Data Science and Engineering (2021) 6:180–191
https://doi.org/10.1007/s41019-021-00153-5

Querying Optimal Routes for Group Meetup

Bo Chen[1] · Huaijie Zhu[1] · Wei Liu[1] · Jian Yin[1] · Wang-Chien Lee[2] · Jianliang Xu[3]

Received: 15 September 2020 / Revised: 28 January 2021 / Accepted: 19 February 2021 / Published online: 15 March 2021

Abstract
Motivated by location-based social networks which allow people to access location-based services as a group, we study a novel variant of *optimal sequenced route* (*OSR*) queries, optimal sequenced route for group meetup (OSR-G) queries. OSR-G query aims to find the optimal meeting POI (point of interest) such that the maximum users' route distance to the meeting POI is minimized after each user visits a number of POIs of specific categories (e.g., gas stations, restaurants, and shopping malls) in a particular order. To process OSR-G queries, we first propose an *OSR-Based* (*OSRB*) algorithm as our baseline, which examines every POI in the meeting category and utilizes existing OSR (called *E-OSR*) algorithm to compute the optimal route for each user to the meeting POI. To address the shortcomings (i.e., requiring to examine every POI in the meeting category) of *OSRB*, we propose an *upper bound based filtering* algorithm, called *circle filtering* (*CF*) algorithm, which exploits the circle property to filter the unpromising meeting POIs. In addition, we propose a *lower bound based pruning* (*LBP*) algorithm, namely *LBP-SP* which exploits a shortest path lower bound to prune the unqualified meeting POIs to reduce the search space. Furthermore, we develop an approximate algorithm, namely APS, to accelerate OSR-G queries with a good approximation ratio. Finally the experimental results show that both *CF* and *LBP-SP* outperform the *OSRB* algorithm and have high pruning rates. Moreover, the proposed approximate algorithm runs faster than the exact OSR-G algorithms and has a good approximation ratio.

Keywords Route queries · Group meeting · Pruning algorithms

1 Introduction

Optimal sequenced route (OSR) [23] queries aim to find an optimal route passing through a sequence of points of interest (POIs) of specific categories (e.g., gas stations, restaurants, and shopping malls) in a particular order. An example of OSR query in a road network is shown in Fig. 1. The example shows four POI categories, where s_1, s_2, s_3 are the supermarkets, r_1, r_2 are the restaurants, and so on. Given a user u_1 starting at the current position, passing through a sequence of POIs (*restaurant*, *supermarket*) and arriving at the destination d_1, the optimal sequenced route is (u_1, r_1, s_1, d_1)[1] with a cost of 14. The OSR query is firstly studied by Sharifzadeh et al. [23, 24], followed up by a number of variants [4–8, 12, 13, 15, 17, 18, 20, 22]. However these prior works assume that the optimal route is designed for only one user instead of multiple users to have a meeting. With the rapid development of location-based social network, it attracts not only single user but also a group of users to access location-based services, e.g., finding the optimal sequenced routes. For example, there is an activity which requires multiple users to take part in. Every user has to visit some different POIs (e.g., finishing some tasks) and finally meets at a place[2] to be determined. The OSR query is not applicable for this situation directly. This realistic need motivates our research in this paper.

✉ Huaijie Zhu
zhuhuaijie@mail.sysu.edu.cn

Wang-Chien Lee
wlee@cse.psu.edu

Jianliang Xu
xujl@comp.hkbu.edu.hk

[1] School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China

[2] Department of Computer Science and Engineering, The Pennsylvania State University, State College, US

[3] Hong Kong Baptist University, Kowloon Tong, China

[1] The detailed route actually is $(u_1, d_2, r_1, s_1, u_2, d_1)$. For brevity, we only list the start point, the corresponding POIs and the destination point.

[2] The meeting places can also be extended to be a lot of candidate POIs, specifically just one is finally chosen for meeting.

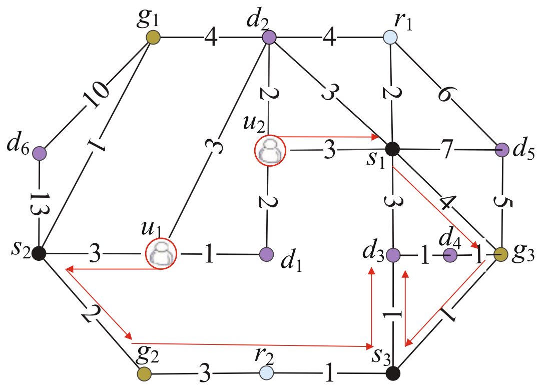


Fig. 1 A road network with "supermarket"=$\{s_1, s_2, s_3\}$, "gas station"= $\{g_1, g_2, g_3\}$, "restaurant"=$\{r_1, r_2\}$ and meeting category c_d ="bar" with POIs $\{d_1, d_2, d_3, d_4, d_5, d_6\}$

Motivated by the OSR query for group meetup, we propose a new query type, *optimal sequenced route for group meetup* (*OSR-G*) query in this paper. Given a group of users U, where each user u_i corresponds to a starting position s_i and a sequence of POI categories C_{u_i} ($C\_U= \{C_{u_1}, C_{u_2}$,..., C_{u_i},..., $C_{u_n}\}$), and the group meeting POI category c_d, the OSR-G query is a tuple (U, C_U, c_d), which is to find the optimal meeting POI d (d belongs to category c_d) and an optimal route for each user to minimize the maximum user's route distance to the meeting POI. An example of the OSR-G query is illustrated in Fig. 1. A group of users u_1 and u_2 would like to meet at a bar (e.g., candidates are $\{d_1, d_2, \ldots, d_6\}$) today. However, before their meeting, u_1 and u_2 should visit a sequence of POIs (*restaurant*, *supermarket*) and (*gas station*, *supermarket*), respectively. The answer to this OSR-G query is that user u_1 visits the route (u_1, r_2, s_3, d_3) with a cost of 10 and user u_2 visits the route (u_2, g_3, s_3, d_3) with a cost of 9. Since they will meet after the last user u_1 finishes his route with the cost of 10 (i.e., the cost of u_1's route is 10 and the cost of u_2's route is 9), the meeting cost with respect to d_3 is 10. While choosing d_1 (or other candidates) as the meeting place, the meeting cost is 14 (or larger than 10). Thus, the optimal meeting place is d_3 with the minimal cost 10 for this OSR-G query. A major challenge faced in processing OSR-G queries is that we need to consider all the query users' optimal route simultaneously for meeting, which is a combinatorial optimization problem. OSR-G query is useful for a popular real-life application, i.e., meeting point recommendation.

Meeting point recommendation A group of friends located at different parts of a city want to have a meeting in a coffee shop. Before the meeting, everyone has some tasks to do such as going to a bank and then shopping in a supermarket. In this scenario, an optimal meeting coffee shop, as well as the routing schedules, can be recommended to the users so that they can meet as soon as possible while satisfying all users' requirements.Our query is useful for recommending the optimal meeting point if the users don't have a good idea about where to meet up.

In this paper, we first propose an OSR-Based (*OSRB*) algorithm as our baseline to solve OSR-G problem. *OSRB* algorithm utilizes an OSR algorithm by examining all meeting POIs in the meeting category to obtain the optimal solution. *OSRB* algorithm requires to enumerate all the POIs in the meeting POI category, thus it is very time-consuming. To improve the baseline, we develop the *circle filtering* (*CF*) algorithm which utilizes the circle property (i.e., an upper bound as the circle radius) to filter the unpromising meeting POIs by narrowing the filtering radius gradually. To further improve the efficiency, we design an efficient *lower bound based pruning* (LBP) algorithm, namely *shortest path lower bound based* (*LBP-SP*) algorithm which exploits a shortest path lower bound to prune the unqualified meeting POIs to do early termination. Furtherly, we develop an approximate algorithm, namely *approximate points select* (*APS*) algorithm. Finally, We validate our ideas and evaluate the proposed OSR-G algorithms using both real and synthetic datasets.

