Xu HF, Gu Y, Qi JZ *et al.* Diversifying top-k routes with spatial constraints. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 34(4): 818–838 July 2019. DOI 10.1007/s11390-019-1944-6

Diversifying Top-k Routes with Spatial Constraints

Hong-Fei Xu[1], *Student Member*, *CCF*, Yu Gu[1], *Senior Member*, *CCF*
Jian-Zhong Qi[2], *Member*, *ACM*, Jia-Yuan He[2], and Ge Yu[1,*], *Fellow*, *CCF*, *Member*, *ACM*, *IEEE*

[1] *College of Computer Science and Engineering, Northeastern University, Shenyang 110169, China*
[2] *Department of Computing and Information Systems, The University of Melbourne, Melbourne, VIC 3010, Australia*

E-mail: xuhongfei_neu@163.com; guyu@mail.neu.edu.cn; jianzhong.qi@unimelb.edu.au
hjhe@student.unimelb.edu.au; yuge@mail.neu.edu.cn

Received May 15, 2018; revised May 15, 2019.

Abstract Trip recommendation has become increasingly popular with the rapid growth of check-in data in location-based social networks. Most existing studies focused only on the popularity of trips. In this paper, we consider further the usability of trip recommendation results through spatial diversification. We thereby formulate a new type of queries named spatial diversified top-k routes (SDkR) query. This type of queries finds k trip routes with the highest popularity, each of which starts at a given starting point, consumes travel time within a given time budget, and passes through points of interest (POIs) of given categories. Any two trip routes returned are diversified to a certain degree defined by the spatial distance between the two routes. We show that the SDkR problem is NP-hard. We propose two precise algorithms to solve the problem. The first algorithm starts with identifying all candidate routes that satisfy the query constraints, and then searches for the k-route combination with the highest popularity. The second algorithm identifies the candidate routes and builds up the optimal k-route combination progressively at the same time. Further, we propose an approximate algorithm to obtain even higher query efficiency with precision bounds. We demonstrate the effectiveness and efficiency of the proposed algorithms on real datasets. Our experimental results show that our algorithms find popular routes with diversified POI locations. Our approximate algorithm saves up to 90% of query time compared with the baseline algorithms.

Keywords trip recommendation, spatial diversity, route search, check-in data

1 Introduction

Location-based social networks (LBSN) such as Flickr and Foursquare allow users to check in and share their activities at various points of interest (POIs). This creates a large volume of check-in data, which provides rich information on POI popularity and people's travel patterns at different POIs. Intuitively, a POI with more check-ins is more popular. This offers a great opportunity to build trip recommendation systems based on check-in data. A trip recommendation system usually returns the k most popular routes given a set of query constraints, such as user location, preferred types of POIs, and/or time budget for the trip[1−6].

In this paper, we aim to provide a novel trip recommendation service that returns the top-k trip routes which are not only popular but also diversified. A recent user survey[7] shows that query users have strong preference on spatial diversity in query results of spatial recommendation systems. It means users expect the spatial recommendation system presents the results that are far away from each other and are distributed in different areas. In the context of trip recommendation, we find that trip routes that are spatially close usually provide users with similar travel experiences since co-located POIs have similar geographical environment and features. Therefore, we focus on spatial diversity in trip recommendation, i.e., we aim to return trip routes where POIs in different routes are far away from each

Regular Paper

This work was supported by the National Key Research and Development Program of China under Grant No. 2018YFB1003404, the National Natural Science Foundation of China under Grant Nos. 61872070, U1435216, U1811261, and 61602103, and the Fundamental Research Funds for the Central Universities of China under Grant No. N171605001.

*Corresponding Author

other. Our study is motivated by the following applications.

Suppose that Alice is planning a 3-day trip in New York. On different days of the trip, she would like to explore different regions. And she expects each day's trip including cafes, parks and markets. Hence, she may pose a query: "find the three most popular trip routes each of which starts from my hotel, covers at least a cafe, a park and a market, and the travel time is less than 10 hours." At the same time, in order to avoid the result routes close to each other, Alice may further require "the POIs (except for the starting point) in different routes need to be at least 3 km away from each other." Consider another example that a travel agency wants to launch five family-day trip routes. To satisfy with the different interests of family members, each route should contain multiple categories of attractions. Thus, the travel agency may pose a query: "find the five most popular trip routes each of which starts from the central train station, covers a historic site, a landscape garden and an amusement park, and the travel time is less than eight hours." As attractions in the same category may have different surroundings in different regions, the travel agency wants the routes to be spatially distant, which allows users to choose routes according to their different regional preferences. Thus, a further requirement is that "the POIs (except for the starting point) in different routes returned need to be at least 5 km away from each other". In addition, many practical applications (e.g., hazardous material transportation, emergency evacuation and cash-in-transit) also need to find several routes spatially distant.

We formulate the queries in the above examples as the spatial diversified top-k routes (SDkR) query. An SDkR query q is a 5-tuple: $q = (v_{\mathrm{q}}, t_{\mathrm{q}}, S_{\mathrm{cat}}, k, \sigma)$. The query returns k trip routes with the highest popularity, each of which starts at a POI v_{q}, has an estimated travel time within the given time budget t_{q}, and covers the query POI categories given by S_{cat}. Meanwhile, the spatial distance between the POIs in any two routes returned must be at least σ (i.e., spatial diversity constraint), which is a user given distance threshold. We use Euclidean distance to measure the minimum distance between POIs. We do not use the road network distance because two POIs with a large network distance due to congested traffic or no direct road connection may still be spatially close, e.g., on two sides of a river, which is against our goal of high spatial diversity.

Fig.1 illustrates an SD3R query for Alice. Suppose that there are four trip routes satisfying her query constraints, denoted by r_1, r_2, r_3 and r_4. Four 3-route combinations are formed: $\{r_1, r_2, r_3\}$, $\{r_1, r_2, r_4\}$, $\{r_1, r_3, r_4\}$ and $\{r_2, r_3, r_4\}$. The POIs in r_1 and r_2 are relatively close and the two routes may not satisfy the POI distance constraint σ (i.e., 3 km). The POIs in $\{r_1, r_3, r_4\}$ and $\{r_2, r_3, r_4\}$ on the other hand are far away from each other in different areas. Suppose both $\{r_1, r_3, r_4\}$ and $\{r_2, r_3, r_4\}$ satisfy the POI distance constraint, and the POIs in r_1 are more popular. Then, $\{r_1, r_3, r_4\}$ is returned as the query answer.

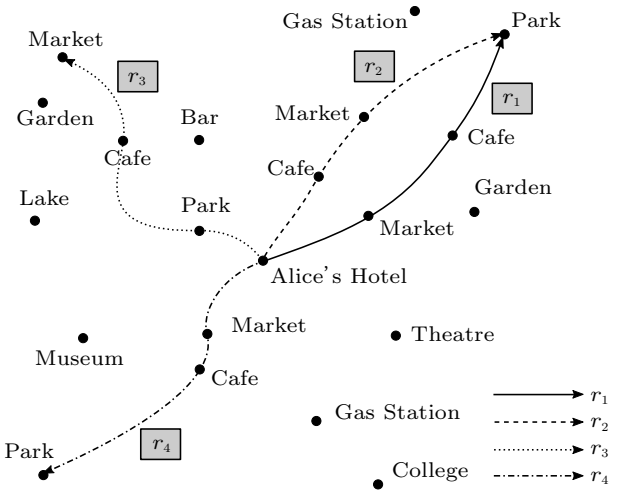


Fig.1. An SD3R query.

The SDkR query has not been studied before. A straightforward query algorithm is to 1) find every trip route that satisfies the starting point, travel time budget, and POI category constraints, 2) enumerate all possible k-route combinations, 3) filter out those k-route combinations that do not satisfy the POI distance constraint, and 4) return the remaining k-route combination with the highest popularity. This algorithm suffers in efficiency when the number of k-route combinations is large. To reduce the number of k-route combinations to be considered, we propose a graph data structure named feasible route graph (FRG), where the nodes represent routes that satisfy the query constraints imposed on the routes, and an edge between two nodes represents that the two corresponding routes satisfy the POI distance constraint. Then, finding the optimal top-k routes is equivalent to finding a maximum weight clique of size k. We propose a precise algorithm named the Two-Stage Precise Search (TSS-P) algorithm to traverse FRG, during which non-promising k-route combinations are pruned. When the traversal is done, the k-route combination with the highest popularity is found and returned.

Generating all trip routes that satisfy the query constraints may be expensive as well, especially when the number of POIs or the travel time budget is large. This motivates us to propose another algorithm named the Single-Stage Incremental Search (3S-I) algorithm that generates routes and k-route combinations at the same time. This algorithm maintains a set of partial solutions and the current optimal k-route combination. A partial solution is a set of less than k routes where any two routes satisfy the POI distance constraint. The algorithm incrementally inserts more routes into the partial solutions as they are generated, during which pruning heuristics are applied to early prune non-promising partial solutions. After the pruning process, if the set of partial solutions is empty, we stop generating a new route and return the current optimal k-route combination as the query answer. To further improve the query efficiency, we adapt 3S-I to obtain an approximate algorithm named the Single-Stage α-Approximate Search (3S-α) algorithm, which uses a popularity upper bound and an approximation ratio α to help early terminate the search once an α-approximate query answer is found.

In summary, this paper makes the following contributions.

• We propose a new type of queries, the spatial diversified top-k routes (SDkR) query, which introduces spatial diversity to top-k routes search.

• We propose three algorithms to process the SDkR query: TSS-P, 3S-I and 3S-α. The TSS-P algorithm processes the query by first generating all the routes that satisfy the query constraints, and then returns the optimal k-route combination that satisfies the spatial diversity constraint. To avoid generating non-promising routes and k-route combinations, 3S-I and 3S-α both form k-route combinations while generating the routes satisfying the query constraints. In particular, 3S-I guarantees to find the k-route combination satisfying the distance constraint with the highest popularity, while 3S-α is an approximate algorithm that early terminates once an α-approximate query answer is obtained. Here, α is a user-defined ratio.

• We conduct extensive experiments using real datasets to evaluate the performance of the proposed algorithms. The results show that the proposed algorithms can process SDkR queries effectively and efficiently.

The rest of the paper is organized as follows. Section 2 reviews related studies. Section 3 formulates the SDkR query. Section 4 details the TSS-P algorithm. Section 5 details the algorithms 3S-I and 3S-α. Section 6 presents experimental results. Section 7 concludes the paper and describes the future research directions.

2 Related Work

We review three categories of studies related to SDkR queries: route search based on trajectories, route search based on POIs, and query result diversification.

2.1 Route Search Based on Trajectories

Trajectory-based route search assumes a set of candidate trajectories and aims to find the trajectories that best match certain search criteria (e.g., query locations and query keywords)[8−13]. For example, Chen *et al.*[8] found k trajectories that best match a set of given points. Here, how well a trajectory matches a given point is defined by the distance between the point and its nearest POI in the trajectory. Shang *et al.*[9] proposed the trajectory search by regions (TSR) query, which aims to retrieve the trajectory that has the highest spatial density correlation with a region specified by a query user. Here, the spatial density correlation between a trajectory and a region is computed as the aggregation of two types of correlations: correlation in spatial object density and correlation in spatial distance. Shang *et al.*[13] proposed the trajectory-to-location join (TL-Join) query, which finds all (trajectory, location) pairs with spatio-temporal correlation above a query threshold. They proposed a parallel collaborative algorithm with effective pruning techniques and a heuristic scheduling strategy to process the TL-Join queries. Wei *et al.*[14] found the most popular k routes that pass a sequence of user-specified locations, based on a collection of trajectories generated by LBSN users. Unlike GPS trajectories that have a high sampling frequency, the trajectories generated by LBSN users are usually low-sampled due to the applications' characteristics. To overcome the uncertainty in LBSN trajectories, they built a route inference framework which constructs a route graph from LBSN trajectories. Zheng *et al.*[11] assumed that every trajectory is associated with a set of keywords and proposed the top-k spatial keyword (TkSK) query for trajectories, which aims to retrieve the trajectories that are spatially close to the query point and cover the query keywords. In these studies, candidate trajectories are either given in a trajectory dataset or derived from existing trajecto-

ries. This differs from our work where candidate trajectories are created at query time by connecting POIs.

2.2 Route Search Based on POIs

Route search based on trajectories only considers observed trajectories in the dataset, while neglects the potential routes that can be derived from the underlying road network graph. Thus, some studies tried to use the existing POIs or extract POIs from existed trajectories to find the top-k optimal routes for users[15−21]. Cao *et al.*[15] assumed POIs associated with keywords. They found k routes with the highest objective function scores, each of which covers a set of query keywords and costs time less than a query time budget. Shang *et al.*[17] proposed the collective travel planning (CTP) query. Given a set of source points and a destination point, a k-CTP query finds the lowest-cost route connecting the source points and the destination point using at most k meeting points. Wen *et al.*[18] proposed a route recommendation framework which allows users to specify a geographical range and a set of preference keywords. To make a recommendation, they first found feasible routes that satisfy the hard constraints, i.e., enclosed by the query range and related (or partially related) to the query keywords. Then they ranked the feasible routes w.r.t. three factors, i.e., attractiveness, visiting time, and the influence of the route generator, and returned the top ranked route(s) to the query user. Shang *et al.*[20] proposed to find unobstructed paths (i.e., the lowest probability of traffic congestion) for query users, where the traffic condition of a spatial network is derived using historical travel trajectories of commuters. Zheng *et al.*[22] investigated the problem of clue-based route search (CRS) in a road network. The CRS query allows a user to provide imprecise constraints (e.g., similar keywords of the POIs and the rough distance between two POIs in the route) along the route such that the best matching route w.r.t. the imprecise constraints is returned. Xu *et al.*[23] proposed an algorithm that continuously returns the fastest route for users in dynamic road network. These studies do not consider the spatial distances between the retrieved routes. Thus, their solutions are not applicable for our problem.

