models the personalized diversity requirement by retrieving POIs indexes related to feature space and route space, as well as various strategies of pruning search space with user preferences and constraints, and the optimal solution of the top-k path search problem is given. *PruningKOSR* [21] uses dominance relationship to filter temporarily unnecessary routes. *KOR-SP* is our proposed optimal algorithm.

5.1.3. Queries. For each KOR-SP query (s, k), we randomly select a source and an integer k, and then we issue query on all the graphs. In each experiment, 50 random query instances were constructed and the average query time was reported. If the query cannot be stopped within 4200 seconds or fails due to a memory overflow exception, we represent its corresponding query time as INF.

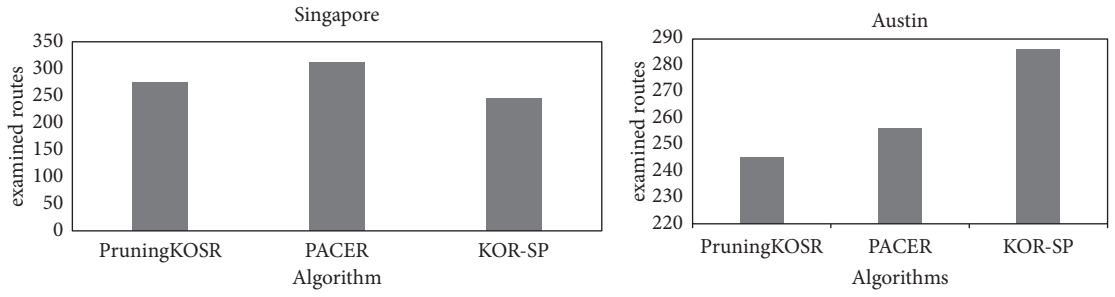
5.1.4. Evaluation Criteria. We evaluate the performance of different methods in four different aspects: the query runtime, the number of examined routes (witnesses), the number of (next) nearest neighbor (shortened as kNN) queries executed by calling Algorithm FindNN, and the cost of the routes.

5.2. Experimental Results. We first evaluate the efficiency of different algorithms answering KOR query in the default



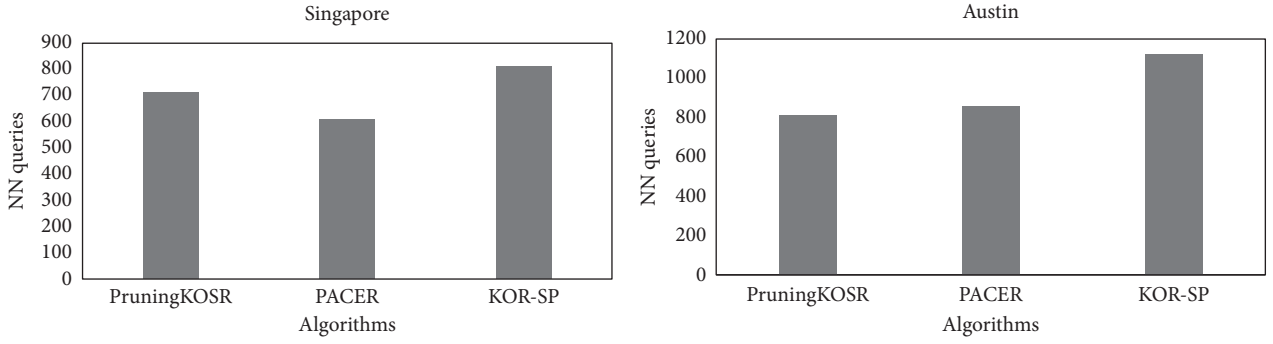
(a) methods on Singapore (b) methods on Austin

Figure 3: Run-time.



(a) methods on Singapore (b) methods on Austin

Figure 4: Examined routes.