To the best of our knowledge, this is the first attempt to tackle OSR-G problem. This paper makes five contributions:

- We formalize a new variant of OSR query, namely OSR-G query, which finds the optimal meeting point for a group of users taking every user's query requirements into consideration.
- We propose three exact OSR-G algorithms, namely *OSRB*, *CF* and *LBP-SP* algorithms. Both *CF* and *LBP-SP* can efficiently answer the OSR-G query.
- We prove that any lower bound estimating the cost of an OSR can be plugged in the LBP algorithms, which can become a general framework to solve OSR-G problem.
- We develop an approximate OSR-G algorithm, namely *APS*. *APS* runs faster than the exact OSR-G algorithms and gives a solution with an approximation ratio not greater than θ.
- We verify our proposed OSR-G algorithms in both real and synthetic datasets. The results show that *CF* and *LBP-SP* outperform the *OSRB* algorithm a lot in running time. The proposed approximate OSR-G algorithm runs faster than the exact algorithms and gives good approximate solutions.

2 Related Work

In this section, we overview the related works on optimal route queries, which can be classified into *single user queries* and *multiple users queries*. The single user queries aim to find the optimal route for a single user, and multiple users queries are designed for multiple users.

2.1 Single User Queries

Li et al. [14] first propose the *trip planning query* (TPQ) where a user starts at q and ends at d passing through a set of POIs without a particular order. Several approximate algorithms have been proposed to solve the TPQ. After that, the *optimal sequenced route* (OSR) query, a variant of TPQ, is proposed by Sharifzadeh and Kolahdouzan [23]. Different from TPQ, the OSR query visits a number of POIs in a particular sequenced order, and the destination is not restricted to one predefined point but the candidate points in the last POI category. Two algorithms, namely LOAD and PNE, are designed to answer the OSR queries [23]. Chen et al. [5] study the *multi-rule partial sequenced route* (MRPSR) query, which finds the optimal route via a number of POIs in a partial visiting order defined by the user. Obviously, the MRPSR query is more general and can be converted to TPQ and OSR queries. Yaron Kanza et al. [13] study sequenced route search with order constraints using interactive search methods. And Ohsawa et al. [18] study the OSR query in Euclidean space, and develop the EOSR algorithm based on incremental Euclidean restriction (IER) [19]. Then, Costa et al. [7] propose TD-OSR algorithm to find an optimal time-dependent sequenced route (OTDSR) in road network. Yuya Sasaki et al. [22] propose the *skyline sequenced route* (SkySR) query, which is similar to OSR query, the difference is that SkySR query does not strictly obey the passing categories, but adopts the similar categories using a semantic hierarchy. Jian Dai et al. [8] propose the *Personalized and Sequenced Route* (PSR) query, which considers both the sequenced constraint and personalized category preferences, and develop a framework to solve the query. Similar to [8], Francesco Lettich et al. [12] propose the *Trade-Off Aware Sequenced Routing* (TASeR) query, which extends the OSR query with a POI cost (that is, each POI has a visiting cost), and an approach using the linear skyline paradigm to process the query. Recently, Li et al. present and tackle the rating constrained optimal sequenced route (RCOSR) query problem in which the POIs in the sequenced route should satisfy category rating thresholds [16].

The above works all consider the routes for a single user while our OSR-G queries are for a group of users. Thus, these works on TPQ queries or OSR queries can not be directly applied to our problem.

2.2 Multiple Users Queries

For the case of multiple users queries, we further categorize them into *different destinations* and *one common destination*.

2.2.1 Different Destinations

Hashem et al. [10] propose *group trip planning* (GTP) queries in Euclidean space, which aim to find the minimum of the total trip distance of group members with different departure points and destination points. GTP queries plan the routes, which pass a common set of POIs. Hashem et al. [9] extends work [10] to solve GTP queries in both Euclidean space and road networks. Both the total and maximum trip distance of the group members are minimized in this work. Ahmadi et al. [2] study a variant of GTP queries, namely SGTP where the visiting POIs are predefined by users in road networks. At the same time, works [2, 21] study *group optimal sequenced route* (GOSR) queries, which are inherently the same as SGTP queries. They employ the elliptical properties as pruning strategies to process the query. Jahan et al. [11] propose the group trip scheduling (GTS) queries which find the independent trips for the group members with minimum aggregate trip distance. Unlike GTP queries, the visiting POIs in GTS queries are scheduled among the group members which means that some POIs are not necessarily visited by every group member.

Although these works, e.g., GTP queries, SGTP queries and GTS queries, have considered a group of users (multiple users), they are not designed for the meeting of a group of users. Thus, the OSR-G query has not been studied previously.

2.2.2 Common Destination

Another relevant work is the optimal meeting point queries (OMP), which are involved in one common destination. OMP queries focus on finding an optimal meeting point for a group of users, which minimizes the aggregate distance of the group members' positions to the meeting point. One exact algorithm and an approximate algorithm are developed for OMP query by Yan et al. [25]. Yan et al. [26] extends work [25] in both Euclidean space and road networks, and proposes the algorithms to solve OMP queries with two aggregate distance (i.e., sum and maximum distance) from the group members' positions to the meeting point. Ahmadi and Nascimento [3] study the k-optimal meeting points for public transit (kOMPPT) queries, which aims to find k optimal meeting points to minimize the aggregate distance of a group of users. Different from [26], the group members have their predefined public transit routes such as a certain subway route and any subway stop can be the possible starting point.

Different from the above works, our OSR-G query is to find the optimal solution which every user must visit different sequences of POIs before reaching the meeting point.

3 Preliminaries

In this section, we define some terms and notations used in the paper. Then we formally give the problem definition and present our baseline idea.

3.1 Definitions

In this work, we focus on route queries over the road network. Thus, we define the notion of road network and POIs, and use them to define OSR query, feasible group routes and optimal group routes.

Definition 1 (*Road Network*) A road network is represented as an undirected graph G, which includes a vertex set V and an edge set $E \subseteq V \times V$. For an edge (a, b) in G, $w(a, b)$ denotes the weight of the edge, e.g., the travel distance when traversing edge (a, b).

Definition 2 (*Point of Interest, POI*). POI, a special kind of vertex in a road network, belongs to one or more categories such as gas stations or restaurants.

Based on the notion of POI, we now define the optimal sequenced route (OSR) query.

Definition 3 (*OSR Query* [23]) Given a starting point q and a sequence of categories $C_q = (c_1, c_2, \ldots, c_m)$ and a destination point d[3], the OSR query is a triple (q, d, C_q), which finds a route $\overrightarrow{R}$=$(q,p_1,p_2,\ldots,p_m, d)$ $(p_i \in c_i, 1 \le i \le m)$ such that $L(\overrightarrow{R})$ [4] $\le L(\overrightarrow{R'})$ where $\overrightarrow{R'}$[5] $\in \{\overrightarrow{route} | \overrightarrow{route} = (q, p'_1, p'_2, \ldots, p'_m, d), p'_i \in c_i, 1 \le i \le m\}$.

By considering the OSR query for a group of users, we now define the *feasible group routes* and *meeting cost* for route queries by a group of users.

Definition 4 (*Feasible Group Routes* (*FGRoutes*)) Given a group of users $U = \{u_1, u_2, \ldots, u_n\}$, where each user u_i corresponds to a starting position s_i, a sequence of categories C_i, and a meeting point d. Assume every user u_i in U starts from s_i, passes through its own category sequences C_{u_i} and finally meets each other at d. Thus, each user has his own feasible route $\overrightarrow{R_i}$ (see it in Definition 3). We say the *feasible group routes* (*FGRoutes*) are consisted of these feasible routes. Correspondingly, the meeting cost (or the cost of *FGRoutes*, denoted by $L(FGRoutes)$) is the maximum cost of the route among each users' route.