2.3 Query Result Diversification

Result diversification has been widely studied to enhance result usability in many areas including information retrieval[24−26], recommender systems[27], and social network searching[28−29]. It aims to generate a result set that minimizes the pairwise similarity between any two elements in the set while preserving the relevance of the result elements to the query. General frameworks for query result diversification are introduced in [30, 31]. An early model named the maximal marginal relevance (MMR) model[30] was proposed for result diversification in text retrieval and summarization. It re-ranks elements in the result set by choosing elements sequentially according to their ranking scores. Vieira *et al.*[31] proposed a framework for optimizing result diversification methods and the evaluation of various diversification techniques. This framework allows users to adjust the trade-off between result relevance and diversity. Chen and Cong[27] proposed the diversity-aware top-k subscription (DAS) query, which aims to provide users a few up-to-date representative documents with a wide coverage of different aspects of their query topics over a stream of documents. Qin *et al.*[32] considered the diversified top-k search problem as a traditional top-k search problem augmented with a constraint where the similarity between any two result elements must not exceed a user-given threshold. They proposed a solution framework (i.e., an incremental search strategy) based on a diversity graph where every node represents a result element. Every node has a weight representing their relevance to the query. An edge is created to connect two nodes if their similarity exceeds a given threshold. Once the diversity graph is constructed, solving the diversified top-k search problem is reduced to finding the size-k maximum weight independent set (MWIS)[33] in the graph. One of our algorithms adopts the incremental search strategy used by Qin *et al.*[32], but we do not rely on the diversity graph. Particularly, Qin *et al.*'s algorithm performs a graph traversal each time when a new result element is found. Our algorithm avoids such repetitive graph traversals through caching intermediate results, and hence enhances the query efficiency. We will compare the performance of the algorithms via experiments.

Spatial diversity has been considered in more recent studies. Jain *et al.*[34] proposed the k-nearest diverse neighbor (kNDN) search. They assumed data points associated with a set of certain attributes. The diversity of a set of data points is computed based on the classical Gower coefficient, wherein the difference between two points is defined as a weighted average difference of their attribute values. Lee *et al.*[35] introduced spatial diversity to the nearest neighbor queries implicitly by finding the nearest surrounding objects. They

assumed polygon data objects and found the set of data objects that fully surround a query point. The angular positions w.r.t. the query point and the data objects are used to solve the problem. This imposes an implicit constraint to diversify the result objects spatially. Kucuktunc and Ferhatosmanoglu[36] studied the kNDN search where the diversity of two objects is defined as the inverse of their angular similarity. They proposed an index-based diverse browsing method based on the R-tree and the best-first kNN search algorithm. Ference *et al.*[37] also studied kNDN search based on angular diversity. They defined the angular diversity of a result set as the variance of the angles of the set elements to the query point, and they proposed two heuristic algorithms named DistBrow and DivBrow. Zhang *et al.*[38] studied the diversified spatial keyword query in road networks. They found k objects that 1) cover a set of query keywords, 2) have the maximum proximity to the query point, and 3) have the maximum diversity (i.e., network distances among the k objects). They proposed a signature-based inverted index, based on which both keyword-based and diversity-based pruning techniques are used to prune the search space. The above studies all focus on the diversity among individual objects. We study the diversity among routes formed by multiple POIs, which cannot be handled by the existing methods.

3 Preliminaries

We start with a few definitions and the query formulation. We list the frequently used symbols in Table 1.

Check-In Dataset. Let CR be a set of user check-in records, where each record $cr \in CR$ is represented as a 6-tuple:

$$cr = (uid, pid, t_{\mathrm{c}}, lon, lat, l_{\mathrm{cat}}).$$

Here, *uid* represents a unique ID of the check-in user, *pid* represents a unique ID of the check-in POI, t_{c} represents the check-in time, *lon* and *lat* represent the longitude and the latitude of the POI, respectively, and l_{cat} represents a set of categories that the POI belongs to. To illustrate our algorithms, we use eight popular categories in the POI classification used by Foursquare, including "Arts & Entertainment", "Shops", "Food", "Nightlife", "Travel & Transport", "Parks & Outdoors", "College & University", and "Building". For simplicity, we denote them by $C_1, C_2, ..., C_8$, respectively. Each POI may belong to multiple categories, e.g., a street market may belong to "Shops" and "Food". Other similar POI categorization systems can be used as well.

Table 1. Notation

Symbol	Description
G	Weighted directed graph of POIs
v_i	A node (POI) in G
$v_i.p_j$	Popularity of v_i w.r.t. a category c_j
$e_{i,j}$	Directed edge from v_i to v_j in G
$e_{i,j}.w$	Weight of $e_{i,j}$
r_i	A route
$r_i.l_{\mathrm{cat}}$	Categories covered by r_i
$r_i.t$	Time cost of r_i
S_{cat}	A set of categories
$pop(r_i)$	Popularity of r_i w.r.t. S_{cat}
q	An SDkR query
Q	A queue to store the partial routes

A check-in record in the Foursquare dataset is: $(7, 91, 3/10/2015\ 17{:}06{:}22, -122.41, 37.79, C_6)$. Here, 7 is the *uid*, 91 is the *pid*, "3/10/2015 17:06:22" is the check in time, -122.41 and 37.79 are the longitude and the latitude, respectively, and C_6 (i.e., Parks & Outdoors) is the category.

POI Graph. We extract the POIs from CR and model them as a POI graph $G = (V, E)$ as a directed and complete graph using the method proposed by Zhang *et al.*[3], where $V = \{v_1, v_2, ..., v_n\}$ is a set of n nodes and E is a set of edges. Each node v_i in G represents a POI and each edge $e_{i,j}$ represents a route from v_i to v_j. The weight of $e_{i,j}$, denoted by $e_{i,j}.w$, is the travel time from v_i to v_j. In this paper, we allow different travel time for different traveling directions between a pair of nodes, i.e., $e_{i,j}.w$ and $e_{j,i}.w$ may have different values. In the experiments, we assume driving as the means of transportation between POIs and estimate the edge weights using Google Map API.

Fig.2 illustrates a fraction of a POI graph, where each node is labelled with their coordinates and each edge is labelled with the edge weight. Every node v_i is represented by a 5-tuple:

$$v_i = (pid, lon, lat, t, l_{\mathrm{p}}).$$

Here, *pid* is the ID of v_i, *lon* and *lat* are the coordinates of v_i, t is the average visiting time that users spend at v_i, and l_{p} is the list of popularity values of v_i w.r.t. its different categories in the form of $(v_i.c_j : v_i.p_j)$, where c_j represents the j-th category that v_i belongs to and p_j represents the corresponding popularity value.

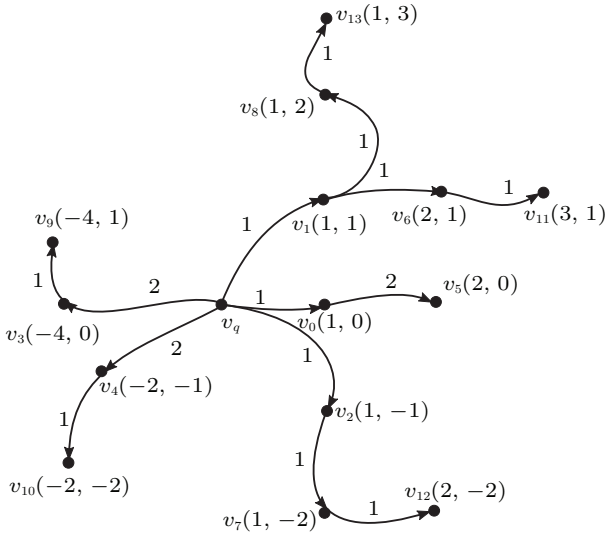


Fig.2. Example of POI graph.

We compute the popularity of v_i w.r.t. a category c_j, $v_i.p_j$, as the ratio of the number of visits at v_i over the total number of visits at POIs that belong to the category c_j. Other methods of deriving POI popularities can also be used here.

In Table 2, the popularity list of v_0 is $v_0.l_{\mathrm{p}} = \{(C_1 : 0.09), (C_2 : 0.08)\}$. This indicates that the popularity of v_0 is 0.09 for "Arts & Entertainment" and 0.08 for "Shops". A larger popularity value means a higher popularity, which means the POI has a strong appeal for users.

Table 2. Popularity Values of the POIs in Fig.2

POI	Popularity	POI	Popularity
v_0	$\{(C_1 : 0.09), (C_2 : 0.08)\}$	v_7	$\{(C_2 : 0.05), (C_3 : 0.03)\}$
v_1	$\{(C_2 : 0.07)\}$	v_8	$\{(C_1 : 0.08)\}$
v_2	$\{(C_1 : 0.08), (C_2 : 0.03)\}$	v_9	$\{(C_2 : 0.08), (C_3 : 0.07)\}$
v_3	$\{(C_1 : 0.04)\}$	v_{10}	$\{(C_1 : 0.06), (C_3 : 0.06)\}$
v_4	$\{(C_2 : 0.06)\}$	v_{11}	$\{(C_1 : 0.05), (C_3 : 0.10)\}$
v_5	$\{(C_2 : 0.09), (C_3 : 0.08)\}$	v_{12}	$\{(C_1 : 0.02), (C_3 : 0.07)\}$
v_6	$\{(C_1 : 0.08)\}$	v_{13}	$\{(C_3 : 0.07)\}$

Trip Route. A trip route r has the form $r = (v_1, v_2, ..., v_m)$, where v_1–v_m represent an ordered list of nodes passed by the route. We use $r.v_j$ to denote the j-th node in r. The categories of r, denoted by $r.l_{\mathrm{cat}}$, are the union of the categories of the nodes in r. The time cost of r, denoted by $r.t$, is the sum of the edge weights between the adjacent nodes in r plus the visiting time at each node in r. Since each POI may belong to multiple categories, for a trip route there may exist multiple POIs containing a certain category. We assume the popularity of r w.r.t. a category c_j, denoted by $r.p_j$, is capped by the most popular POI w.r.t. c_j in r, i.e., $r.p_j = \max\{v.p_j | v \in r\}$. The corresponding node is named as the delegate node of r for category c_j, denoted by $r.v(c_j)$. Given a category set S_{cat} that satisfies $S_{\mathrm{cat}} \subset r.l_{\mathrm{cat}}$, the overall popularity of a route r w.r.t. S_{cat}, denoted by $pop(r, S_{\mathrm{cat}})$, is defined as the sum of the popularities of r w.r.t. each category in S_{cat}:

$$pop(r, S_{\mathrm{cat}}) = \sum_{c_j \in S_{\mathrm{cat}}} \max\{v.p_j | v \in r\}.$$

Take the trip route $r = (v_1, v_6, v_{11})$ which starts from node v_1 in Fig.2 as an example. The categories of r are the union of categories of v_1, v_6 and v_{11}, i.e., $r.l_{\mathrm{cat}} = \{C_1, C_2, C_3\}$. We use a default visiting time of one hour for each POI. Thus, the total time cost of r is the sum of the edge weights of $e_{1,6}$ and $e_{6,11}$ plus the visiting time at each node of v_1, v_6 and v_{11}, i.e., $r.t = e_{1,6}.w + e_{6,11}.w + 1 \times 3 = 1 + 1 + 3 = 5$. The popularity of r w.r.t. C_1, $r.p_1$, is 0.08, and the delegate node for C_1 is v_6 since its popularity w.r.t. C_1 is the largest. The delegate nodes for C_2 and C_3 are v_1 and v_{11}, respectively. Given a set of categories $S_{\mathrm{cat}} = \{C_1, C_2\}$, the popularity of r w.r.t. S_{cat} is $pop(r, S_{\mathrm{cat}}) = 0.08 + 0.07 = 0.15$.

Spatial Distance Between Routes. Given two routes r_i and r_j, we compute the spatial distance between them using the minimum distance between their nodes:

$$dist(r_i, r_j) = \min\{dist_{\mathrm{e}}(v_m, v_n) | v_m \in r_i, v_n \in r_j\}.$$

Here, $dist_{\mathrm{e}}(\cdot)$ is a function that returns the Euclidean distance between two POIs.

Query Formulation. A spatial diversified top-k routes (SDkR) query q is represented by a 5-tuple:

$$q = (v_{\mathrm{q}}, t_{\mathrm{q}}, S_{\mathrm{cat}}, k, \sigma).$$

Here, v_{q} is the desired starting point, t_{q} is the travel time budget, which can be set to exclude the visiting time cost of POIs, i.e., the visiting time of each POI is 0, S_{cat} is a set of desired POI categories, k is the desired number of routes, and σ is a desired distance threshold between the routes.

Given an SDkR query q, a feasible route for q is a trip route that starts from v_{q}, consumes travel time less than t_{q}, and covers the categories in S_{cat}. In addition, we require that each node (except v_{q}) contained by a feasible route must be a delegate node of at least one category in S_{cat}. As a result, the length of a feasible route is at most $|S_{\mathrm{cat}}| + 1$ considering the starting point. We enforce this requirement to prune those

routes that contain nodes that do not contribute to the overall popularity of the route.