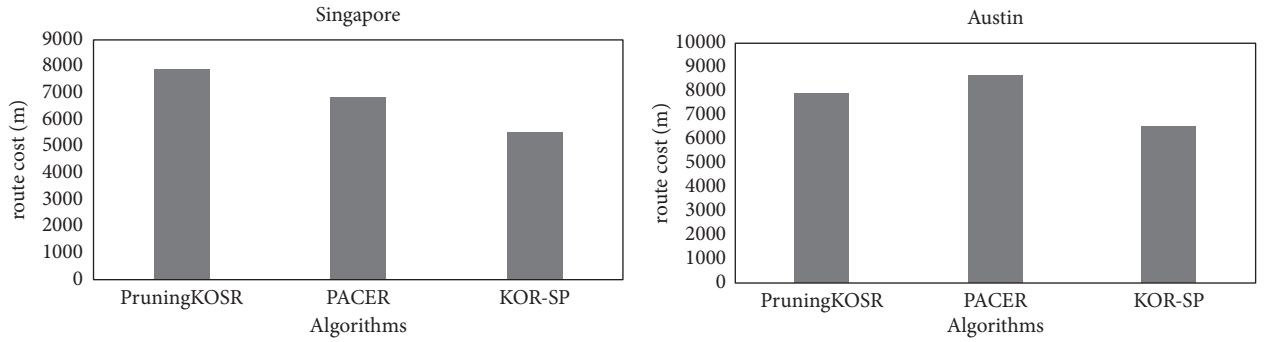


(a) methods on Singapore (b) methods on Austin

Figure 5: NN queries.

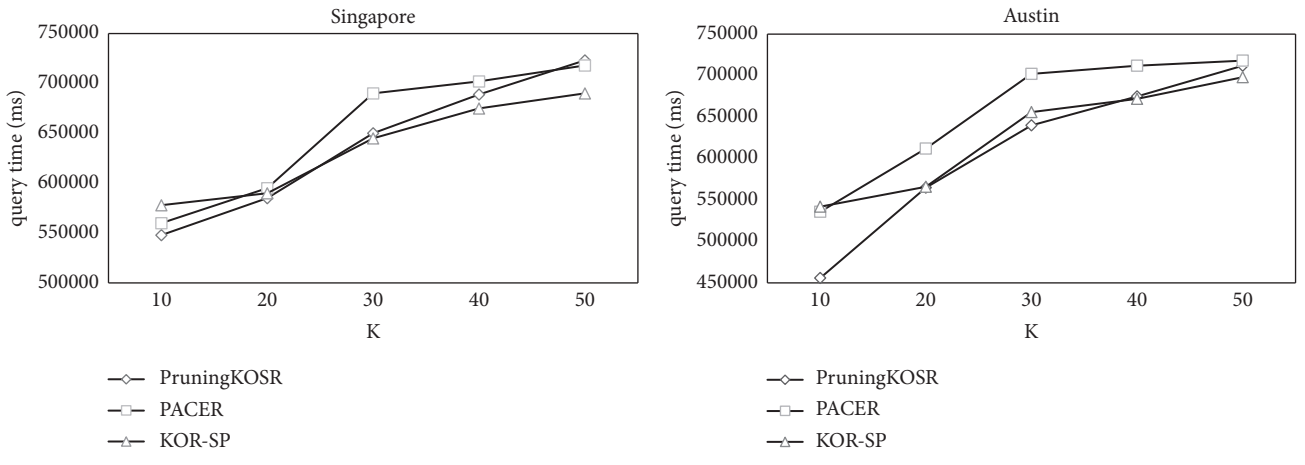
parameter setting on two real graphs and then evaluate the impact of parameters on the results.

5.2.1. Overall Performance under Default Parameter Settings. Figures 3–6 show the performance of three different algorithms on two graphs. The runtime of the algorithms on different graphs is displayed in Figure 3. Since all the algorithms have reduced the searching space, these can return the results on all graphs. At the same time, all the algorithms express efficient queries by using 2-hop label index. Figures 4 and 5 show the number of examined routes and NN queries, respectively. We can find that the number of examined routes in KOR-SP is much fewer than PruningKOSR on all graphs and the number of NN queries is larger than other algorithms. From this phenomenon, we can know the importance of a rich candidate. Because of semantic matching, KOR-SP has more candidate POIs and it can complete the route query with examining fewer routes. The consequence means KOR-SP is better than PruningKOSR. Figure 6 shows the cost of routes. As it is shown, the cost of routes of KOR-SP is much smaller than other algorithms, because this method has more



(a) methods on Singapore (b) methods on Austin

Figure 6: Route cost.