In many cases, there exists multiple feasible group routes for a given group of users and a meeting point.

Example 1 An example of *FGRoutes* is illustrated in Fig. 1. Assume there is a group of users U=$\{u_1, u_2\}$ where user u_1 needs to visit the category sequence (*restaurant*, *supermarket*), and user u_2 is to visit the category sequence (*gas station*, *supermarket*). They then meet at d_4 after visiting the POIs corresponding to its own category sequences. The user u_1 can follow the route $\overrightarrow{R_1} = (u_1, r_1, s_1, d_4)$ with $L(\overrightarrow{R_1}) = 13$, and u_2 can follow the route $\overrightarrow{R_2} = (u_2, g_1, s_2, d_4)$ with $L(\overrightarrow{R_2}) = 15$. Since $L(\overrightarrow{R_2}) > L(\overrightarrow{R_1})$, the meeting cost for U is 15. $\overrightarrow{R_1}$ and $\overrightarrow{R_2}$ constitute *FGRoutes* for U.

Definition 5 (*Optimal Group Routes* (*OGRoutes*)). Given a group of users U=$\{u_1, u_2, \ldots, u_n\}$, where each user u_i corresponds to a starting position s_i, a sequence of categories C_{u_i}, and a meeting point d. Each user has the optimal route $\overrightarrow{ORoute^{u_i}}$ with respect to s_i, d and C_{u_i}. The meeting cost for U and d, denoted by $L(U, d)$, is the maximum cost of the route among those optimal routes $\{\overrightarrow{ORoute^{u_1}}, \overrightarrow{ORoute^{u_2}}, \ldots, \overrightarrow{ORoute^{u_n}}\}$. The optimal routes for U are called the *optimal group routes (OGRoutes)*. Accordingly, $L(U, d)$ is the cost of OGRoutes. The OGRoutes ending at d is denoted by $\overrightarrow{OGRoutes_d}$, and the $\overrightarrow{ORoute}$ for u ending at d is denoted by $\overrightarrow{ORoute^u_d}$.

Continue with the Example 1 in Fig. 1. The route (u_1, r_2, s_3, d_4) with a cost of 11 is the optimal route for u_1, and the route (u_2, g_3, s_3, d_4) with a cost of 10 is the optimal route for user u_2. Therefore, the meeting cost is 11 with the maximum cost among these two optimal routes. Correspondingly, these two routes constitute the optimal group routes.

3.2 Problem Formulation and Baseline Algorithm

In this section, we formally define the OSR-G query.

OSR-G query Given a group of users U, where each user u_i corresponds to a starting position s_i and a sequence of categories C_{u_i}, and the group meeting POI category c_d, the OSR-G query is a tuple (U, C_U, c_d), which aims to find the optimal meeting POI d_o $(d_o \in c_d)$ and the optimal route for each user such that the meeting cost $L(U, d_o)$ is the minimum, i.e., $L(U, d_o) = \underset{d \in c_d}{min} L(U, d)$.

According to the definition of OSR-G query, the basic idea is to examine all POIs in c_d to guarantee the optimality of an OSR-G without missing the optimal meeting POI. For each candidate meeting POI d, we invoke the existing OSR

[3] Original OSR query [23] does not have a destination point explicitly, thus we add d as the last query category for the original OSR query to fit our problem.

[4] $L(\overrightarrow{R})$ denotes the cost of the route $\overrightarrow{R}$.

[5] Such a route is also called a feasible route.

algorithm (i.e., *E-OSR*[6]) to find the optimal route for each user. After examining all the candidate meeting POIs in c_d, the solution with minimal cost is returned as the OSR-G result. This *OSR-based (OSRB)* algorithm, serves as a baseline for comparison in our evaluation.

4 Upper Bound Based Filtering Algorithm : CF

As shown in *OSRB* algorithm, it costs too much overhead to examine every POI in the meeting category. To address this shortcoming, a natural idea is to filter the unpromising meeting point candidates to reduce the search space. Accordingly, we propose an *upper bound based filtering* algorithm, namely *circle filtering* (*CF*) algorithm.

The main idea of circle filtering is to utilize the circle property (which is introduced next) to do filtering. For the current examining meeting POI, every user takes a feasible route firstly to make the feasible group routes (FGRoutes), and every user takes the cost of FGRoutes as his radius which is an upper bound of the cost of OGRoutes to draw a circle. By computing the intersection area of each user's circle, if one candidate meeting POI is located outside this intersection area, it can be filtered safely, as specified in *Property* 1 (the circle property). By narrowing the radius gradually, more unpromising meeting points can be filtered.

Property 1 *Given an OSR-G query, a candidate meeting POI p, and the intersection area which is generated by each user's circle, p can be filtered iff p is outside the intersection area.*

Proof Suppose the current radius is obtained from d and its length is $L(FGRoutes_d)$ and there is a meeting POI d_{out} outside the intersection area of every user's drawing circle.

We prove it by contradiction (suppose d_{out} is the optimal meeting point). Since d_{out} is outside the intersection area, there must exists a user u such that $L(u, d_{out}) > L(FGRoutes_d)$. By the definition of OGRoutes (see *Definition*5), we have $L(OGRoutes_{d_{out}}) \geq L(\overrightarrow{ORoute^{u}_{d_{out}}}) \geq L(u, d_{out}) > L(FGRoutes_d) \geq L(OGRoutes_d)$. We have $L(OGRoutes_{d_{out}}) > L(OGRoutes_d)$. While d_{out} is the optimal meeting point, thus $L(OGRoutes_{d_{out}}) \leq L(OGRoutes_d)$, which makes a contradiction. The proof completes. □

Figure 2 shows an example of illustrating *Property*1. Before meeting at a "bar", user u_1 has a visiting sequence (*restaurant*, *supermarket*) and user u_2 has another visiting sequence (*gas station*, *supermarket*). If users u_1 and u_2 take

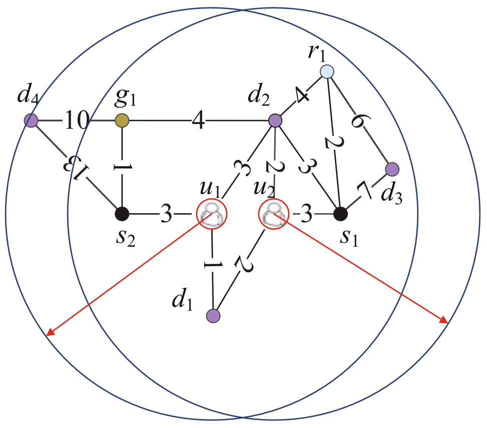


Fig. 2 A road network with "supermarket"={s_1, s_2}, "gas station"={g_1}, "restaurant"={r_1} and "bar"= {d_1, d_2, d_3, d_4}

d_1 as the initial meeting point, u_1 queries a feasible route $\overrightarrow{R_1}$ =(u_1,r_1,s_1,d_1) with a cost of 14 while u_2 queries a feasible route $\overrightarrow{R_2} = (u_2, g_1, s_2, d_1)$ with a cost of 11. Then we have $L(FGRoutes_{d_1})$=14, and u_1,u_2 take a radius of 14 with his starting point as the center to draw his own circle respectively (u_1 and u_2 will draw 2 circles totally) and make an intersection of these two circles. We find that the destination point d_4 is outside the intersection area. Since the network distance (the shortest path distance) from the starting point of u_2 to d_4 is larger than 14, d_4 can be filtered safely.