Definition 1 (Feasible Route). *Given an SDkR query q, a feasible route $r = (v_1, v_2, \ldots, v_m)$ satisfying that* 1) $r.v_1 = v_q$, 2) $r.l_{cat} \supseteq S_{cat}$, 3) $r.t \leqslant t_q$, *and* 4) $v_2, \ldots, v_m$ *are delegate nodes, i.e.,* $\forall v_i \in r$ $(i \neq 1)$, $\exists c_j \in v_i.l_{cat}$ *such that* $v_i.p_j > v'.p_j, \forall v' \in r \setminus \{v_i\}$.

Note that since the starting POI v_q is shared by every feasible route of the same SDkR query, we omit it in the computation of the route popularity and the distance between the routes.

To give an example, suppose that we need to perform an SD3R query q over the POI graph in Fig.2, where

$$q = (v_q, 6 \text{ hours}, \{C_1, C_2, C_3\}, 3, 2 \text{ km}).$$

Two exemplar feasible routes are $r_I = (v_q, v_1, v_6, v_{11})$ and $r_{II} = (v_q, v_2, v_7, v_{12})$. From Table 2, we can see that the delegate node of r_{II} for C_1 is v_2 since the popularity w.r.t. C_1 of v_2 is the largest, i.e., $r_{II}.p_1 = v_2.p_1 = 0.08$. The delegate nodes for C_2 and C_3 are v_7 and v_{12}, respectively, and $r_{II}.p_2 = v_7.p_2 = 0.05$, $r_{II}.p_3 = v_{12}.p_3 = 0.07$. Thus, the popularity of r_{II} w.r.t. $\{C_1, C_2, C_3\}$ is the sum of $v_2.p_1, v_7.p_2$ and $v_{12}.p_3$, i.e., $pop(r_{II}, \{C_1, C_2, C_3\}) = v_2.p_1 + v_7.p_2 + v_{12}.p_3 = 0.20$. The spatial distance between the two routes is computed as $dist(r_I, r_{II}) = dist_e(v_1, v_2) = 2$.

Definition 2 (Spatial Diversified Top-k Routes Query). *An SDkR query $q = (v_q, t_q, S_{cat}, k, \sigma)$ aims to find a size-k set R of feasible routes, such that the spatial distance between any two feasible routes of R is not less than σ, and for any other size-k set R' of feasible routes satisfying the spatial diversity constraint, the total popularity of R w.r.t. S_{cat} is not less than that of R':*

$$\sum_{r_i \in R} pop(r_i, S_{cat}) \geqslant \sum_{r'_i \in R'} pop(r'_i, S_{cat}),$$
$$\forall R'(|R'| = k), \forall r'_i, r'_j \in R' : dist(r'_i, r'_j) \geqslant \sigma,$$
$$\forall r_i, r_j \in R : dist(r_i, r_j) \geqslant \sigma.$$

Note that the SDkR query does not specify the destination POI of the routes returned. However, our algorithms (detailed in Section 4 and Section 5) can be easily adapted to the case where the destination location is specified by modifying the generation process of the feasible routes.

Theorem 1. *The SDkR query is NP-hard.*

Proof. In what follows, we will show that the SDkR problem can be reduced to the maximum independent set (MIS) problem, which aims to find a maximum vertex set in a graph, such that each pair of vertices in this set cannot be connected by an edge. Since the MIS is proven to be NP-hard[39], the SDkR query is also NP-hard.

Given an SDkR query, we use $\mathcal{F}$ to denote a set that consists of all the feasible routes on the POI graph G. In order to reduce the SDkR problem to the MIS problem, we construct a graph G_m. Every feasible route is a node of G_m. An edge between two nodes is added to G_m if their distance (i.e., the distance of two corresponding feasible routes) is smaller than the distance threshold σ. We consider a special case of the SDkR problem, where all the feasible routes in $\mathcal{F}$ have the same popularity (e.g., $pop(r_i, S_{cat}) = 1, \forall r_i \in \mathcal{F}$) and $k = |\mathcal{F}|$. In such a case, the SDkR query on G is equivalent to finding the maximum independent set on G_m, which is proven to be NP-hard. □

4 Two-Stage Algorithm

As mentioned above, the SDkR query is NP-hard. Thus, when the number of trip routes is large, it is costly to process an SDkR query due to the large search space of possible k-route combinations. In this section, we propose the TSS-P algorithm for SDkR query processing. TSS-P consists of two stages. In the first stage, it computes all feasible routes. In the second stage, it computes the optimal k-route combination, i.e., the set of k feasible routes that satisfies the spatial diversity constraint and has the highest overall popularity. We start with computing feasible routes in Subsection 4.1. We discuss the optimal k-route combination finding by TSS-P in Subsection 4.2.

4.1 Computing the Feasible Routes

Given a POI graph G and an SDkR query $q = (v_q, t_q, S_{cat}, k, \sigma)$. We compute the feasible routes using a simple breadth-first traversal on G starting from v_q. In the case where v_q is not a POI in graph G, we first add v_q to G and connect it to each POI using an edge. The weights of these edges are computed on the fly using the method described in Section 3. The newly added edges will be deleted when the query processing is completed. Algorithm 1 summarizes the procedure of computing the feasible routes.

Algorithm 1. Computing Feasible Routes

Input: G: POI graph, v_{q}: query starting node, t_{q}: the travel time budget, S_{cat}: set of query categories
Output: $\mathcal{F}$: feasible route set

```
1  F ← ∅, Q ← ∅, r_i ← v_q
2  Q.enqueue(r_i)
3  while not Q.empty() do
4      r_i ← Q.dequeue()
5      foreach v_j ∈ V \ {r_i} do
6          r_n ← r_i ∪ v_j
7          if r_n.t ⩽ t_q and Ref(r_n) then
8              if r_n.l_cat ⊃ S_cat then
9                  F.push(r_n)
10                 F.update(r_n)
11             if |r_n| < |S_cat| + 1 then
12                 Q.enqueue(r_n)
13                 Q.update(r_n)
14         else
15             Discard r_n
16 Return F
```

At the beginning of the algorithm, a route of only one node v_{q} is inserted into a queue Q, which we use to store the routes to be expanded. Then the traversal starts. Each time, a route r_i is dequeued from Q. For each node v_j except the nodes in r_i, a new route r_{n} is generated by adding v_j to the tail of r_i. The algorithm checks if r_{n} satisfies the query time budget. It also calls a function $Ref(\cdot)$ to check if all nodes contained by r_{n} are delegate nodes. If any of the two conditions is not satisfied, the algorithm discards r_{n}. Otherwise, the algorithm checks if r_{n} covers query category set. If so, it adds r_{n} to the feasible routes set $\mathcal{F}$ (lines 8–10). A function $\mathcal{F}.update(r_{\mathrm{n}})$ is then called to check if there exists any route in $\mathcal{F}$ that has the same set of nodes as r_{n} but with a different visiting order. If so, the algorithm only keeps the one with the minimum travel cost (line 10). If r_{n} is a feasible route, then the algorithm checks if r_{n} can be further expanded (lines 11–13). If the length of r_{n} is less than $|S_{\mathrm{cat}}| + 1$, which means at least one of its nodes is the delegate nodes of more than two categories, r_{n} is again enqueued into Q for further expansion. At the same time, if r_{n} does not cover the categories in S_{cat}, the algorithm also adds it to Q for further route expansion. A function $Q.update(r_{\mathrm{n}})$ is called to update Q. The function checks if there exits any route in Q that passes the same set of nodes as r_{n} and has the same ending node as r_{n}. If so, the algorithm only keeps the partial route with the minimum time cost in Q (line 13). This process repeats until Q becomes empty.

4.2 Finding the Optimal K-Route Combination by TSS-P

We call a set of k feasible routes a candidate solution if it satisfies the spatial diversity constraint, i.e., the distance between any two routes in the set is greater than or equal to σ. In the second stage, our goal is to find the candidate solution with the highest popularity w.r.t. the query categories, i.e., the optimal solution. To simplify the discussion, in what follows, we omit "w.r.t. the query categories" and use $pop(r)$ to represent $pop(r, S_{\mathrm{cat}})$ as long as the context is clear.

A straightforward algorithm to find the optimal solution is to enumerate all size-k subsets of the feasible routes set $\mathcal{F}$, prune those that do not satisfy the spatial diversity constraint, and return the remaining k-route combination with the highest overall popularity. The time complexity of the algorithm is $O(|\mathcal{F}|^k)$, which means the query cost is expensive when $|\mathcal{F}|$ is larger. To improve the query efficiency, we design the TSS-P algorithm to compute the optimal solution without enumerating all size-k subsets.

Feasible Route Graph. We use an auxiliary data structure named the feasible route graph (FRG) to manage the pairs of feasible routes satisfying the spatial diversity constraint. FRG is an undirected graph where each node stores a feasible route r_i and its popularity $pop(r_i)$. An edge between two nodes denotes that the two corresponding routes satisfy the spatial diversity constraint. Building FRG takes a two-layer nested loop over all the feasible routes to identify those satisfying the spatial diversity constraint.

Consider the SD3R query example in Section 3, i.e., $q = (v_{\mathrm{q}}, 6 \text{ hours}, \{C_1, C_2, C_3\}, 3, 2 \text{ km})$. There are six feasible routes:

$$r_0 = \{v_{\mathrm{q}}, v_0, v_5\}, \quad r_1 = \{v_{\mathrm{q}}, v_1, v_6, v_{11}\},$$
$$r_2 = \{v_{\mathrm{q}}, v_1, v_8, v_{13}\}, \quad r_3 = \{v_{\mathrm{q}}, v_2, v_7, v_{12}\},$$
$$r_4 = \{v_{\mathrm{q}}, v_3, v_9\}, \quad r_5 = \{v_{\mathrm{q}}, v_4, v_{10}\}.$$

Based on the spatial distances between them, we construct an FRG as shown in Fig.3. Each of the six nodes in the FRG is labeled by their corresponding feasible routes and their popularities. For example, the node labeled by r_0 corresponds to the feasible route r_0. The popularity of r_0 is 0.26. Since $\sigma = 2$ and the two feasible routes r_4 and r_5 have a distance greater than or equal to σ to r_0, there are two edges to connect r_4 and r_5 with r_0 in the FRG.

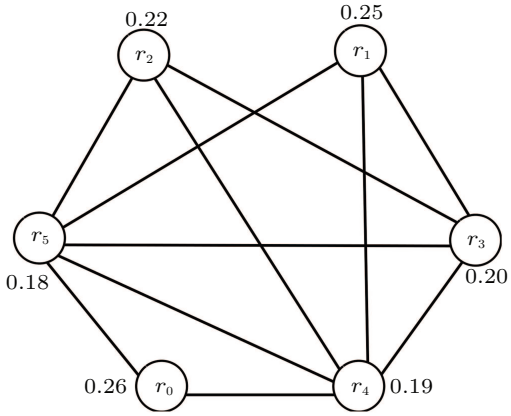


Fig.3. FRG of the SD3R query example in Section 3.

TSS-P Algorithm. Once the FRG is built, finding the optimal solution is equivalent to finding a k-clique in the FRG with the largest popularity. Here, a k-clique is a size-k set of nodes where each pair of nodes is connected with an edge[40]. TSS-P performs a best-first traversal on the FRG to find such a clique. Algorithm 2 summarizes the TSS-P algorithm, where Q_i denotes the last non-empty queue. For simplicity, we still use the function $pop(\cdot)$ to return the sum of the popularities of a set of nodes. The traversal is supported by a stack S and k queues $Q_0, ..., Q_{k-1}$. Stack S stores the nodes in the FRG that form the clique currently under consideration. The queue Q_i ($i \in [0, k-1]$) stores the nodes in the FRG that can be chosen to construct a $(i+1)$-route combination with S. The queues are prioritized by the popularities of the nodes. At start, Q_0 is initialized to contain all the nodes in the FRG, while the other queues are empty (line 4). Then the algorithm dequeues a node r_j from the last non-empty queue Q_i and adds it to S. Then we consider two cases.

1) If there are k nodes in S, the routes corresponding to these k nodes form a candidate solution. Further, if this candidate solution has a higher popularity than the current optimal solution, denoted by R, we update R to this candidate solution (lines 14–18).

2) If there are less than k nodes in S, we further add the set of nodes, which is the intersection of Q_i and the adjacent nodes of node r_j, into the next queue Q_{i+1} (lines 20 and 21).

We repeat this process until all queues become empty. Then we return R. To further enhance the efficiency, when starting a new iteration we use the following two heuristics to early prune some nodes from consideration. Let $|S|$ be the number of nodes in S including the node r_j currently under consideration. Let adj be the number of nodes in $Q_{|S|}$.

Algorithm 2. TSS-P

Input: G: POI graph, v_q: query starting node, t_q: the travel time budget, S_{cat}: set of query categories
Output: R: optimal solution

```
1  F, R, S, Q_0, Q_1, ..., Q_{k-1} ← ∅
2  F ← Compute Feasible Routes (Algorithm 1)
3  A two-layer nested loop over F to create the FRG
4  i ← 0, Q_0 ← nodes in FRG
5  while |Q_i| > 0 do
6      if |Q_i| < k − |S| or pop(S ∪ top-(k − |S|) nodes in
       Q_i) < pop(R) then
7          Q_i.clear(), S.pop()
8          i ← i − 1
9          Continue
10     r_j ← Q_i.dequeue()
11     if the number of adjacent nodes of r_j < k − 1 then
12         Continue
13     S.push(r_j)
14     if |S| = k then
15         if pop(R) < pop(S) then
16             R ← S
17         S.pop(), Q_i.clear
18         i ← i − 1
19     else
20         Q_{i+1} ← intersection of Q_i and adjacent nodes of
           r_j
21         i ← i + 1
22 Return R
```

Pruning Heuristic 1. If $adj < k - |S|$, we can safely prune the nodes in $Q_{|S|}$. This is because we need $k-|S|$ more nodes to form a candidate solution. Since fewer than $k-|S|$ nodes remain connected to all the nodes in S, it is impossible to obtain a candidate solution based on the current S (line 6).