(a) Effect of k on Singapore (b) Effect of k on Austin

Figure 7: Running time vs.k.

candidate POIs that contains some POIs of which distance is shorter by semantic matching.

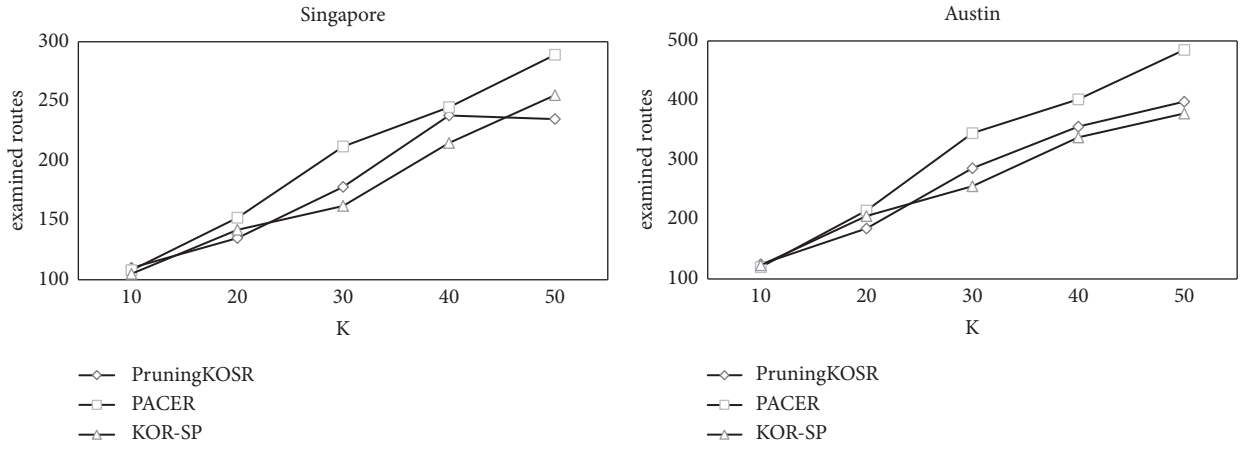
5.2.2. Effect k. Figures 7–10 show the influence of parameter k on the runtime of the three different methods on the two graphs. As shown in Figure 7, we know that all three methods can complete the function of query within the specified time, and with the increase of k, the advantages of KOR-SP are more and more obvious. Figures 8 and 9 show the impact of the number of routes and NN queries checked in different methods on different graphs. We can find that there are far fewer routes and NN queries checked in KOR-SP. Compared with other algorithms, this algorithm has significant advantages under different k conditions. Figure 10 shows the influence of k on routing cost. It can be seen from the figure that, due to semantic matching, routing cost of KOR-SP at different k is the lowest.

5.2.3. Effect β. Figure 11 shows the effect of parameter β. We can find that different β in KOR-SP algorithm has a great influence. The smaller the value of β is, the less the route cost is, because there are more candidate POIs with decreasing the β value, and then we can find more and more nearer neighbors to extend the route.

Figure 12 shows the difference among the four different KOR-SP algorithms. In the figure, the KOR-SP is the basic algorithm, the PKOR-SP is the KOR-SP combining with the priority relationship, the OKOR-SP is the KOR-SP combining with optimize domination, and RKOR-SP is KOR-SP combining with the route replacement. Form Figure 12(a), we can find that the PKOR-SP has the best performance. We know that the priority relationship helps us reducing the size of query. And the route replacement only refines the result of KOR-SP, so RKOR-SP's running time is longer than KOR-SP. From Figure 12(b), we can find that the OKOR-SP has the best performance. At the same time, other algorithms do not make much difference. By comparing other algorithms, we think that the optimized domination has a good effect.