While this circle filtering idea is simple but efficient for filtering the unpromising candidates, however, it still faces three issues in processing the OSR-G query. One is how to select a good initial meeting point to achieve the optimal filtering ability. The second one is how to calculate a feasible route (or a smaller upper bound) for every user in the group efficiently. The last issue is how to narrow the filtering radius more faster to filter more unpromising meeting points.

$$d_{initial} = \arg\min_{d} \max_{u_i \in U, d \in c_d} ShortestPathCost(u_i, d) \qquad (1)$$

With respect to the first issue, the initial meeting point $d_{initial}$ in *CF* algorithm is selected by Equation 1. While for the second one, it is efficient to perform the greedy search by nearest neighbor query to get a feasible route. Assuming a user u_i at a start point s_i who has a visiting sequence $C = (c_1, c_2, \ldots, c_m)$, the user u_i obtains the feasible route $\vec{R} = (s_i, p'_1, p'_2, \ldots, p'_m, d)$ where p'_1 is the nearest neighbor in category c_1 of s_i and p'_j is the nearest neighbor in category c_j of p'_{j-1} $(2 \leq j \leq m)$. The cost of acquired $FGRoutes_{d_{initial}}$ is used as the initial filtering radius. We can try every POI in c_d to get the minimum radius to filter the unpromising meeting points by doing the same greedy search. Since the difference between those feasible routes returned by the greedy search is reaching at different meeting points, we use

[6] *E-OSR* is an OSR algorithm modified in [7].

$FGRoutes_{d_{initial}}$ to retrieve the minimum radius among all *FGRoutes* to avoid the same nearest neighbor search.

To narrow the filtering radius more quickly when examining the remaining candidates, *CF* retrieves the optimal route for each user as $OGRoutes_d$ rather than executing greedy search to obtain the feasible route. If $L(OGRoutes_d)$ is less than the current filtering radius, we update the filtering radius .Otherwise, we repeat the above steps until there is no meeting point to be examined and the query processing ends.

Algorithm 1: CIRCLE FILTERING

```
Input: A group of users $U = \{u_1, u_2, ..., u_n\}$, and a meeting
       category $c_d$;
Output: The OSR-G result $OGRoutes$;
1  Select the initial meeting point $d_{initial}$ by Equation 1;
2  Retrieve $FGRoutes_{d_{initial}}$ by greedy search;
3  $radius \leftarrow L(FGRoutes_{d_{initial}})$;
4  Filter the meeting points in $c_d$ if $\exists u \in U, L(u,d) > radius$;
5  Get the minimum radius $r' \leftarrow min\{L(FGRoutes_d), \forall d \in c_d\}$;
6  if $radius < r'$ then
7      $radius \leftarrow r'$;
8      Filter the meeting points in $c_d$ if
         $\exists u \in U, L(u,d) > radius$;
9  Select a candidate meeting point $d$ from $c_d$;
10 $OGRoutes \leftarrow OGRoutes_d$;
11 while $c_d$ is not empty do
12     if $L(OGRoutes) < radius$ then
13         $radius \leftarrow L(OGRoutes)$;
14         Filter the meeting points in $c_d$ if
             $\exists u \in U, L(u,d) \geq radius$;
15     if $c_d$ is not empty then
16         Select a candidate meeting point $d$ from $c_d$;
17         if $L(OGRoutes_d) < L(OGRoutes)$ then
18             $OGRoutes \leftarrow OGRoutes_d$;
19 return $OGRoutes$;
```

The pseudo code of CF algorithm is show in Algorithm 1. We show the running example in Fig. 2.Firstly, CF algorithm selects the initial meeting point d_1 by Equation 1 and $FGRoutes_{d_1}$=$\{(u_1,r_1,s_1,d_1),(u_2,g_1,s_2,d_1)\}$ is obtained by greedy nearest neighbor search and *radius* is set to $L(FGRoutes_{d_1})$=14. d_6 is filtered firstly since $max\{L(u_1,d_6), L(u_2,d_6)\} = 16 > 14$. Then CF algorithm obtains a new radius which is acquired from $FGRoutes_{d_2}$ among all feasible group routes and radius is updated to 12. Similar to d_6, d_5 is filtered because it is not located into the intersection area. In the next stage, the candidate meeting points are examined according to the cost of their feasible group routes in an ascending order. Thus d_2 is examined firstly and *OGRoutes* is set to $OGRoutes_{d_2}$. Again CF algorithm examines d_1 and does nothing since $L(OGRoutes_{d_1}) = 14 > 12$. Next, d_3 is examined, *OGRoutes* and the current radius are updated which causes d_4 to be filtered. Finally, there is no candidate meeting points to be examined, and CF algorithm returns the optimal solution.

5 Lower Bound Based Pruning Algorithm: LBP-SP

Although the proposed *CF* algorithm can filter many unpromising meeting POIs, it still takes much overhead on narrowing the filtering radius gradually. In contrast to the *upper bound based filtering* idea, we propose a *lower bound based pruning* (*LBP*) algorithm to process OSR-G query.

If we sort the POIs in c_d in an ascending order according to the lower bound of the cost of the corresponding OGRoutes, the optimal meeting POI will be found more quickly by the early termination using its lower bound. Once having the current optimal meeting POI, if the lower bound of the OGRoutes is not less than the current meeting cost, there is no need to examine the remaining POIs. By this way, a lot of unpromising POIs in c_d can be pruned and the early termination is also achieved. Thus, motivated by this idea, the key issue is how to compute a tight lower bound of the OGRoutes to design a LBP algorithm.

For an user u in the group U, let $lb(u, d)$ be the lower bound of the cost of ORoute for u. It is easy to deduce that for a group U, $lb(U, d) = \max\limits_{u \in U} lb(u, d)$ is also the lower bound of the cost of the OGRoutes w.r.t d, which is shown in *Lemma* 1.

Lemma 1 *If $lb(u, d)$ is a lower bound of the cost of user u's $\overrightarrow{ORoute_d^u}$, then $lb(U, d) = \max\limits_{u \in U} lb(u, d)$ is a lower bound of the cost of $OGRoutes_d$.*

Proof If $lb(u, d)$ is a lower bound, then we have $lb(u, d) \leq L(\overrightarrow{ORoute_d^u})$ ($\overrightarrow{ORoute_d^u}$ is an ORoute ending at d for user u). Supposing $L(\overrightarrow{ORoute_d^{u_{x_1}}}) \leq L(\overrightarrow{ORoute_d^{u_{x_2}}}) \leq ... \leq L(\overrightarrow{ORoute_d^{u_{x_n}}})$, then we have $L(U, d) = L(\overrightarrow{ORoute_d^{u_{x_n}}})$. We prove $lb(U, d) \leq L(U, d)$ by the following two cases.

Case 1 If $lb(u_{x_n}, d) \geq lb(u_{x_i}, d), \forall u_{x_i} \in U$, obviously we have $lb(U, d)$=$lb(u_{x_n}, d) \leq L(\overrightarrow{ORoute_d^{u_{x_n}}}) \leq L(U, d)$. Therefore, $lb(U, d) \leq L(U, d)$.

Case 2 Otherwise, $\exists u_{x_j}, \forall u_{x_i} \in U, lb(u_{x_j}, d) \geq lb(u_{x_i}, d)$. Then we have $lb(U, d)$=$lb(u_{x_j}, d) \leq L(\overrightarrow{ORoute_d^{u_{x_j}}}) \leq L(\overrightarrow{ORoute_d^{u_{x_n}}}) \leq L(U, d)$. Therefore, $lb(U, d) \leq L(U, d)$ holds.

Combining case 1 and 2 above, the proof completes. □

According to *Lemma* 1, our goal is transferred to design a tight lower bound $lb(u, d)$ of the cost of ORoute for u. For the lower bound of the cost of ORoute for u, one direct idea is to utilize the cost of the shortest path from user u's starting point to the meeting point d as the lower bound $lb(u, d)$, which is called *SP* lower bound. Based on this *SP* lower bound, we propose a *Shortest Path based LBP* algorithm, namely *LBP-SP*.