Pruning Heuristic 2. If $adj \geqslant k - |S|$, let subset $\mathcal{U}$ be the $k - |S|$ nodes in $Q_{|S|}$ with the largest popularities. If the sum of the popularities of the nodes in $S \cup \mathcal{U}$ is smaller than that of the nodes in R, we can safely prune all the nodes in $Q_{|S|}$. This is because, if the most popular adjacent nodes cannot form a better candidate solution with S, no adjacent node needs to be considered any further (line 6).

Fig.4 illustrates the processing of the above SD3R query example using TSS-P. We first initialize Q_0 to contain all the nodes in Fig.3, prioritized by the popularity. Then r_0 is dequeued and added to S. Now there are fewer than three nodes in S. We add the adjacent nodes of r_0, namely r_4 and r_5, to Q_1. Then r_4 is dequeued and added to S, and Q_2 is updated to contain the adjacent nodes of r_4 that are also adjacent to r_0.

There is only one such node r_5 in Q_2. Adding r_5 to S forms the first candidate solution $\{r_0, r_4, r_5\}$. We update R to this set. We then pop out r_5 and r_4 from S. Now the sizes of set S and Q_1 are both 1. According to pruning heuristic 1, no further processing on r_0 is needed since only one node in Q_1 (i.e., r_5) is connected to r_0, which is fewer than $k - |S| = 2$ nodes. The next node r_1 in Q_0 is then dequeued and added to S. Repeating the above procedure, we find another candidate solution $\{r_1, r_3, r_4\}$. Since the popularity of the new candidate solution (i.e., 0.64) is higher than that of the previous R (i.e., 0.63), we update R to this new candidate solution. After processing r_1, the next three nodes in Q_0 are r_2, r_3, and r_4. The sum of the popularities of these three nodes (i.e., 0.61) is less than that of R. According to pruning heuristic 2, we can terminate the algorithm.

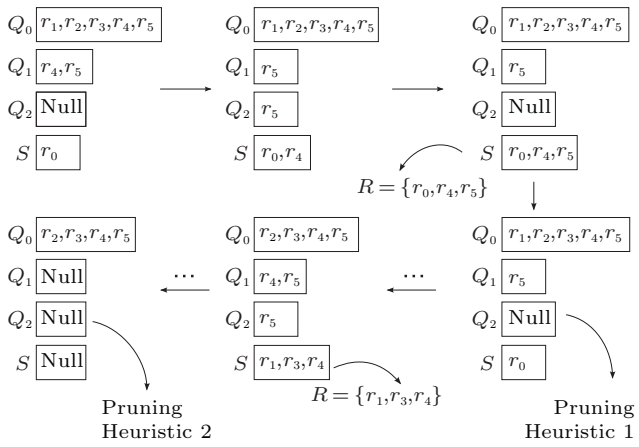


Fig.4. Processing of the SD3R query example using TSS-P.

Cost Analysis. The time complexity of TSS-P consists of two parts: 1) the time for computing the feasible routes set $\mathcal{F}$ and 2) the time for finding the optimal solution. Given the time budget t_q, the number of edges in a feasible route is at most $n_e = \min\{\lfloor t_q/w_{\min} \rfloor, |S_{cat}|\}$, where $w_{\min}$ is the minimum weight of all edges in G. Thus, the time complexity of the first stage is $O((|V|^{n_e} - 1)/(|V| - 1))$. The time cost of the second stage is the sum of the time cost for creating the FRG and the traversal time over the FRG. To build the FRG, a two-layer nested loop is performed, which takes $O(|\mathcal{F}|^2)$ time. Assuming that the maximum out-degree of the nodes in FRG is $d_{\max}$, the traversal over FRG takes $O(|\mathcal{F}| \times d_{\max}^{k-1})$ time. In the worst case where each pair of nodes in FRG is connected, the time complexity of TSS-P will become $O(|\mathcal{F}|^k)$. However, as shown in the experiments, the FRG can usually help prune combinations of nearby feasible routes, and the two pruning heuristics further reduce the number of k-route combinations considered, which leads TSS-P to perform well.

In terms of the space complexity, TSS-P needs to store the POI graph which costs $O(|V| + |E|)$ space. The space cost of FRG and the memory needed for query processing are query-dependent. Assuming that the maximum out-degree of the nodes in FRG is $d_{\max}$, and the number of desired routes is k. The overall space complexity of TSS-P is $O(|V|+|E|+|\mathcal{F}|+|\mathcal{F}| \times d_{\max} + k \times d_{\max})$, where $|\mathcal{F}|+|\mathcal{F}| \times d_{\max}$ is the space cost of the FRG and $k \times d_{\max}$ is the space cost of the queues used for the traversal over FRG. Note that the cardinality of $\mathcal{F}$ is bounded by $O(\binom{|V|}{n_e})$.

5 Single-Stage Algorithms

TSS-P has to generate all feasible routes before finding the optimal solution. It is costly to generate all routes that satisfy the query constraints when the number of POIs or the travel time budget is large. This section presents two algorithms that generate feasible routes and find the optimal solution at the same time (i.e., in a single stage), so as to avoid generating the unnecessary feasible routes. The two single-stage algorithms are based on a concept named the partial solution set (PSS). We introduce this concept in Subsection 5.1. We then describe the two algorithms in Subsection 5.2 and Subsection 5.3, respectively. These two algorithms share the same overall procedure, where one produces precise answers and the other produces α-approximate answers.

5.1 Partial Solution Set

A partial solution is a set of less than k feasible routes where any two of the routes satisfy the spatial diversity constraint. A PSS is a set where each element is a partial solution. A PSS has $k - 1$ queues denoted by $\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_{k-1}$, where $\mathcal{P}_i$ stores partial solutions of size-i prioritized by their popularity values. We use $\mathcal{P}_{i,j}$ to denote the j-th partial solution in $\mathcal{P}_i$.

The single-stage algorithms also use Algorithm 1 to generate feasible routes. When a new feasible route r is generated, if r and a size-i partial solution $\mathcal{P}_{i,j}$ $(i < k - 1)$ satisfy the spatial diversity constraint, we insert $\mathcal{P}_{i,j} \cup \{r\}$ into $\mathcal{P}_{i+1}$. Meanwhile, r is inserted into $\mathcal{P}_1$ as a size-1 partial solution. If r and a size-$(k-1)$ partial solution $\mathcal{P}_{k-1,j}$ satisfy the spatial diversity constraint, then they form a candidate solution. The can-

didate solution R with the highest popularity will be returned if no more feasible routes can be found.

Consider again the above SD3R query example in Fig.2. Table 3 shows how the queues of PSS update as new feasible routes are generated for the query.

Table 3. Updating PSS for the SD3R Query Example

Action	Status of PSS
Inserting r_0	$\mathcal{P}_1$: $\{r_0\}$
	$\mathcal{P}_2$: Null
Inserting r_1	$\mathcal{P}_1$: $\{r_0\}$ $\{r_1\}$
	$\mathcal{P}_2$: Null
Inserting r_3	$\mathcal{P}_1$: $\{r_0\}$ $\{r_1\}$ $\{r_3\}$
	$\mathcal{P}_2$: $\{r_1, r_3\}$
Inserting r_4	$\mathcal{P}_1$: $\{r_0\}$ $\{r_1\}$ $\{r_3\}$ $\{r_4\}$
	$\mathcal{P}_2$: $\{r_1, r_3\}$ $\{r_0, r_4\}$ $\{r_1, r_4\}$ $\{r_3, r_4\}$

Suppose that we have sequentially generated the routes r_0, r_1, r_3, and r_4. When the first route r_0 is generated, we simply insert it into $\mathcal{P}_1$. When r_1 is generated, we first insert it into $\mathcal{P}_1$. Since r_0 and r_1 do not satisfy the spatial diversity constraint, the set $\{r_0, r_1\}$ is discarded. When r_3 is generated, we also insert it into $\mathcal{P}_1$. Since the distance between r_1 and r_3 is greater than σ, we insert $\{r_1, r_3\}$ into $\mathcal{P}_2$. Next, we process route r_4, which produces the first candidate solution $\{r_1, r_3, r_4\}$.

5.2 Single-Stage Incremental Search Algorithm

Our first single-stage algorithm 3S-I follows a procedure similar to Algorithm 1. The algorithm performs a best-first traversal on the POI graph starting from query node v_q to form feasible routes satisfying the query constraints. Recall that a queue Q is used to store the partial routes found so far. The partial routes in Q are prioritized by their popularity upper bounds. We will detail the computation of this bound in the following paragraph. The partial routes grow longer as more nodes in the POI graph are visited. When a partial route becomes a feasible route, instead of adding it into the feasible routes set $\mathcal{F}$, 3S-I calls a function named *UpdatePSS* to add this feasible route to the PSS and compute new candidate solutions from it. We will detail this function later in this subsection. When Q or PSS becomes empty, 3S-I returns the optimal solution and terminates. The pseudo code of 3S-I is similar to that of Algorithm 1, except that lines 9 and 10 are replaced by a call of *UpdatePSS*, and that the optimal solution R is returned instead of $\mathcal{F}$. We omit the pseudo code to keep the discussion concise.

Popularity Upper Bound of a Partial Route. We compute an upper bound of the popularity that a partial route may reach when it becomes a feasible route. This bound guides the traversal on the POI graph to expand the more promising partial routes first, and helps to prune unpromising partial solutions.

Definition 3 (Popularity Upper Bound of a Partial Route). *Given a partial route r_p and its last node v_m, let $r_\mathrm{p}.t$ be the time cost of r_p, S_cat the query categories, and t_q the travel time budget. Let $V(v_\mathrm{m}, t_\mathrm{q} - r_\mathrm{p}.t)$ be the set of nodes that can be reached from v_m within $t_\mathrm{q} - r_\mathrm{p}.t$ time. The popularity upper bound of r_p w.r.t. S_cat, denoted by $pop^\tau(r_\mathrm{p})$, is the sum of the maximum popularities of the nodes in $r_\mathrm{p} \cup V(v_\mathrm{m}, t_\mathrm{q} - r_\mathrm{p}.t)$ w.r.t. each category in S_cat:*

$$pop^\tau(r_\mathrm{p}) = \sum_{c_i \in S_\mathrm{cat}} \max\{v.p_i | v \in r_\mathrm{p} \cup V(v_\mathrm{m}, t_\mathrm{q} - r_\mathrm{p}.t)\}.$$

By Definition 3, the popularity of any feasible route grown from the partial route r_p does not exceed $pop^\tau(r_\mathrm{p})$. However, to compute $pop^\tau(r_\mathrm{p})$ at query time needs to compare every node in $V(v_\mathrm{m}, t_\mathrm{q} - r_\mathrm{p}.t)$ with the nodes in r_p on the popularity value w.r.t. each category in S_cat. In order to compute $pop^\tau(r_\mathrm{p})$ efficiently, we predefine ρ different time budget values $t_0, t_1, ..., t_{\rho-1}$. We then precompute for every node v in the POI graph a matrix of $n_\mathrm{c} \times \rho$ popularity upper bound values offline where n_c is the number of POI categories. Fig.5 illustrates the structure of a pre-computed node. Every node v is represented as:

$$v = (pid, lon, lat, t, l_\mathrm{p}, l_{t_0}, \ldots, l_{t_{\rho-1}}).$$

Here, *pid* is the ID of v, *lon* and *lat* are the coordinates of v, t is the average visiting time that users spend at v, l_p is the list of popularity values of v w.r.t. its different categories, and l_{t_i} $(i \in [0, \rho - 1])$ is the list of maximum popularities of the nodes in $V(v, t_i)$ w.r.t. each category, where $V(v, t_i)$ is the set of nodes that v can reach within t_i time. Each element in the list, denoted by $v.p_{t_i - c_j}$, is the maximum popularity of the nodes in $V(v, t_i)$ w.r.t. category c_j, i.e., $v.p_{t_i - c_j} = \max\{v_l.p_j | v_l \in V(v, t_i)\}$. For example, $v.p_{t_0 - c_1}$ is the maximum popularity of the nodes in $V(v, t_0)$ w.r.t. C_1. At query time, when a node v is visited, its matrix of popularity upper bound values is used to compute the popularity upper bound of the partial route. Particularly, the smallest predefined time

budget that is larger than or equal to $t_q - r_p.t$ and the matrix elements corresponding to the query POI categories are used in the computation.

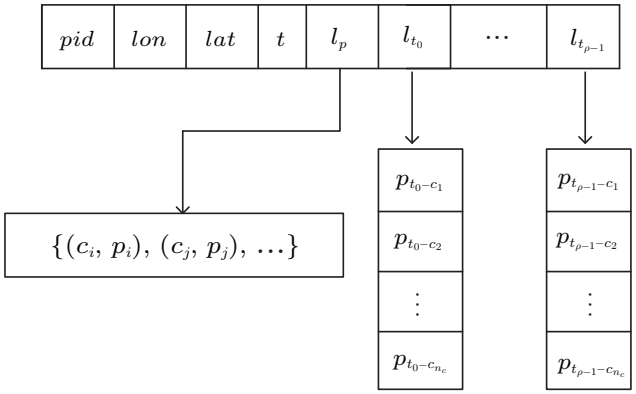


Fig.5. Node structure after pre-computation.