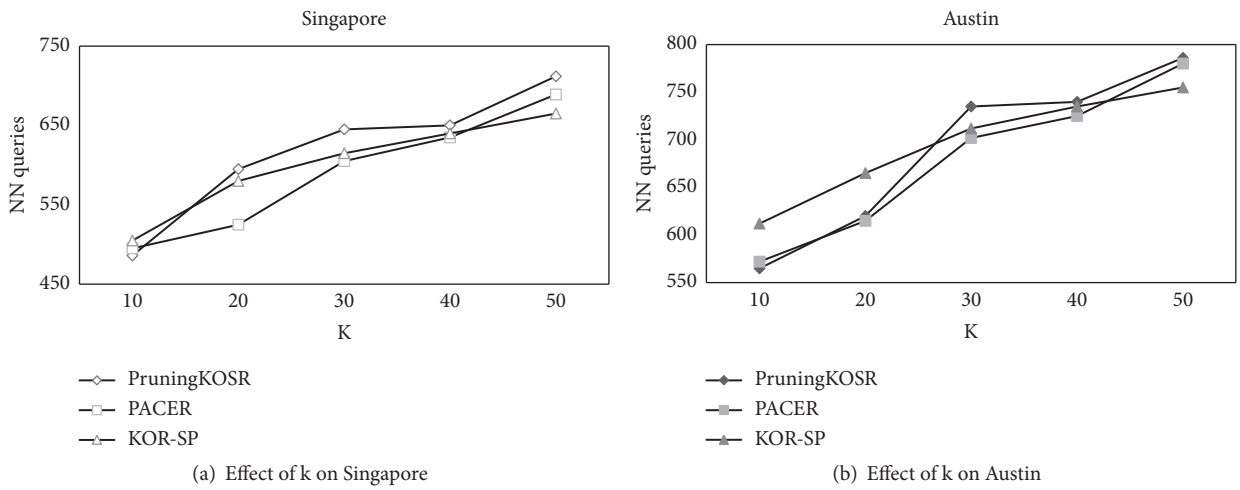
6. Conclusion

In this paper, we study the top-k optimal sequenced routes problem. We propose an efficient algorithm called KOR-SP, based on a novel route dominate relationship and a semantic



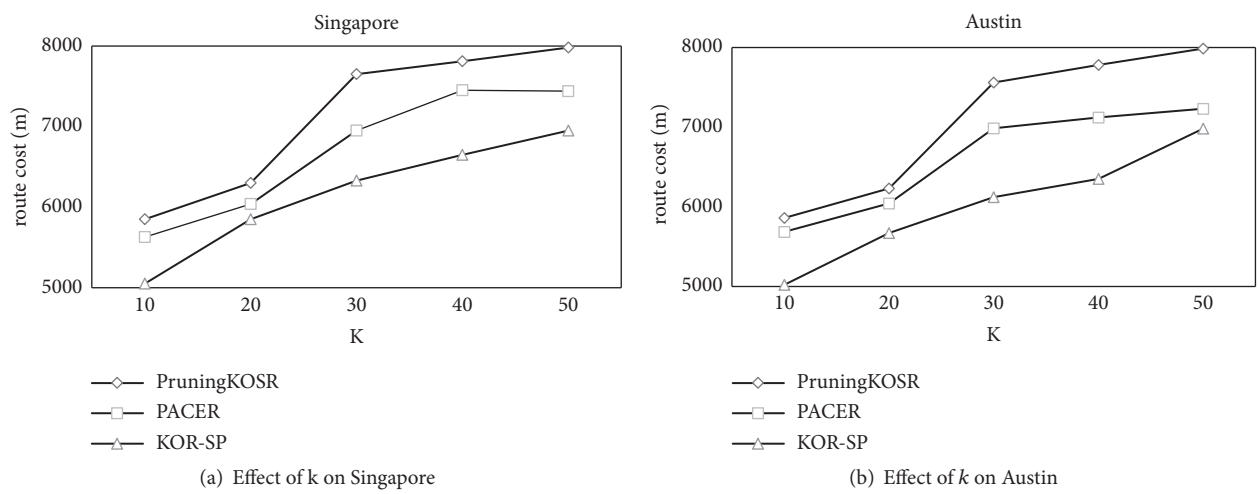
(a) Effect of k on Singapore (b) Effect of k on Austin

Figure 8: Examined routes vs.k.