Algorithm 2 shows the pseudo-code of the framework of LBP algorithm. Given a group of users and the meeting category c_d, LBP first computes $lb(U, d)$ for all d in c_d to generate a sorted list $List_s$ (line 2). Next, it examines the candidate meeting POIs in $List_s$ one by one (lines 3-10). If $lb(U,d) \geq L(OGRoutes)$, the examination of the remaining canditate POIs terminates (lines 4-5). Otherwise, LBP performs the OSR algorithm (i.e., E-OSR) to retrieve the optimal route for each user and these users' routes form the OGRoutes (lines 6-8). Moreover, it checks whether the cost of new *tempOGRoutes* is smaller than the current OGRoutes. If it is, *OGRoutes* is updated as *tempOGRoutes* (lines 9-10). Finally, the algorithm returns the OSR-G result (line 11).

Algorithm 2: LBP-ALGORITHM

```
Input: A group of users $U = \{u_1, u_2, ..., u_n\}$, and a meeting
       category $c_d$;
Output: The OSR-G result $OGRoutes$;
1  $OGRoutes \leftarrow null$; $//L(null) = \infty$
2  Compute $lb(U,d)$ by a lower bound function for all $d$ in $c_d$ to
     generate a sorted list $List_s$;
3  for each $d$ in $List_s$ do
4      if $lb(U,d) \geq L(OGRoutes)$ then
5          break;
6      for $u_i \leftarrow u_1$ to $u_n$ do
7          $\overrightarrow{tempOGRoutes[i]} \leftarrow$ E-OSR($u_i, C_{u_i}, d$);
8      $L(tempOGRoutes)=MAX\{L(\overrightarrow{tempOGRoutes[i]})|i = 1,2,...,n\}$;
9      if $L(OGRoutes) > L(tempOGRoutes)$ then
10         $OGRoutes \leftarrow tempOGRoutes$;
11 return $OGRoutes$;
```

An example of illustrating *LBP-SP* algorithm is recalled in Fig. 1. Firstly, *LBP-SP* calculates $lb(U, d)$ for all d in c_d and gets $lb(U, d_1)$= 2, $lb(U, d_2)$= 3, $lb(U, d_3)$= 9, $lb(U, d_4)$= 10, $lb(U, d_5)$= 13, $lb(U, d_6)$= 16. $List_s$= (d_1, d_2, d_3, d_4, d_5, d_6) is obtained by sorting $lb(U, d)$. Then the POIs in *sList* are examined sequentially. When computing the OGRoutes w.r.t d_1, the $\overrightarrow{ORoute}$ of user u_1 is returned as (u_1, r_1, s_1, d_1) with a cost of 14 and the $\overrightarrow{ORoute}$ of user u_2 is returned as (u_2, g_1, s_2, d_1) with a cost of 11. Thus, $L(U, d_1) = 14$ and the current optimal solution is $OGRoutes_{d_1}$. After that, *LBP-SP* examines d_2 for $lb(U, d_2) < L(U, d_1)$ and computes the OGRoutes for d_2. The $\overrightarrow{ORoute}$ of user u_1 is (u_1, r_1, s_1, d_2) with a cost of 12 and the $\overrightarrow{ORoute}$ of user u_2 is (u_2, g_1, s_2, d_2) with a cost of 12. Since $L(U, d_2) = 12 < L(U, d_1)$, the current best solution is updated to $OGRoutes_{d_2}$. Following the order in $List_s$, d_3 is examined, the $\overrightarrow{ORoute}$ of user u_1 is (u_1, r_2, s_3, d_3) with a cost of 10 and the $\overrightarrow{ORoute}$ of user u_2 is (u_2, g_3, s_3, d_3) with a cost of 9. The current optimal solution is updated to $OGRoute_{d_3}$ since $L(U, d_3) = 10 < 12$. Finally, *LBP-SP* compares $lb(U, d_4)$ and $L(U, d_3)$ before computing $OGRoutes_{d_4}$. Due to $lb(U, d_4) = 10 \geq L(U, d_3)$, d_4 and the rest points (d_5 and d_6) in $List_s$ can not be the optimal meeting point and the algorithm terminates and returns $OGRoute_{d_3}$ as the OSR-G result.

6 Approximate Points Selection: An Approximate Algorithm

Although the proposed *LBP-SP* algorithm can prune more unpromising candidate POIs than *CF*, it still requires a lot of time to do the OSR queries for the candidate meeting POIs which is inefficient for large datasets. The main issue is that the main cost of LBP algorithm heavily depends on multiple times' performing the existing OSR (i.e., E-OSR) algorithm. Inspired by the *minimum distance* (namely *MD*) algorithm [14] for approximately addressing the TPQ query, which can be generalized to OSR, we develop *APS* algorithm with θ -approximate (the approximate ratio is not greater than θ, see *Definition* 6 introduced next) solution for OSR-G problem by utilizing MD to replace E-OSR.

We first define the Δ-approximate OSR-G problem as follows.

Definition 6 (*θ-approximate OSR-G problem*) Given an OSR-G query Q and a scalar θ where $\theta > 1$, suppose the cost of the optimal solution S_{opt} for Q is $L(S_{opt})$. The *θ-approximate OSR-G problem* aims to find an approximate solution S_{appr} such that $\theta \geq \frac{L(S_{appr})}{L(S_{opt})}$, where $L(S_{appr})$ is the cost of S_{appr} and $\frac{L(S_{appr})}{L(S_{opt})}$ is called the approximate ratio of S_{appr}.

Then we recall the MD algorithm to approximately process the OSR query. Given an OSR query where the user starts from q, the destination is d and a query sequence $C = \{c_1, c_2, \dots, c_m\}$. For every category $c_i \in C$, MD selects a point p_i such that $L(q, p_i) + L(p_i, d)$ is minimum among all POIs in the category c_i. Correspondingly, one POI in each category c_i is selected to form a feasible route $\overrightarrow{R}$=(q,p_1,p_2 ,...,p_m,d). Since MD provides a $(m + 1)$-approximate solution where m is the size of the query category, applying the MD algorithm on the OSR query is also with an approximation ratio not greater than $(m + 1)$.

For an OSR-G query, directly applying MD for the OSR query by all users in U results in a large approximation ratio. To make a trade-off between the approximation ratio and time, a natural idea is that when examining a candidate meeting POI, only some users in U perform the exact OSR queries and the rest of users perform an approximate OSR algorithm such as MD algorithm. We define the parameter μ as the percentage of users in U who perform exact OSR queries. Notice that the larger μ is, the slower the algorithm becomes and the smaller the approximation ratio is. In addition, in order to early terminate the algorithm and not to lose a guaranteed approximation ratio bound, we define a parameter λ as a termination parameter for *APS* algorithm where $1 < \lambda \leq \theta$. λ is greater than 1, which provides a loose termination condition for examining a candidate meeting point. Notice that the larger the λ is, the faster the algorithm finds a solution with a larger approximation ratio. In order to assure a good approximation ratio, we restrict λ not to be greater than θ. Following the examination order sorted by $lb(U, d)$, we check whether the cost of current solution is greater than or equal to λ times of the cost of the lower bound (i.e., $lb(U, d)$) of $OGRoutes_d$. If it is, the examination of the remaining POIs in $List_s$ is terminated. By *Lemma* 2, we known that *APS* algorithm offers an θ-approximate solution for an OSR-G query, where $\theta = \max_{C_{u_i} \in C\_U} \{|C_{u_i}|\} + 1$. The pseudo-code of APS algorithm is shown in algorithm 3.

```
Algorithm 3: APS ALGORITHM
Input: A group of users $U$, a meeting category $c_d$, $lb(u,d)$, $\lambda$, $\mu$;
Output: The APS result $FGRoutes_{APS}$;
1  $FGRoutes_{APS} \leftarrow null; //L(null) = \infty$
2  Compute $lb(U,d)$ by $lb(u,d)$ for all $d$ in $c_d$ and get a sorted list $List_s$;
3  for each $d$ in $List_s$ do
4      if $\lambda \times lb(U,d) \geq L(FGRoutes_{APS})$ then
5          break;
6      $y \leftarrow \lfloor \mu \times |U| \rfloor$;
7      for $u_i \leftarrow u_1$ to $u_y$ do
8          $\overrightarrow{routes[i]} \leftarrow$ E-OSR$(u_i, C_{u_i}, d)$;
9      for $u_i \leftarrow u_{y+1}$ to $u_n$ do
10         $//C_{u_i} = \{c_1^i, c_2^i, ..., c_{|C_{u_i}|}^i\}$
11         $\overrightarrow{routes[i]} \leftarrow MD(u_i, C_{u_i}, d)$;
12     if $L(FGRoutes_{APS}) > L(routes)$ then
13         $FGRoutes_{APS} \leftarrow routes$
14 return $FGRoutes_{APS}$
```