We use the above SD3R query example in Fig.2, to illustrate how the popularity upper bound of a partial route is computed. Recall that in the query, $t_q = 6$, $S_{cat} = \{C_1, C_2, C_3\}$. Let $r_p = \{v_q, v_1, v_6\}$ be a partial route under consideration. We first compute the current popularities of the partial route w.r.t. each query category. We have $r_p.p_1 = 0.08, r_p.p_2 = 0.07$, and $r_p.p_3 = 0$. Let the time cost $r_p.t$ be 4, then $t_q - r_p.t = 2$. Suppose that there are three predefined time budget values 2, 5 and 9. We fetch the popularity matrix elements corresponding to $t = 2$ to compute popularity upper bound. We assume that the set of nodes $V(v_6, 2)$ that v_6 can reach within two hours is $\{v_1, v_5, v_6, v_{11}\}$. From Table 2, v_6 has the maximum popularity w.r.t. C_1 among the four reachable nodes. Thus, the maximum popularity w.r.t. C_1 that v_6 may achieve within two hours is 0.08, i.e., $v_6.p_{2-c_1} = 0.08$. Similarly, v_5 has the maximum popularity w.r.t. C_2 among the four reachable nodes. Thus, the maximum popularity w.r.t. C_2 that v_6 may achieve within two hours is 0.09, i.e., $v_6.p_{2-c_2} = 0.09$. Node v_{11} has the maximum popularity w.r.t. C_3 among the four reachable nodes. Thus, the maximum popularity w.r.t. C_3 that v_6 may achieve within 2 hours is 0.10, i.e., $v_6.p_{2-c_3} = 0.10$. Combining with the current popularities of r_p w.r.t. each query category, the popularity upper bound values of r_p w.r.t. C_1, C_2, and C_3 are 0.08, 0.09, and 0.10, respectively. Thus, the popularity upper bound of r_p is $0.08 + 0.09 + 0.10 = 0.27$.

Updating the PSS. Based on the popularity upper bounds of the partial routes, we present the following pruning heuristic to prune unpromising partial solutions which cannot form any candidate solution with a higher popularity than the current optimal solution R.

Pruning Heuristic 3. Given a partial solution $\mathcal{P}_{i,j}$, the current optimal solution R, and the first $k - i$ partial routes in queue Q denoted by $r_p^0, r_p^1, ..., r_p^{k-i-1}$, if the popularity of $\mathcal{P}_{i,j}$, denoted by $pop(\mathcal{P}_{i,j})$, satisfies $pop(\mathcal{P}_{i,j}) + \sum_{l=0}^{l=k-i-1} pop^{\tau}(r_p^l) \leqslant pop(R)$, it is guaranteed that $\mathcal{P}_{i,j}$ cannot form any candidate solution with a higher popularity than R. Thus, $\mathcal{P}_{i,j}$ can be pruned. After the pruning process, if the PSS is empty, we can stop generating new feasible routes and return R as the query answer.

The correctness of the heuristic is straightforward. Partial solution $\mathcal{P}_{i,j}$ contains i feasible routes and needs $k-i$ more routes to form a candidate solution. The first $k - i$ routes in Q are those with the highest popularity upper bounds. If the sum of the popularities of $\mathcal{P}_{i,j}$ and these $k-i$ routes do not exceed $pop(R)$, it is impossible for $P_{i,j}$ to extend to be a candidate solution better than R.

Similarly, when a new feasible route r is generated, before combining it with an existing partial solution $\mathcal{P}_{i,j}$, we test whether $pop(\mathcal{P}_{i,j}) + pop(r) + \sum_{l=0}^{l=k-i-2} pop^{\tau}(r_p^l) \leqslant pop(R)$ holds. If so, we do not combine them as $\mathcal{P}_{i,j} \cup \{r\}$ is unpromising.

The function $UpdatePSS$ uses this pruning heuristic to update the PSS as new feasible routes are generated. As summarized in Algorithm 3, when a new feasible route r is generated, we first check if there exists a route r' in $\mathcal{P}_1$ that has the same set of nodes as r and $r'.t > r.t$ (lines 1 and 2). If so, we replace r' with r in every partial solution that contains r'. Otherwise, we discard the route r. If we do not find any route with the same set of nodes as r, we then test if r satisfies the spatial diversity constraint with a partial solution $\mathcal{P}_{k-1,j}$ in $\mathcal{P}_{k-1}$ and they form a new candidate solution with a higher popularity than R found so far. If so, we update R with $\mathcal{P}_{k-1,j} \cup \{r\}$ (lines 7–11). Then, we update the rest of the partial solution queues $\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_{k-2}$ by trying to combine r with the existing partial solutions where they satisfy the spatial diversity constraint. During this process, pruning heuristic 3 is applied to avoid generating new unpromising partial solutions, and also to remove existing unpromising partial solutions since R may have been updated (lines 12–17).

We continue with the example in Subsection 5.1 to explain $UpdatePSS$ (i.e., continue to extend Table 3). We assume that after inserting r_4, the two highest popularity upper bounds of the current partial routes under consideration are 0.23 and 0.19. According to

pruning heuristic 3, we do not need to insert $\{r_3, r_4\}$ into PSS since its popularity (i.e., 0.39) plus the popularity upper bound (i.e., 0.23) is less than the popularity of the current result set $R = \{r_1, r_3, r_4\}$ (i.e., 0.64). As shown in the first row of Table 4, $\{r_3, r_4\}$ is pruned. Similarly, $\{r_3\}$ and $\{r_4\}$ are also pruned. When a new route r_2 is generated, the popularity upper bound is updated to 0.19. According to pruning heuristic 3, all the current partial solutions in PSS can be pruned. Now that PSS is empty, we can safely terminate the algorithm.

Algorithm 3. UpdatePSS

Input: r: a new feasible route, $r_{\mathrm{p}}^0, \ldots, r_{\mathrm{p}}^{k-2}$: the first $(k-1)$ partial routes, $\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_{k-1}$: queues of the partial solution set, R: the current optimal solution, σ: distance threshold

Output: updated $\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_{k-1}$, R

```
if a route r' in P_1 has the same set of nodes as r then
    if r.t < r'.t then
        Change r' to r in PSS
    else
        Discard r
else
    for j = 1 to |P_{k-1}| do
        if dist(r, P_{k-1,j}) ⩾ σ then
            if pop(P_{k-1,j}) + pop(r) > pop(R) then
                R ← P_{k-1,j} ∪ {r}
            Break
    for i = k − 2 to 1 do
        for j = 1 to |P_i| do
            if pop(P_{i,j}) + pop(r) + Σ_{l=0}^{l=k−i−2} pop^τ(r_p^l) > pop(R) and dist(r, P_{i,j}) ⩾ σ then
                P_{i+1}.enqueue(P_{i,j} ∪ {r})
            if pop(P_{i,j}) + Σ_{l=0}^{l=k−i−1} pop^τ(r_p^l) ⩽ pop(R) then
                P_i.dequeue(P_{i,j})
```

Table 4. Extending Table 3

Action	Status of PSS
Inserting r_4	$\mathcal{P}_1$: $\{r_0\}$ $\{r_1\}$ ~~$\{r_3\}$~~ ~~$\{r_4\}$~~
	$\mathcal{P}_2$: $\{r_1, r_3\}$ $\{r_0, r_4\}$ $\{r_1, r_4\}$ ~~$\{r_3, r_4\}$~~
Inserting r_2	$\mathcal{P}_1$: ~~$\{r_0\}$~~ ~~$\{r_1\}$~~ ~~$\{r_2\}$~~ ~~$\{r_3\}$~~ ~~$\{r_4\}$~~
	$\mathcal{P}_2$: ~~$\{r_1, r_3\}$~~ ~~$\{r_0, r_4\}$~~ ~~$\{r_1, r_4\}$~~ ~~$\{r_2, r_3\}$~~ ~~$\{r_2, r_4\}$~~

Cost Analysis. Suppose that we have generated n_{f} feasible routes so far. When a new feasible route is generated, we first check if it can form a candidate solution with any partial solution in $\mathcal{P}_{k-1}$. The time complexity of this process is $O(\log \binom{n_{\mathrm{f}}}{k-1})$. Next, we check whether the new feasible route can form new partial solutions with existing partial solutions in $\mathcal{P}_1$, $\mathcal{P}_2$, ..., $\mathcal{P}_{k-2}$. The time complexity of this process is bounded by $O((k-2) \times \binom{n_{\mathrm{f}}}{k-2})$. Thus, the overall time complexity of 3S-I is bounded by $O((k-2) \times \binom{n_{\mathrm{f}}}{k-2})$. Usually, $n_{\mathrm{f}} \ll |\mathcal{F}|$ since we use pruning heuristic 3 to prune those unpromising feasible routes and partial solutions. As a result, 3S-I is more efficient than TSS-P. In terms of space complexity, storing the POI graph takes $O(|V| + |E|)$ space. Besides, 3S-I needs to maintain a PSS, which needs $O(\binom{|\mathcal{F}|}{k-1})$ space to store the partial solutions in the worst case. Thus, the space complexity of 3S-I is $O(|V| + |E| + |\mathcal{F}|^{k-1})$.

5.3 Single-Stage α-Approximate Search Algorithm

To further improve the query efficiency, based on 3S-I, we propose an approximate algorithm 3S-α, which uses a popularity upper bound and an approximation ratio α to help early terminate the search for the optimal query answer. 3S-α uses the PSS and the partial routes to compute the popularity upper bound for the optimal solution. For the current first k partial routes $r_{\mathrm{p}}^0, ..., r_{\mathrm{p}}^{k-1}$, the sum of their popularity upper bounds is $\sum_{l=0}^{l=k-1} pop^{\tau}(r_{\mathrm{p}}^l)$. According to the definition of PSS, partial solution $\mathcal{P}_{i,1}$ ($i \in [1, k-1]$) has the maximum popularity in $\mathcal{P}_i$ which stores the partial solutions of size-i. Then, $pop(\mathcal{P}_{i,1}) + \sum_{l=0}^{l=k-i-1} pop^{\tau}(r_{\mathrm{p}}^l)$ is the popularity upper bound of the candidate solutions extended from the partial solutions in $\mathcal{P}_i$. Therefore, the maximum among the k values is the popularity upper bound of the optimal solution, i.e.,

$$pop_{\mathrm{opt}}^{\tau} = \max\{pop(\mathcal{P}_{i,1}) + \sum_{l=0}^{l=k-i-1} pop^{\tau}(r_{\mathrm{p}}^l) | \; i \in [1, k-1], \sum_{l=0}^{l=k-1} pop^{\tau}(r_{\mathrm{p}}^l)\}.$$

Let R_{c} be the current optimal solution and R the optimal solution. Since $pop_{\mathrm{opt}}^{\tau}$ is the popularity upper bound of R, we have $pop_{\mathrm{opt}}^{\tau} \geqslant pop(R)$. Thus, when the popularity of R_{c} is not less than $\alpha \times pop_{\mathrm{opt}}^{\tau}$ ($0 < \alpha \leqslant 1$), i.e., $pop(R_{\mathrm{c}}) \geqslant \alpha \times pop_{\mathrm{opt}}^{\tau}$, we have $pop(R_{\mathrm{c}}) \geqslant \alpha \times pop(R)$, which means R_{c} is an α-approximation solution.

Now we use 3S-α to process the above SD3R query example. Suppose that we have found routes r_0, r_1, r_3

and r_4, and the current PSS is shown in the first row in Table 4. The current popularity upper bounds of the first three partial routes under consideration are 0.23, 0.19 and 0.18, respectively. Thus, the popularity upper bound of $\mathcal{P}_1$ is computed as $pop^\tau(\mathcal{P}_1) = pop(\mathcal{P}_{1,1}) + 0.23 + 0.19 = 0.68$. Similarly, the popularity upper bound of $\mathcal{P}_2$ is computed as $pop^\tau(\mathcal{P}_2) = pop(\mathcal{P}_{2,1}) + 0.23 = 0.68$. The sum of the popularity upper bounds of the first three partial routes is 0.60. Then the overall popularity upper bound $pop^\tau_{\text{opt}} = 0.68$. Let α be 0.9. The current optimal solution $R = \{r_1, r_3, r_4\}$ is returned as the final result since its popularity (i.e., 0.64) is larger than $\alpha \times pop^\tau_{\text{opt}}$ (i.e., 0.612). Compared with 3S-I, 3S-α does not generate route r_2 and thus is more efficient.

6 Experiments

We evaluate the effectiveness and efficiency of our algorithms (i.e., TSS-P, 3S-I and 3S-α) empirically in this section. The experiments are conducted on a desktop computer with a 3.40 GHz Intel Core i7-2600 CPU, 8 GB memory, and 64-bit Windows operating system. All algorithms are implemented in C++.

6.1 Settings

We use the check-in datasets in San Francisco and New York[41], denoted as SF and NY, respectively. We preprocess the check-in records and filter out the POIs that have no check-ins. The result numbers of POIs extracted from these two datasets are shown in Table 5, where we show the total number of POIs and the number of POIs for each of the eight predefined categories. We also visualize the geographical distributions of the POIs in different categories in Fig.6 and Fig.7.