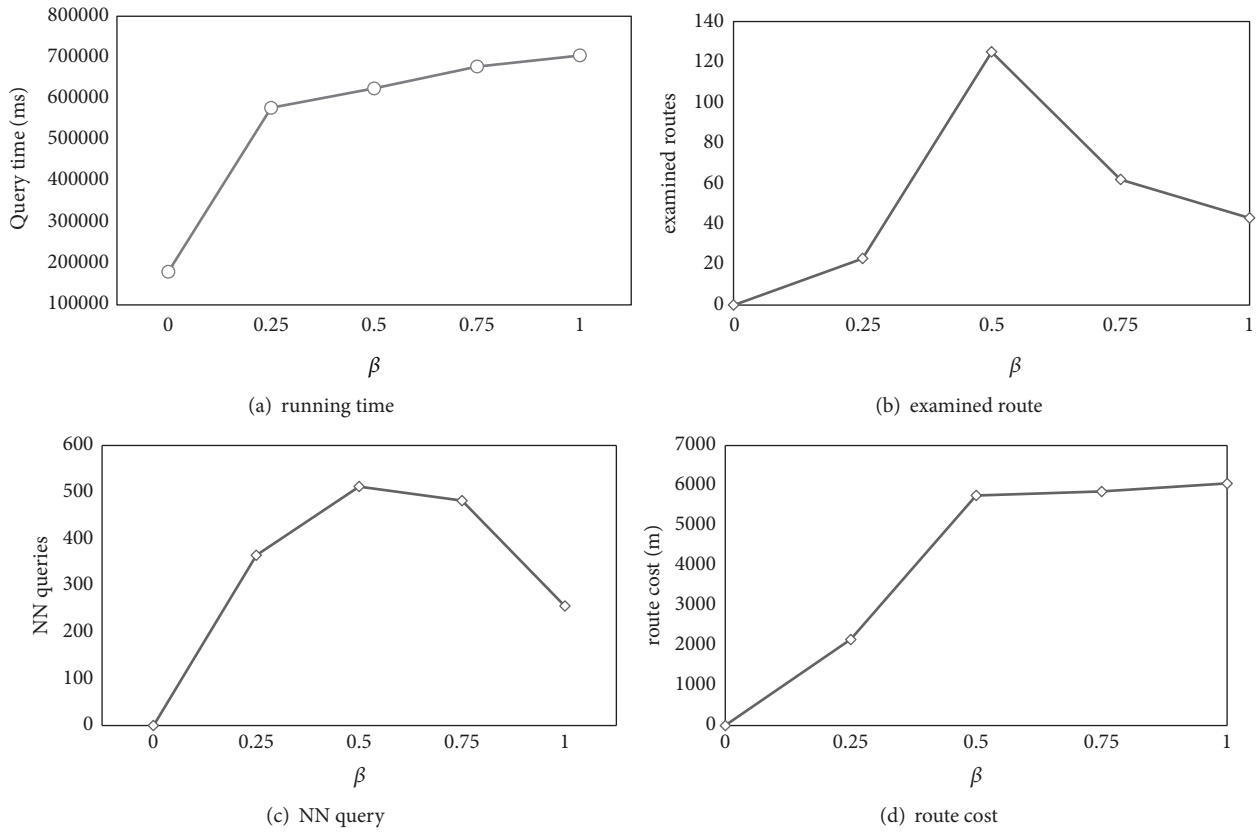
(a) Effect of k on Singapore (b) Effect of k on Austin

Figure 9: NN queries vs.k.