Lemma 2 *APS algorithm offers an θ-approximate solution for an OSR-G query.*

Proof Let $OGRoutes_d$ denote the OGRoutes ending at d. Then, $FGRoutes_d^{APS}$ denotes the FGRoutes ending at d found by APS algorithm and $L(OSR\text{-}G)$ denotes the cost of exact solution.

In order to prove $L(FGRoutes_d^{APS}) \leq \theta \times L(OSR\text{-}G)$, we first prove that $L(FGRoutes_d^{APS}) \leq \theta \times L(OGRoutes_d)$. When examining a candidate meeting POI d, assume users u_1 ,u_2,...,u_y (y is determined by μ where $y = \lfloor \mu \times |U| \rfloor$) do exact OSR queries by E-OSR algorithm while other users $u_{y+1}, u_{y+2}, \ldots, u_n$ perform an approximate OSR algorithm using MD algorithm. Let the query route of i-th user be $\overrightarrow{R_d^{u_i}}$ and the corresponding ORoute ending at d denote $\overrightarrow{ORoute_d^{u_i}}$. If $L(\overrightarrow{R_d^{u_x}})$ ($1 \leq x \leq y$) is maximum among all query routes, then we have $L(\overrightarrow{R_d^{u_x}}) = L(\overrightarrow{ORoute_d^{u_x}}) \leq L(OGRoutes_d) \leq \theta \times L(OGRoutes_d)$. Otherwise, suppose $L(\overrightarrow{R_d^{u_z}})$ is the maximum among all query routes ($y < z \leq n$). We have $L(\overrightarrow{R_d^{u_z}}) \leq (|C_{u_z}| + 1) \times L(\overrightarrow{ORoute_d^{u_z}})$[7] $\leq \theta \times L(\overrightarrow{ORoute_d^{u_z}}) \leq \theta \times L(OGRoutes_d)$. Thus, we have $L(FGRoutes_d^{APS}) \leq \theta \times L(OGRoutes_d)$.

Next, we prove $L(FGRoutes_d^{APS}) \leq \theta \times L(OSR\text{-}G)$. Before examining a candidate meeting POI d_i' in the sorted list $List_s$ ($List_s = \{d_1', d_2', \ldots, d_\xi'\}$), if we find $\lambda \times lb(U, d_i') \geq L(FGRoutes^{APS})$, we have found a FGRoutes $FGRoutes_{d_x'}^{APS}$ ($1 \leq x < i$) and $L(FGRoutes_{d_x'}^{APS})$ is minimum among current examined FGRoutes such that $L(FGRoutes_{d_x'}^{APS}) \leq \theta \times L(OSR\text{-}G)$. If the optimal meeting POI is d_k' ($1 \leq k < i$), we have $L(FGRoutes_{d_x'}^{APS}) \leq L(FGRoutes_{d_k'}^{APS}) \leq \theta \times L(OGRoutes_{d_k'}) \leq \theta \times L(OSR\text{-}G)$. Otherwise, supposing the optimal meeting POI is d_j' ($i \leq j \leq \xi$), then $L(FGRoutes_{d_x'}^{APS}) \leq \lambda \times lb(U, d_i') \leq \lambda \times lb(U, d_j') \leq \lambda \times L(OSR\text{-}G) \leq \theta \times L(OSR\text{-}G)$. As a result, we find an θ-approximate solution before examining d_i'.

In the worst case, all POIs in the sorted list $List_s$ are examined, we still have a θ-approximate solution. Supposing d_x' ($1 \leq x \leq \xi$) is the optimal meeting POI found by APS algorithm and the optimal meeting POI is d_y' ($1 \leq y \leq \xi$), we have $L(FGRoutes_{d_x'}^{APS}) \leq L(FGRoutes_{d_y'}^{APS}) \leq \theta \times L(OGRoutes_{d_y'}) \leq \theta \times L(OSR\text{-}G)$. □

6.1 Selection of λ and μ

Since the approximate OSR-G algorithm uses parameters λ and μ, selecting the appropriate values of λ and μ for a network dataset is important because larger λ can lead to faster termination of examination with higher approximate ratio while larger μ has lower approximate ratio with more time for the approximate algorithm to run.

So some trade-offs must be made in the selection of λ and μ. However, choosing the values of λ and μ is empirical and different people have different requirements of between the query time and the query quality. In general, we have two ways of selecting λ and μ. For a specific network, we should do sufficient OSR-G queries to decide the values of which values should be set in terms of the network size. One way is selecting μ first. Set λ to 1.0 and vary different μ values and choose the μ in terms of the time and the approximate ratio. If the query time is more concerned, select the value where the time curve tends to converge with acceptable approximate ratio. Or the query quality is more cared, select the

[7] It is deduced by the approximation ratio of MD algorithm.

value where the approximate ratio curve tends to converge with acceptable query time. After μ is set, fix the μ and vary different λ values and select the appropriate λ which satisfies the requirements in the similar way. Another way is selecting λ first. Set μ to 1.0 and vary different λs and select the value according to the query time and query quality preference. After λ is set, similar process can be done to select the appropriate value of μ.

7 Complexity Analysis

Here we show the time complexity analysis of our proposed OSR-G algorithms.

Let $|U|$ be the number of the users, $|c_d|$ denote the number of POIs which is in meeting category c_d, and $|V|$ be the number of vertices in the road network. Since OSR-G algorithms utilize OSR algorithm, we use $O(OSR)$ to denote the time complexity of the E-OSR algorithm.

For *OSRB* algorithm, every user in the group U performs an OSR query in every candidate meeting POI in c_d. Therefore, the time complexity of OSRB is $O(|U||c_d|O(OSR))$. For *LBP-SP* algorithm, suppose the pruning rate is r_{SP}. Then it is easy to obtain the time complexity of the *LBP-SP* algorithm is $O((1-r_{SP})|U||c_d|O(OSR))$ without considering the cost of computing the sorted list of the lower bounds.

Hence, pruning rate is a key factor contributing to the actual computational cost, which is to be evaluated experimentally in the next section.

For *CF* algorithm, assume the pruning rate is r_{CF}. *CF* algorithm firstly performs $|U|$ times shortest path algorithm for latter computation of $L(U,d)$ which takes $O(|U||V|log|V|)$ time complexity. Selecting the initial meeting point takes $O(c_d)$ time. The first time of performing greedy search requires $O(\sum_{i=1}^{|U|}(|C_{u_i}|+1)O(Greedy))$ ($O(Greedy)$ is the time complexity of the greedy neighbor search), while the second time of performing greedy search takes $O(|U||c_d|O(Greedy))$. The last stage of performing OSR algorithm takes $(1-r_{CF})\times|U||c_d|O(OSR)$. The filtering process of examining candidate POIs in c_d takes $O(r_{CF}|c_d|^2)$. In summary, the time complexity of *CF* algorithm is $O(|U||V|log|V|) + O(c_d) + O(\sum_{i=1}^{|U|}(|C_{u_i}|+1))O(Greedy) + O(|U||c_d|)O(Greedy) + (1-r_{CF})|U||c_d|O(OSR) + r_{CF}|c_d|^2$.