We build the POI graph from the POIs extracted by the procedure described in Section 3. We set the visiting time at each POI as one and a half hours for simplicity since this is not the focus of our work. The travel time between each pair of POIs is estimated using Google map API assuming driving as the means of transportation.

In each set of experiments, we randomly choose 20 locations as the source location v_q and report the average algorithm performance over 20 queries.

In the initialization of the algorithms, we perform online filtering of the irrelevant POIs and only keep the POIs that are of at least one query category. To test the impact of different parameters, we vary the spatial

Table 5. Number of POIs in the Datasets

Dataset	Total	Art	Shop	Food	Nightlife	Travel	Park	College	Building
SF	3 358	179	524	1 739	501	224	189	42	392
NY	8 415	458	1 744	3 227	800	549	584	188	1 447

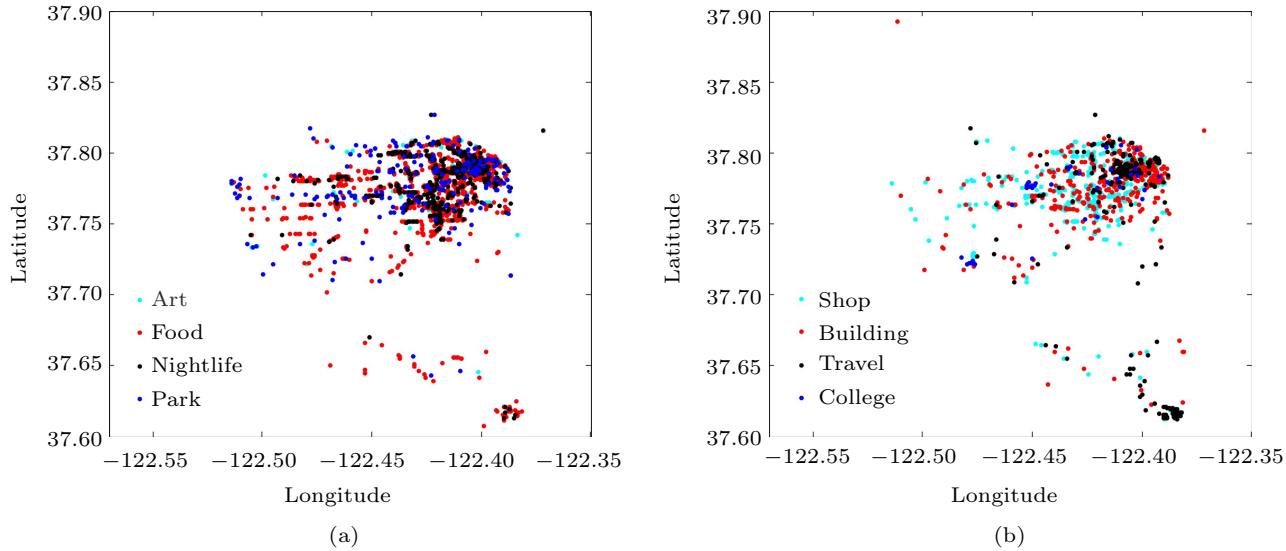


Fig. 6. Distribution of POIs in San Francisco. (a) Categories of "art", "food", "nightlife" and "park". (b) Categories of "shop", "building", "travel" and "college".

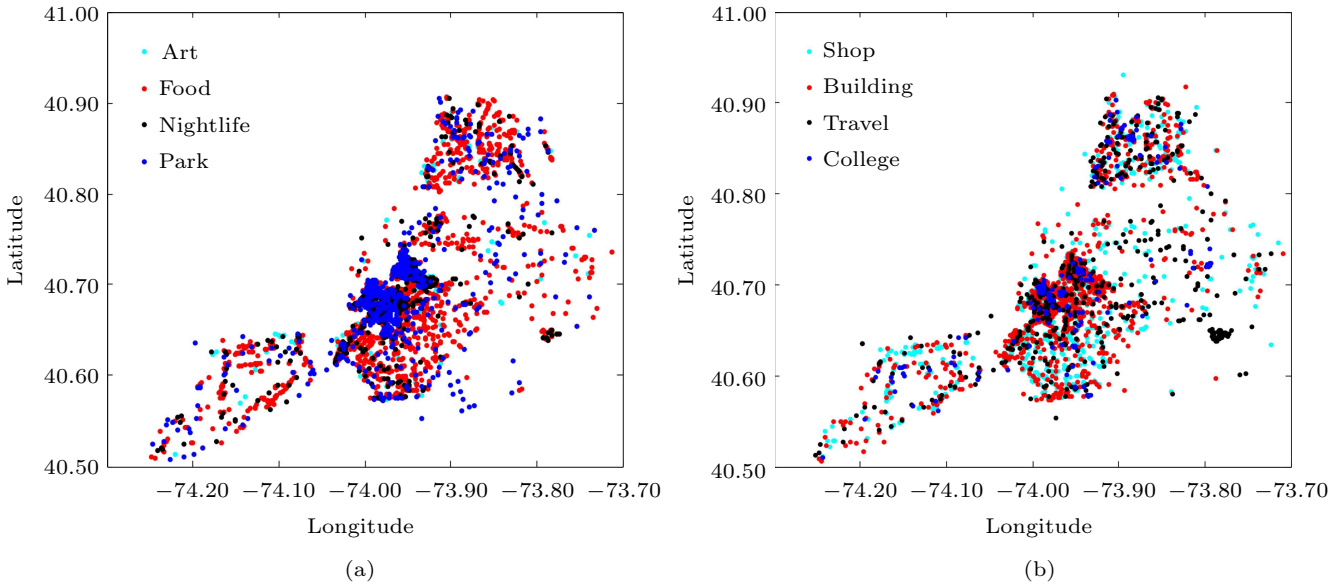


Fig.7. Distribution of POIs in New York. (a) Categories of "art", "food", "nightlife" and "park". (b) Categories of "shop", "building", "travel" and "college".

diversity constraint σ from 1 km to 16 km, the time budget t_q from 5 hours to 9 hours, the cardinality of query categories $|S_{cat}|$ from 1 to 5, the desired number of routes k from 2 to 9, and the approximation ratio α from 0.6 to 1. We precompute popularity values of each POI under the time budget values 2, 5, and 9 in single-stage algorithms, respectively. Table 6 summarizes the parameter values. We report the algorithm running time and the precision of the results produced by the approximate algorithm.

Table 6. Experimental Parameters

Parameter	Default	Value
σ	4.0	1, 2, 4, 8, 16
t_q	7.0	5, 6, 7, 8, 9
k	3.0	2, 3, 5, 7, 9
$\|S_{cat}\|$	3.0	1, 2, 3, 4, 5
α	0.8	0.6, 0.7, 0.8, 0.9, 1.0

6.2 Results

Effect of σ. Fig.8 shows the effect of spatial diversity constraint where the value of σ is varied from 1 to 16. As Fig.8(a) shows, the running time of the three algorithms decreases when σ increases, because fewer pairs of feasible routes satisfy the spatial diversity constraint. From Fig.8(b) we can see that as σ increases, the single-stage algorithms need to generate more feasible routes. The reason is that when σ is larger, it is more difficult to find k routes that satisfy the spatial diversity constraint, and hence more routes need to be generated. A similar result is observed in the NY dataset (Fig.A1). As the results are similar to those for the SF dataset, we move the experimental results for the NY dataset to Appendix.

Effect of α. Fig.9(a) shows the effect of α on 3S-α. As the value of α increases, the running time of 3S-α increases. This is expected as α increases, and it needs to examine more feasible routes to find a candidate solution with a popularity value closer to the optimal solution. As 3S-I is a precise algorithm, the running time remains the same when α increases. Meanwhile, from Fig.9(b) we can see the precision of the results exceeds the required approximation ratio consistently. This confirms the effectiveness of 3S-α.

Effect of t_q. Next, we vary the query time budget t_q from 5 hours to 9 hours. The results in Fig.10(a) show that, as t_q increases, the running time of the three algorithms increases. This is expected that a larger t_q leads to more feasible routes. The single-stage algorithms outperform the two-stage algorithm as they do not need to find all the possible k-route combinations. We also evaluate the effect of t_q on single-stage algorithms when different sets of time budget values are used in the precomputation of popularity upper bounds. In addition to the default set of three budget values 2, 5, and 9 (denoted by $\rho = 3$), we use another set of budget values 3 and 9 (denoted by $\rho = 2$). From Fig.10(b), we see that both 3S-I and 3S-α perform well when $\rho = 3$. This is

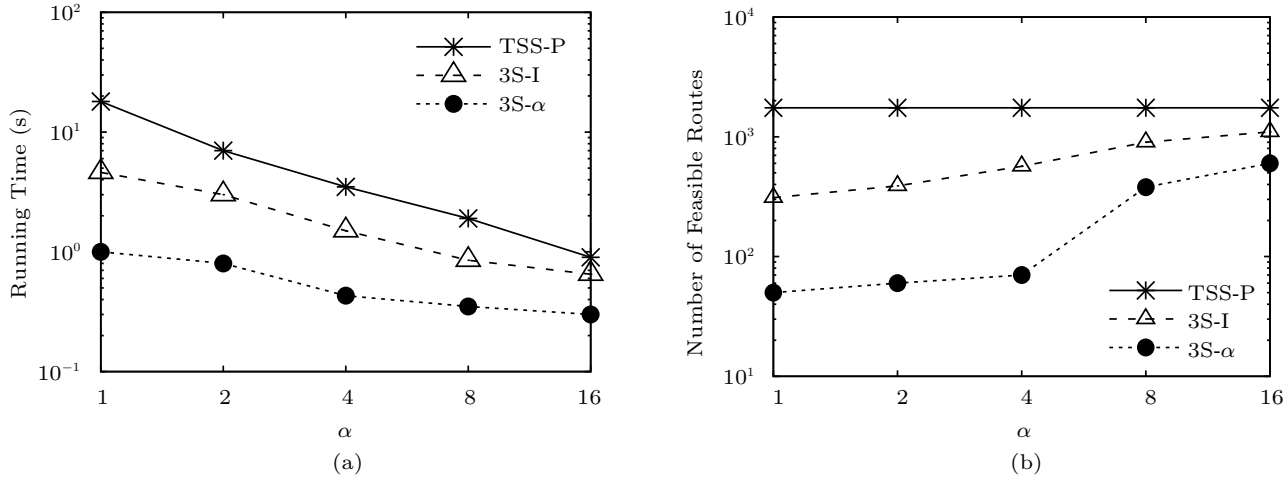


Fig.8. Effect of σ in SF. (a) Running time. (b) Number of feasible routes.

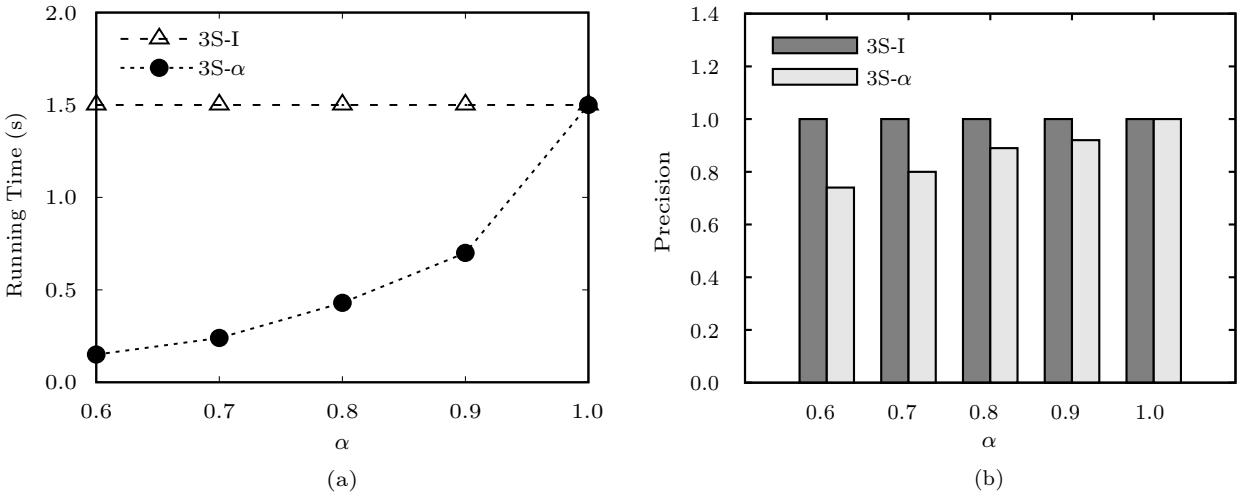


Fig.9. Effect of α in SF. (a) Running time. (b) Precision.