(a) Effect of k on Singapore (b) Effect of k on Austin

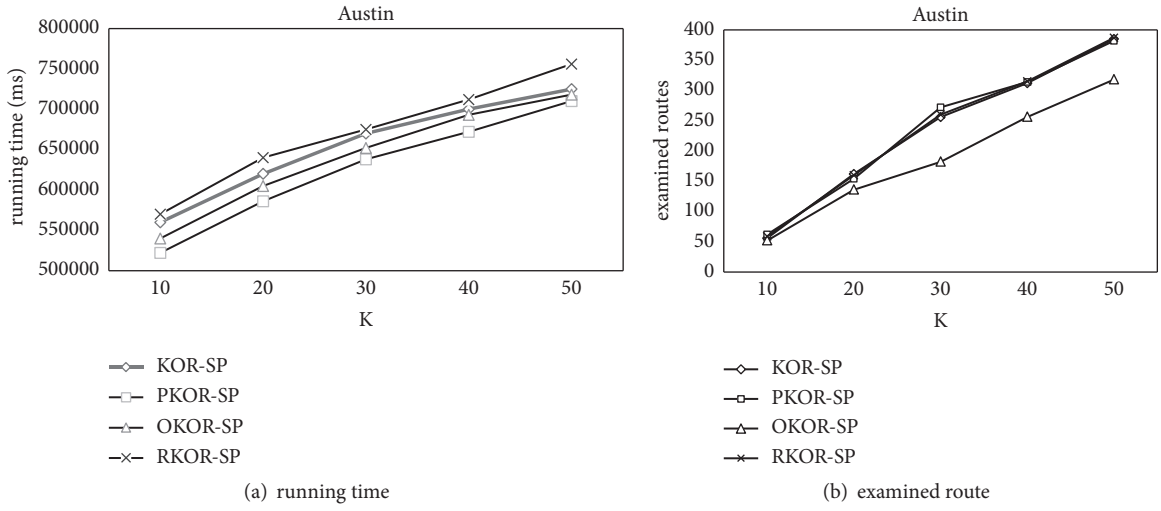
Figure 10: Route cost vs.k.



(a) running time (b) examined route

(c) NN query (d) route cost

Figure 11: Effect of β.



(a) running time (b) examined route

Figure 12: Performance on different KOR-SP algorithms in Austin.

matching by using the LDA model. Extensive experiments on real-world graphs demonstrate that the proposed algorithms are efficient. KOR-SP algorithm improves the flexibility of POI query and provides rich candidate sets for POI query by keyword semantic matching. And KOR-SP algorithm can quickly find the x-th nearest neighbor of the current POI by using FindNN algorithm and reduce the route search space by dominating relation. In addition, the algorithm uses route refinement mechanisms to improve route quality.

As a future work, we plan to study the keyword unordered query which is disordered for the whole, but it is order for part of keywords that have causality.

Data Availability

The graph data used to support the findings of this study are from [8, 26], and the datasets are available at https://github.com/LazyAir/SIGIR18.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research is partially funded by National Natural Science Foundation of China, under Grant no. 61602102 and no. 61872069, and the Fundamental Research Funds for the Central Universities, under Grant no. N161704004.

References

[1] S. H. Fang, E. H. Lu, and V. S. Tseng, "Trip recommendation with multiple user constraints by integrating point-of-interests and travel packages," in *Proceedings of the 2014 15th IEEE International Conference on Mobile Data Management (MDM)*, pp. 33–42, 2014.

[2] E. H. Lu, C. Lin, and V. S. Tseng, "Trip-Mine: an efficient trip planning approach with travel time constraints," in *Proceedings of the 2011 12th IEEE International Conference on Mobile Data Management (MDM)*, pp. 152–161, Lulea, Sweden, June 2011.

[3] J. Dai, C. Liu, J. Xu, and Z. Ding, "On personalized and sequenced route planning," *World Wide Web*, vol. 19, no. 4, pp. 679–705, 2016.

[4] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng, "On trip planning queries in spatial databases," in *Proceedings of the 9th International Symposium on Spatial and Temporal Databases, SSTD 2005*, pp. 273–290, Brazil, August 2005.

[5] J. Eisner and S. Funke, "Sequenced route queries: getting things done on the way back home," in *Proceedings of the 20th ACM Sigspatial International Conference on Advances in Geographic Information Systems*, pp. 502–505, USA, 2012.