For *APS* algorithm, we denote $O(AOSR_{APS})$ as the time complexity of the approximate OSR algorithm used by *APS* algorithm and assume the pruning rate is r_{APS}[8] and the μ parameter used is μ_{APS}, the time complexity of *APS* algorithm is $(1-r_{APS})|U||c_d|(\mu_{APS}\times O(OSR)+(1-\mu_{APS})\times O(AOSR_{APS}))$.

[8] The λ parameter used is λ_{APS} not greater than θ, it affects r_{APS}.

Table 1 Road network datasets

Network	Vertices	Edges
DE	48,812	119,004
VT	95,672	209,288
ME	187,315	412,352
SNW	158,181	183,865
NW	1,089,933	2,545,844

Table 2 Experiment settings

Parameter	Values
$\|U\|$	**4**, 5, 6, 7, 8
$\|C\|$	3, **4**, 5, 6, 7, 8
λ	1.5, 2.0, 2.5, **3.0**, 3.5, 4.0
μ	0%, 12.5% ,25%, **50%**, 100%
$\|c_d\|$	10, 40, 200, 1000, 5000

8 Experiments

In this section, we investigate the performance evaluation of our proposed algorithms. All the algorithms are implemented in C++ and the experiments are conducted on a 2.10 GHz Intel Xreon CPU with 128 GB RAM, using both two real (SNW and NW) and three synthetic datasets (DE, VT and ME). These five network datasets are obtained from [1] listed in Table 1, where the POIs in real datasets are captured from OpenStreetMap.[9] Since DE, VT and ME datasets do not contain the POI information, we generate the POIs of 8 categories randomly with the same density 0.01.

We conduct the performance evaluation in two aspects: (1) the efficiency (including the running time and pruning rate) of OSR-G algorithms—we compare the running time and pruning rate[10] of the proposed OSR-G algorithms under various parameters); and (2) the effectiveness of the approximate OSR-G algorithm—we compare the approximation ratio of the approximate algorithm under various parameters. The parameters are summarized in Table 2, numbers in bold are the default settings except the $|c_d|$ parameter is done separately. Query user size $|U|$ denotes the number of users in an OSR-G query. Query category size $|C|$ denotes the number of categories for every user in an OSR-G query (this parameter of the approximate algorithm is 8). Tested algorithms include *OSRB*, *CF*, *LBP-SP* and *APS*. We randomly generate several groups of OSR-G queries by the parameters where each group of queries consists of 100 OSR-G queries (20 queries for NW dataset). In each experiment, we

[9] http://www.openstreetmap.org.

[10] Pruning rate is the ratio of meeting points that are not necessary to be examined.

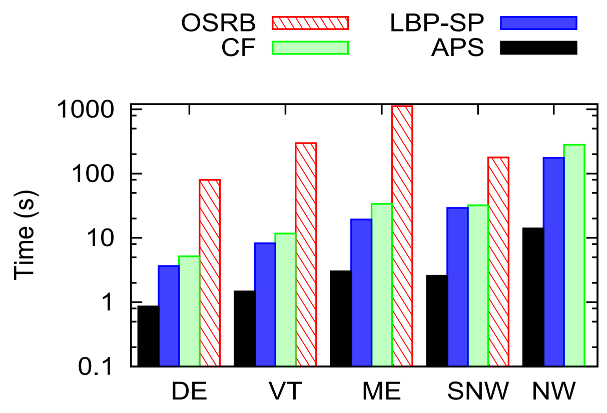


Fig. 3 Time versus five datasets

test one parameter at a time (other parameters are fixed at their default values). The reported experimental results are obtained by averaging the processing time of queries, pruning rates and approximation ratio. For space reasons, mostly we show only two figures per experiment.

8.1 Efficiency: Running Time

In this section, we evaluate the running time of the proposed OSR-G algorithms under various parameters.

Running time versus different datasets In this experiment, we first compare the running time of all the OSR-G algorithms on all datasets. Figure 3 shows that *CF* and *LBP-SP* algorithms outperform the *OSRB* algorithm and the approximate algorithm *APS* runs faster than all exact OSR-G algorithms. That is because *CF* can exploit the circle property to do filtering and *LBP-SP* can utilize the lower bound to prune the unpromising meeting points while *OSRB* needs to examine all the candidate meeting points to assure the optimality. *LBP-SP* outperforms *CF* mainly due to the higher pruning rate. As expected, *APS* is much more efficient than the exact algorithms mainly because *APS* takes larger λ than the exact algorithms (we can equally think the λ in the exact algorithms is set 1, while λ in approximate algorithms is set 3) which makes it have stronger pruning rate than the exact algorithms. Since the baseline algorithm run relatively too long, we do not report the corresponding results in the following experimental figures.

Running time versus query user size $|U|$. We investigate the running time of the OSR-G algorithms by varying different query user sizes in Fig. 4. As shown in Fig. 4a on ME dataset, the query time of all the OSR-G algorithms increases as the user size of OSR-G queries increases. This is because the larger the query user size is, the more query overhead will be spent on executing OSR algorithms or approximate OSR algorithms. Similar result can be found in SNW dataset from Fig. 4b.

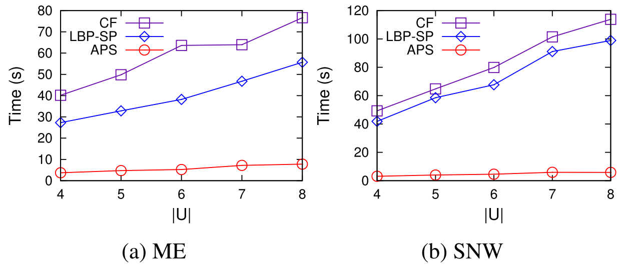


Fig. 4 Time versus $|U|$

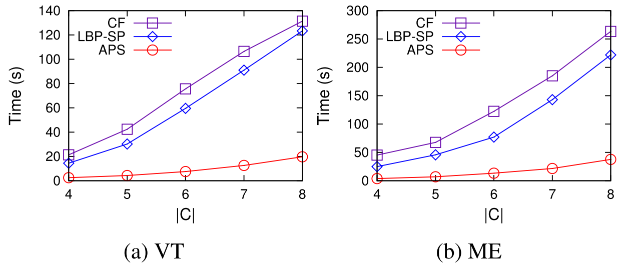


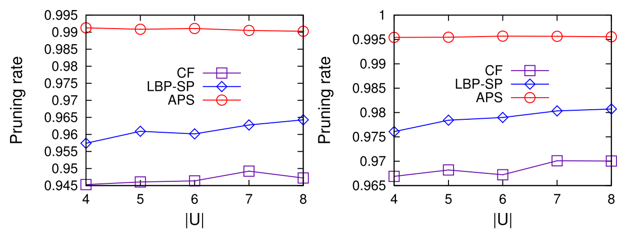
Fig. 5 Time versus $|C|$

Running time versus query category size $|C|$. Figure 5 shows the running time with respect to the query category size. The running time of OSR-G algorithms increases as the query category size increases. For VT dataset in Fig. 5a, when the query category size increases, the query sequence of a user becomes larger. This leads to increasing the time of executing an OSR query or an approximate OSR query for every user in the group. Therefore, the larger query category size costs more processing time of the OSR-G query. Figure 5b shows the similar results.