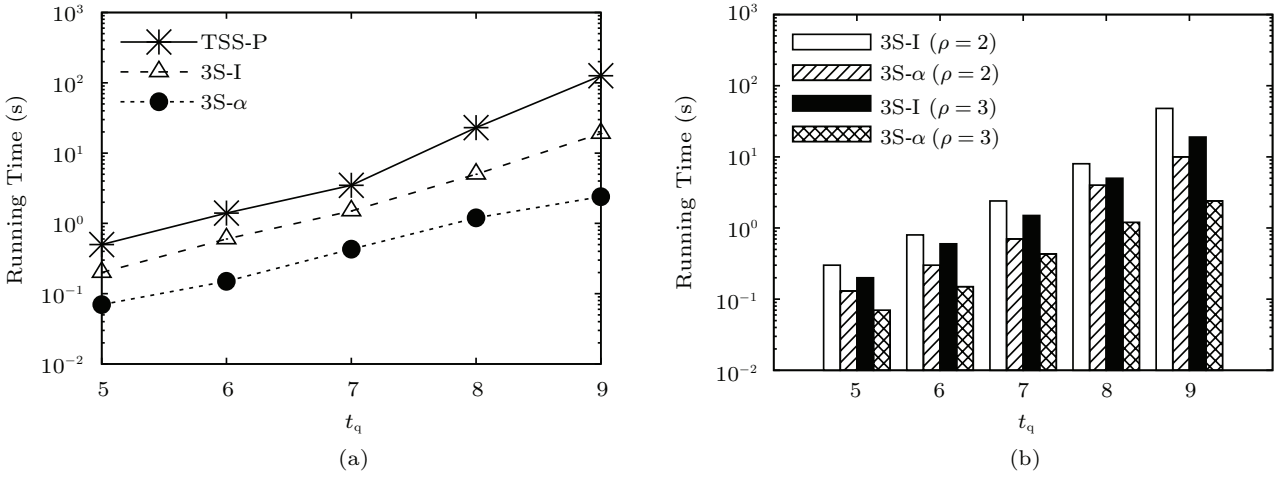


Fig.10. Effect of t_q in SF. (a) Running time. (b) ρ.

because when we use more time budget values in the precomputation, we have a higher probability to obtain a popularity upper bound that is closer to the actual popularity, which increases the pruning efficiency.

Effect of $|S_{\text{cat}}|$. Next we vary the number of POI categories queried, $|S_{\text{cat}}|$, from 1 to 5. Fig.11 shows the result. We find that the running time of the three algorithms increases when $|S_{\text{cat}}|$ increases and then drops after $|S_{\text{cat}}|$ reaches 4. This is because when $|S_{\text{cat}}|$ is large enough (i.e., $|S_{\text{cat}}| > 4$), the number of feasible routes is much less and thus the algorithms need much less time to find the results.

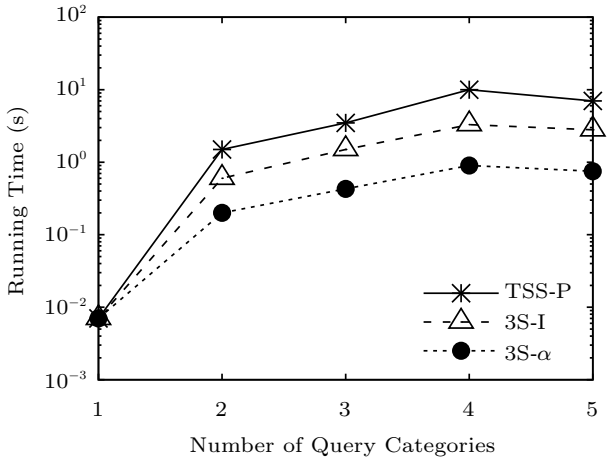


Fig.11. Effect of $|S_{\text{cat}}|$ in SF.

Effect of k. Next we vary the value of k from 2 to 9. Fig.12 shows the running time of the three algorithms. We find that the time cost of TSS-P increases dramatically when k gets larger. When k exceeds 9, the running time of TSS-P is beyond 100 seconds. The two single-stage algorithms generate feasible routes and find candidate solutions at the same time, so as to avoid generating the unnecessary feasible routes. Although the two single-stage algorithms need more time to maintain the partial solution set as k increases, they always outperform TSS-P.

Memory Cost of the POI Graphs. We compare the space overhead in terms of the memory costs of the POI graphs, i.e., $G_{\rho=0}$, $G_{\rho=2}$ and $G_{\rho=3}$, used in the single-stage algorithms. Here, $G_{\rho=0}$ represents a POI graph where every node does not have a matrix of popularity upper bound, which is used by TSS-P. $G_{\rho=3}$ represents a POI graph where every node has a matrix of 8×3 popularity upper bound. This matrix is constructed by the popularity upper bound values of the nodes w.r.t. $C_1, C_2, ..., C_8$ under the three predefined time budgets (i.e., 2, 5, and 9). Similarly, $G_{\rho=2}$ represents a POI graph where every node has a matrix of 8×2 popularity upper bound (the two predefined time budgets are 3 and 9). From Fig.13, we find that as ρ increases, the corresponding POI graph consumes more memory. However, the differences of the memory consumption among the three graphs are small, which means the matrix of popularity upper bound dose not lead to a large space overhead.

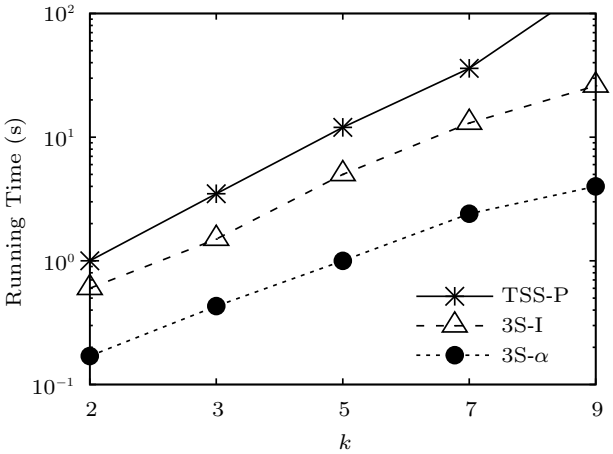


Fig.12. Effect of k in SF.

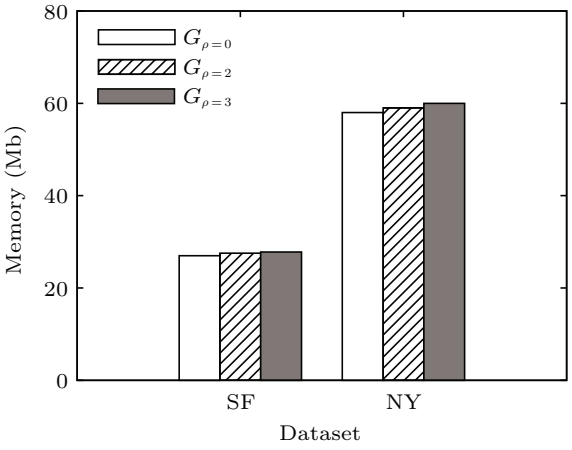


Fig.13. Memory cost of the POI graphs.

Comparison with the DIV-DP Algorithm[32]. We compare our single-stage algorithms with an adapted version of the DIV-DP algorithm proposed by Qin *et al.*[32] We adapt the DIV-DP algorithm to process SDkR queries as follows. We first construct a feasible route graph based on the current feasible routes. Then the DIV-DP algorithm is called to find a local optimal result. If this result is also global optimal as estimated by the algorithm, we return the query answer and terminate the algorithm. Otherwise, we add more feasible routes to the computation and repeat the above process. We conduct this set of experiments using the NY

dataset. Fig.14 shows the running time of the adapted DIV-DP algorithm and the two single-stage algorithms. We see that both 3S-I and 3S-α consistently outperform the DIV-DP algorithm when we vary k and t_{q}. This is because for 3S-I and 3S-α, we maintain a PSS for intermediate results, while the adapted DIV-DP algorithm recomputes and traverses the feasible route graph repeatedly.

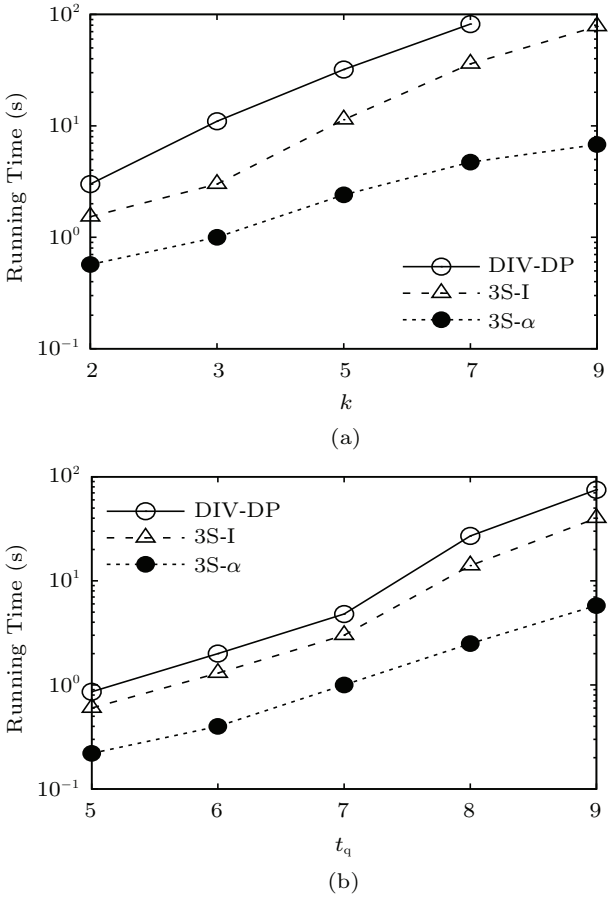


Fig.14. Comparison with DIV-DP. (a) Effect of k. (b) Effect of t_{q}.

Effectiveness of the SDkR Query. To show the impact of the spatial diversity constraint σ, we perform three SD3R queries on the SF dataset with spatial diversity constraints $\sigma = 0$, 2, and 4, respectively. The three queries share the same constraints in the desired starting POI (i.e., v_{q} = "Hostelling International"), time budget (i.e., $t_{\mathrm{q}} = 7$ hours), and the set of query categories (i.e., $S_{\mathrm{cat}} = \{C_1, C_6, C_8\}$). We denote the three queries as q_1, q_2, and q_3, respectively. Note that q_1 (i.e., $\sigma = 0$) is equivalent to the traditional top-3 routes query, which returns the routes with the highest popularity. We show the result routes of q_1, q_2, and q_3 in Figs.15(a)–15(c), respectively. As shown in the figures, the distances between the result routes increase when σ increases. According to Table 7, the result routes of q_1 (i.e., $\{r_1, r_2, r_3\}$) have overlapping POIs, while the result routes of q_2 (i.e., $\{r_1, r_2, r_4\}$) and q_3 (i.e., $\{r_1, r_5, r_6\}$) become spatially distant.

Fig.15. Effectiveness of the SDkR query. (a) $\sigma = 0$. (b) $\sigma = 2$. (c) $\sigma = 4$.

Table 7. Result Routes of the Three SD3R Queries

Route	POI
r_1	Alta Plaza Park (C_6)→Fine Arts Theatre (C_1)→Golden Gate Bridge (C_8)
r_2	Union Square (C_6)→Museum of Modern Art (C_1)→Coit Tower (C_8)
r_3	Union Square (C_6)→Coit Tower (C_8)→Lombard Street (C_1)
r_4	City Hall (C_8)→De Young Museum (C_1)→Golden Gate Park (C_6)
r_5	Museum of Modern Art (C_1)→Rincon Park (C_6)→Coit Tower (C_8)
r_6	Haight Street (C_8)→De Young Museum (C_1)→Golden Gate Park (C_6)

7 Conclusions

We formulated the spatial diversified top-k routes query and proposed three algorithms to process the query. We analyzed the algorithm costs and performed an empirical study on the algorithm performance. The first algorithm TSS-P processes the query in two stages. In the first stage, all the feasible routes that satisfy the query constraints are identified. In the second stage, the k-route combination with the highest popularity is found and returned. The other two algorithms, 3S-I and 3S-α, avoid generating all the feasible routes. They build up the optimal k-route combination progressively in the process of identifying feasible routes. The experimental results showed that the proposed algorithms are both effective and efficient. The 3S-α algorithm has the best performance, which saves up to 60% and 90% of the query processing time when $\alpha = 0.8$ compared with 3S-I and TSS-P, respectively.

In the future, we plan to investigate the possibility of further improving the efficiency of SDkR query via designing a spatial index structure that fits the SDkR query. In addition, we plan to investigate the effect of different spatial metrics (e.g., angle of routes) on the query result. Finally, we plan to consider processing the SDkR query on road networks.

References

[1] Su H, Zheng K, Huang J M, Liu T Y, Wang H Z, Zhou X F. A crowd-based route recommendation system — CrowdPlanner. In *Proc. the 30th International Conference on Data Engineering*, March 2014, pp.1178-1181.

[2] Lu H C, Chen C Y, Tseng V S. Personalized trip recommendation with multiple constraints by mining user check-in behaviors. In *Proc. the 20th International Conference on Advances in Geographic Information Systems*, November 2012, pp.209-218.

[3] Zhang C Y, Liang H W, Wang K, Sun K L. Personalized trip recommendation with PoI availability and uncertain traveling time. In *Proc. the 24th ACM International Conference on Information and Knowledge Management*, October 2015, pp.911-920.

[4] Hsieh H P, Li C T. Mining and planning time-aware routes from check-in data. In *Proc. the 23rd International Conference on Information and Knowledge Management*, November 2014, pp.481-490.

[5] Shang S, Ding R G, Yuan B, Xie K X, Zheng K, Kalnis P. User oriented trajectory search for trip recommendation. In *Proc. the 15th International Conference on Extending Database Technology*, March 2012, pp.156-167.

[6] Dai J, Liu C F, Xu J J, Ding Z M. On personalized and sequenced route planning. *World Wide Web: Internet and Web Information Systems*, 2016, 19(4): 679-705.