[6] Y. Ohsawa, H. Htoo, N. Sonehara, and M. Sakauchi, "Sequenced route query in road network distance based on incremental Euclidean restriction," *Database and Expert Systems Applications*, vol. 7446, no. 1, pp. 484–491, 2012.

[7] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi, "The optimal sequenced route query," *The VLDB Journal*, vol. 17, no. 4, pp. 765–787, 2008.

[8] H. Liang and K. Wang, "Top-k route search through submodularity modeling of recurrent POI features," in *Proceedings of the 41st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2018*, pp. 545–554, USA, July 2018.

[9] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Hierarchical hub labelings for shortest paths," *Algorithms–ESA*, vol. 7501, pp. 24–35, 2012.

[10] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proceedings of the 2013 ACM SIGMOD Conference on Management of Data*, pp. 349–360, USA, 2013.

[11] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *Siam Journal on Computing*, vol. 32, no. 5, pp. 937–946, 20032.

[12] R. Bramandia, B. Choi, and W. K. Ng, "On incremental maintenance of 2-hop labeling of large graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 5, pp. 682–698, 2010.

[13] J. J. Ying, W. Kuo, V. S. Tseng, and E. H. Lu, "Mining user check-in behavior with a random walk for urban point-of-interest recommendations," *ACM Transactions on Intelligent Systems and Technology*, vol. 5, no. 3, pp. 1–27, 2014.

[14] E. H.-C. Lu, W.-C. Lee, and V. S.-M. Tseng, "A framework for personal mobile commerce pattern mining and prediction," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, pp. 769–782, 2012.

[15] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[17] E. M. Arkin and R. Hassin, "Approximation algorithms for the geometric covering salesman problem," *Discrete Applied Mathematics: The Journal of Combinatorial Algorithms, Informatics and Computational Sciences*, vol. 55, no. 3, pp. 197–218, 1994.

[18] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. Teng, "On trip planning queries in spatial databases," in *Proceedings of the International Symposium on Spatial and Temporal Databases*, vol. 31 of *Lecture Notes in Computer Science*, no.1, pp. 273–290, Springer, Boston, MA, USA, 2005.

[19] K. Menger, "Ergebnisse eines mathematischen Kolloquiums," *Monatshefte Fur Mathematik - Monatsh Math*, vol. 39, no. 1, 1932.

[20] E. Ahmadi and M. A. Nascimento, "A mixed breadth-depth first search strategy for sequenced group trip planning queries," in *Proceedings of the 16th IEEE International Conference on Mobile Data Management*, pp. 24–33, USA, 2015.

[21] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k optimal sequenced routes," *International Council for Open and Distance Education*, pp. 569–580, 2018.

[22] L. Tang, D. Cai, Z. Duan, J. Ma, M. Han, and H. Wang, "Discovering travel community for POI recommendation on location-based social networks," *Complexity*, vol. 2019, Article ID 8503962, 8 pages, 2019.

[23] J. Li, C. Liu, J. X. Yu, Y. Chen, T. Sellis, and J. S. Culpepper, "Personalized influential topic search via social network summarization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 7, pp. 1820–1834, 2016.

[24] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, no. 4-5, pp. 993–1022, 2003.

[25] M. Babenko, A. V. Goldberg, A. Gupta, and V. Nagarajan, "Algorithms for hub label optimization," *ACM Transactions on Algorithms (TALG)*, vol. 13, no. 1, pp. 1–17, 2016.

[26] Y. Zeng, X. Chen, X. Cao, S. Qin, M. Cavazza, and Y. Xiang, "Optimal route search with the coverage of users' preferences," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI 2015*, pp. 2118–2124, Argentina, July 2015.

[27] X. Cao, L. Chen, G. Cong, and X. Xiao, "Keyword-aware optimal route search," *VLDB Endowment*, vol. 5, no. 11, pp. 1136–1147, 2012.

[28] K. H. Lim, J. Chan, S. Karunasekera, and C. Leckie, "Personalized itinerary recommendation with queuing time awareness," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2017*, pp. 325–334, Japan, August 2017.