[7] Tang J Y, Sanderson M. Spatial diversity, do users appreciate it? In *Proc. the 6th Workshop on Geographic Information Retrieval*, February 2010, Article No. 22.

[8] Chen Z B, Shen H T, Zhou X F, Zheng Y, Xie X. Searching trajectories by locations: An efficiency study. In *Proc. the 29th ACM SIGMOD International Conference on Management of Data*, June 2010, pp.255-266.

[9] Shang S, Chen L S, Jensen C S, Wen J R, Kalnis P. Searching trajectories by regions of interest. *IEEE Transactions on Knowledge and Data Engineering*, 2017, 29(7): 1549-1562.

[10] Shang S, Ding R, Zheng K, Jensen C S, Kalnis P, Zhou X F. Personalized trajectory matching in spatial networks. *The International Journal on Very Large Data Bases*, 2014, 23(3): 449-468.

[11] Zheng K, Zheng B L, Xu J J, Liu G F, An L, Li Z X. Popularity-aware spatial keyword search on activity trajectories. *World Wide Web: Internet and Web Information Systems*, 2017, 20(4): 749-773.

[12] Zheng K, Yang Y, Shang S, Yuan N J. Towards efficient search for activity trajectories. In *Proc. the 29th International Conference on Data Engineering*, April 2013, pp.230-241.

[13] Shang S, Chen L S, Zheng K, Jensen C S, Wei Z, Kalnis P. Parallel trajectory-to-location join. *IEEE Transactions on Knowledge and Data Engineering*. doi: 10.1109/TKDE.2018.2854705.

[14] Wei L Y, Zheng Y, Peng W C. Constructing popular routes from uncertain trajectories. In *Proc. the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2012, pp.195-203.

[15] Cao X, Chen L S, Cong G, Xiao X K. Keyword-aware optimal route search. *Proceedings of the VLDB Endowment*, 2012, 5(11): 1136-1147.

[16] Li Y J, Yang W D, Dan W, Xie Z P. Keyword-aware dominant route search for various user preferences. In *Proc. the 20th International Conference on Database Systems for Advanced Applications*, April 2015, pp.207-222.

[17] Shang S, Chen L S, Wei Z W, Jensen C S, Wen J R, Kalnis P. Collective travel planning in spatial networks. *IEEE Transactions on Knowledge and Data Engineering*, 2016, 28(5): 1132-1146.

[18] Wen Y T, Yeo J, Peng W C, Hwang S W. Efficient keyword-aware representative travel route recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 2018, 29(8): 1639-1652.

[19] Liu H, Jin C Q, Yang B, Zhou A Y. Finding top-k optimal sequenced routes. In *Proc. the 34th International Conference on Data Engineering*, April 2018, pp.569-580.

[20] Shang S, Liu J, Zheng K, Lu H, Pedersen T B, Wen J R. Planning unobstructed paths in traffic-aware spatial networks. *Geo Informatica*, 2015, 19(4): 723-746.

[21] Soma S C, Hashem T, Cheema M A, Samrose S. Trip planning queries with location privacy in spatial databases. *World Wide Web: Internet and Web Information Systems*, 2017, 20(2): 205-236.

[22] Zheng B L, Su H, Hua W, Zheng K, Zhou X F, Li G H. Efficient clue-based route search on road networks. *IEEE Transactions on Knowledge and Data Engineering*, 2017, 29(9): 1846-1859.

[23] Xu J J, Gao Y J, Liu C F, Zhao L, Ding Z M. Efficient route search on hierarchical dynamic road networks. *Distributed and Parallel Databases*, 2015, 33(2): 227-252.

[24] Liang S S, Yilmaz E, Shen H, Rijke M D, Croft W B. Search result diversification in short text streams. *ACM Transactions on Information Systems*, 2017, 36(1): Article No. 8.

[25] Angel A, Koudas N. Efficient diversity-aware search. In *Proc. the 30th ACM SIGMOD International Conference on Management of Data*, June 2011, pp.781-792.

[26] Khan H A, Sharaf M A. Model-based diversification for sequential exploratory queries. *Data Science and Engineering*, 2017, 2(2): 151-168.

[27] Chen L S, Cong G. Diversity-aware top-k publish/subscribe for text stream. In *Proc. the 34th ACM SIGMOD International Conference on Management of Data*, May 2015, pp.347-362.

[28] Fan W F, Wang X, Wu Y H. Diversified top-k graph pattern matching. *Proceedings of the VLDB Endowment*, 2013, 6(13): 1510-1521.

[29] Yuan L, Qin l, Lin X M, Chang L J, Zhang W J. Diversified top-k clique search. *The International Journal on Very Large Data Bases*, 2016, 25(2): 171-196.

[30] Carbonell J G, Goldstein-Stewart J. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proc. the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, August 1998, pp.335-336.

[31] Vieira M R, Razente H L, Barioni M C, Hadjieleftheriou M, Srivastava D, Traina C, Tsotras V J. On query result diversification. In *Proc. the 27th International Conference on Data Engineering*, April 2011, pp.1163-1174.

[32] Qin L, Yu Y J, Chang L J. Diversifying top-k results. *Proceedings of the VLDB Endowment*, 2012, 5(11): 1124-1135.

[33] Garey M R, Johnson D S. Computers and intractability: A guide to the theory of NP-completeness. *Society for Industrial and Applied Mathematics*, 1982, 24(1): 90-91.

[34] Jain A, Sarda P, Haritsa J R. Providing diversity in k-nearest neighbor query results. In *Proc. the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, May 2004, pp.404-413.

[35] Lee K C K, Lee W C, Leong H V. Nearest surrounder queries. *IEEE Transactions on Knowledge and Data Engineering*, 2010, 22(10): 1444-1458.

[36] Kucuktunc O, Ferhatosmanoglu H. λ-diverse nearest neighbors browsing for multidimensional data. *IEEE Transactions on Knowledge and Data Engineering*, 2013, 25(3): 481-493.

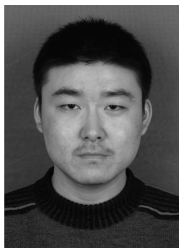
[37] Ference G, Lee W C, Jung H J, Yang D N. Spatial search for K diverse-near neighbors. In *Proc. the 22nd ACM International Conference on Information and Knowledge Management*, October 2013, pp.19-28.

[38] Zhang C Y, Zhang Y, Zhang W J, Lin X M, Cheema M A, Wang X Y. Diversified spatial keyword search on road networks. In *Proc. the 17th International Conference on Extending Database Technology*, March 2014, pp.367-378.

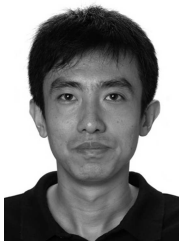
[39] Godsil C, Royle G F. Algebraic Graph Theory. Springer, 2001.

[40] Chiba N, Nishizeki T. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 1985, 14(1): 210-223.

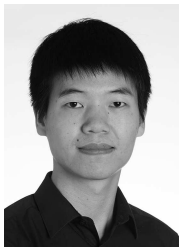
[41] Bao J, Zheng Y, Mokbel M F. Location-based and preference-aware recommendation using sparse geo-social networking data. In *Proc. the 20th International Conference on Advances in Geographic Information Systems*, November 2012, pp.199-208.



Hong-Fei Xu received his M.E. degree in computer science from Shenyang Jianzhu University, Shenyang, in 2014. He is currently a Ph.D. candidate in computer software and theory at Northeastern University, Shenyang. His research interests include spatial data management and data mining.



Yu Gu received his Ph.D. degree in computer software and theory from Northeastern University, Shenyang, in 2010. Currently, he is a professor in Northeastern University, Shenyang. His current research interests include spatial data management and graph data management. He is a senior member of CCF.



Jian-Zhong Qi is a lecturer in the Department of Computing and Information Systems at the University of Melbourne, Melbourne. He received his Ph.D. degree in computer science from the University of Melbourne, Melbourne, in 2014. He was an intern at Toshiba China R&D Center and Microsoft Redmond in 2009 and 2013, respectively. His research interests include spatio-temporal databases, location-based social networks, information extraction, and text mining.

Jia-Yuan He received her M.E. degree in computer science from Huazhong University of Science and Technology, Wuhan, in 2011. She is currently a Ph.D. candidate in computer science at the University of Melbourne, Melbourne. Her research interests include spatial data management and data mining.

Ge Yu received his Ph.D. degree in computer science from Kyushu University, Fukuoka, in 1996. He is currently a professor at Northeastern University, Shenyang. His research interests include distributed and parallel database, OLAP and data warehousing, data integration, graph data management, etc. He has published more than 200 papers in refereed journals and conferences. He is a fellow of CCF, and a member of ACM, IEEE, and the IEEE Computer Society.

Appendix

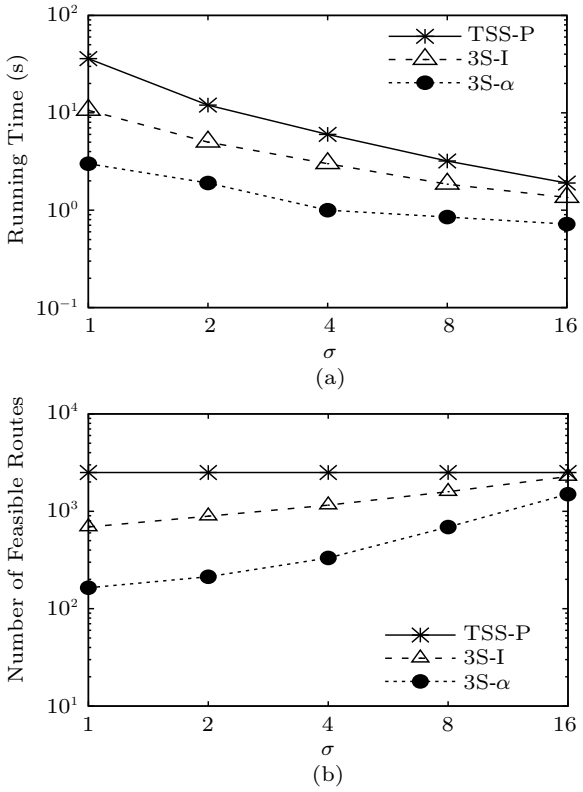


Fig.A1. Effect of σ in NY. (a) Running time. (b) Number of feasible routes.

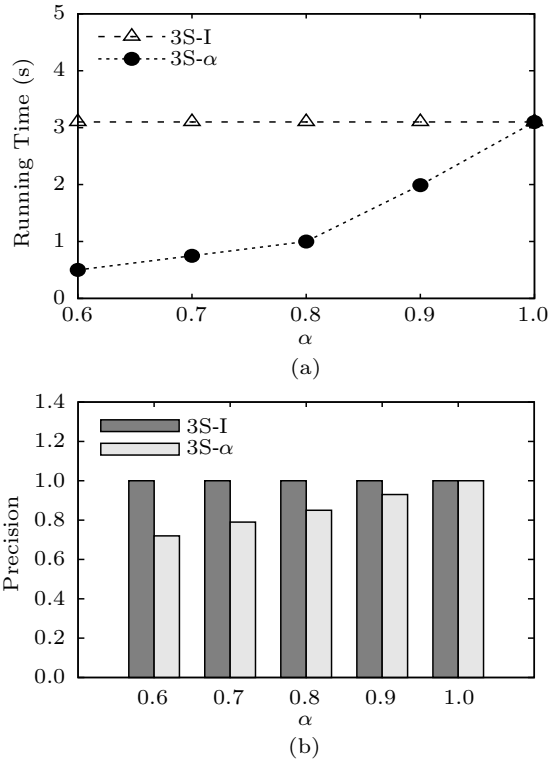


Fig.A2. Effect of α in NY. (a) Running time. (b) Precision.

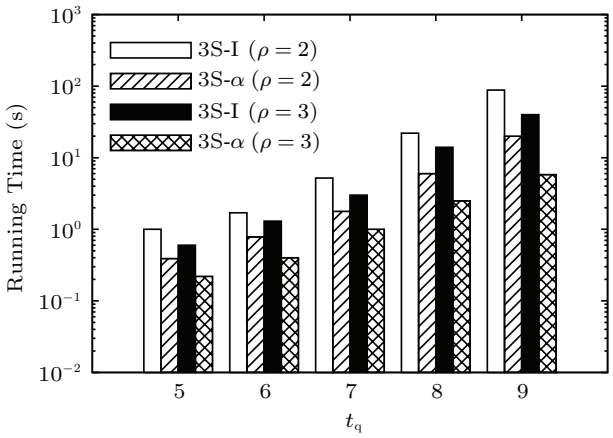


Fig.A3. Effect of t_{q} in NY.

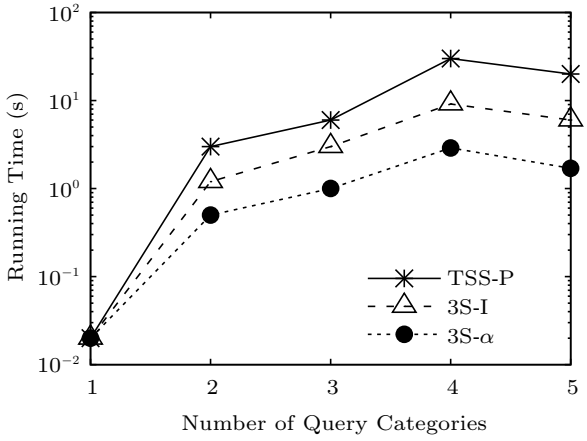


Fig.A4. Effect of $|S_{\mathrm{cat}}|$ in NY.