8.2 Efficiency: Pruning Rate

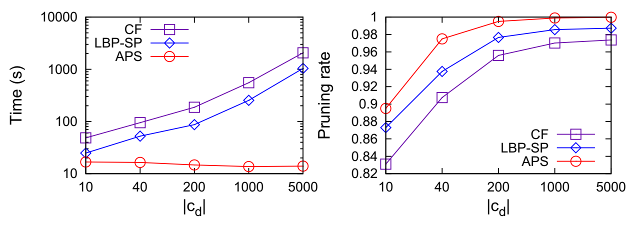
In this section, we evaluate another aspect about the efficiency: the pruning rate of the proposed algorithms under various parameters.

Pruning rate versus query user size $|U|$. We evaluate the pruning rate by varying query user size in Fig. 6. Figure 6b shows the pruning rates of all OSR-G algorithms remain relatively stable (no more than 1%) as the query user size increases. Recall that the meeting cost is determined by maximum cost of the last meeting user no matter how many users take part in an OSR-G query. Thus the pruning rate does not depend greatly on the query user size when the lower bound is tight enough. Note that the pruning rate of *LBP-SP* is higher than that of *CF* mainly for *LBP-SP* has good estimation ability which is faster to find the minimum 'radius' comparing to *CF*. Similar results can be seen from Fig. 6a.



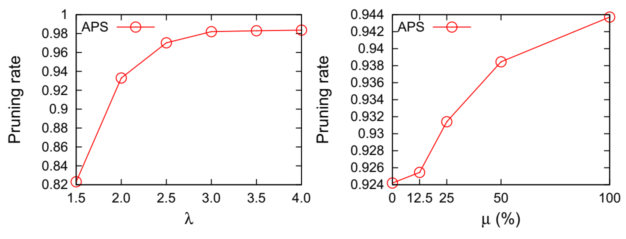
(a) VT (b) ME

Fig. 6 Pruning rate versus |*U*|



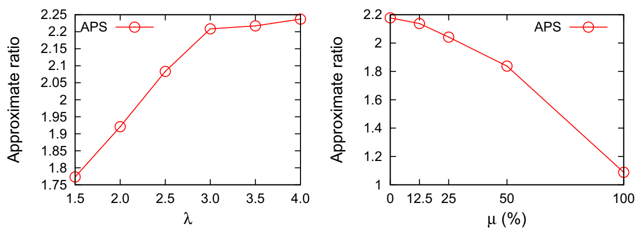
(a) Running time (b) Pruning rate

Fig. 9 Effect of $|c_d|$ on NW network



(a) DE (b) SNW

Fig. 7 Pruning rate versus λ and μ



(a) DE (b) SNW

Fig. 8 Approximate ratios versus λ and μ

Pruning rate versus λ and μ. Figure 7a shows the effect of varying λ on the pruning rates of the *APS* algorithm. In Fig. 7a, the pruning rate of *APS* increases dramatically as λ increases. This is because the larger λ is, the higher the pruning rate *APS* has. Similarly, we evaluate the pruning rate of *APS* by varying μ in Fig. 7b. The pruning rate of *APS* increases only about 2% by varying μ from 0% to 100% which is less significant comparing to increasing λ.

8.3 Effectiveness Evaluation

In this section, we evaluate the effectiveness of the approximate algorithm by varying λ and μ.

Approximation ratio versus λ. We evaluate the effect of varying λ on the approximation ratios of *APS* in Fig. 8a, the approximate ratio of *APS* increases greatly as λ increases. Since the pruning rate of *APS* increases greatly as λ increases, *APS* examines less points in the meeting category and returns an approximate solution more quickly with the increasing of the approximate ratio.

Approximation ratio versus μ. Then we evaluate the effect of varying μ on the approximation ratios of *APS* algorithm. As shown in Fig. 8b, the approximation ratios of *APS* algorithm decreases as μ increases. The approximate ratio of *APS* decreases greatly and converges to nearly 1.0 on SNW dataset when μ is 100%. Though the approximate ratio is mainly determined by the part of users who executing approximate OSR algorithms, as μ increases, the percentage of approximate OSRs decreases which results in better approximate solutions.

8.4 Evaluation of Effect on $|c_d|$

Since the size of meeting category (i.e., $|c_d|$) is a key factor for OSR-G queries, we evaluate the effect (i.e., query time and pruning rate) corresponding to the size of c_d. As shown in Fig. 9a, the query time of the exact algorithms is increasing as $|c_d|$ increases because they require to examine more candidate meeting POIs when increasing $|c_d|$. While for the approximate algorithm *APS*, the query time starts to decrease when $|c_d|$ becomes larger. Because the pruning rate of the approximate algorithms is quite high and is increasing, the number of actual examined candidate meeting points is decreasing. The query time of *APS* algorithm increases slightly when $|c_d|$ is 5000 because the number of actual examined candidate meeting points increases (here the pruning rate of *APS* algorithm tends to converge to a point. That is, the increase of $|c_d|$ is faster than that of the pruning rate). In Fig. 9b, the pruning rates of all algorithms are increasing because the larger the size of the meeting POI category becomes, the more unpromising points can be pruned by *CF*, *LBP-SP* and *APS* algorithms.

9 Conclusion

In this paper, we formulate a new query, namely, *optimal sequenced route for group meetup* (*OSR-G*) query, for finding the optimal meeting point such that all users meet as soon as possible after each user visits a number of POIs. We carry out a systematic study on the OSR-G query. First, we propose the *OSRB* algorithm as a baseline to tackle OSR-G problem. To address the shortcomings of *OSRB*, we propose an *upper bound based filtering* algorithm (i.e., *CF* algorithm) which utilizes the circle property to filter unpromising meeting POIs. In contrast, we design a *lower bound based pruning* (LBP) algorithm, *LBP-SP*, which exploits the shortest path lower bound to prune unqualified meeting points to have efficient query processing. Moreover, we prove that any lower bound of an OSR can be plugged in the LBP algorithms which makes it serve as a general framework to design different algorithms when using different lower bounds. Furthermore, an approximate OSR-G algorithm, namely *APS*, is developed to accelerate OSR-G queries in large network datasets. Finally, a comprehensive performance evaluation is conducted to validate the proposed ideas and demonstrate the efficiency and effectiveness of the proposed algorithms.

This work may lead towards several new directions for future work, e.g., top *k* OSR-G problem and more tighter lower bounds.

Acknowledgements This work is supported by National Natural Science Foundation of China (U1811264, U1911203, 61902438, 61902439), National Science Foundation for Post-Doctoral Scientists of China under Grant (2018M643307, 2019M663237), Natural Science Foundation of Guangdong Province under Grant (2019A1515011704, 2019A1515011159), Young Teacher Training Project of Sun Yat-sen University under Grant (19lgpy214, 19lgpy223) and Guangdong Basic and Applied Basic Research Foundation (2019B1515130001).

References

1. http://users.diag.uniroma1.it/challenge9/
2. Ahmadi E, Nascimento MA (2015) A mixed breadth-depth first search strategy for sequenced group trip planning queries. In: MDM. pp 24–33
3. Ahmadi E, Nascimento MA (2018) Optimal meeting points for public transit users. In: MDM. pp 15–23
4. Cao X, Chen L, Cong G, Xiao X (2012) Keyword-aware optimal route search. PVLDB 5(11):1136–1147
5. Chen H, Ku W, Sun M, Zimmermann R (2008) The multi-rule partial sequenced route query. In: ACM-GIS. p 10
6. Chen H, Ku W, Sun M, Zimmermann R (2011) The partial sequenced route query with traveling rules in road networks. GeoInformatica 15(3):541–569
7. Costa CF, Nascimento MA, de Macêdo JAF, Theodoridis Y, Pelekis N, Machado JC (2015) Optimal time-dependent sequenced route queries in road networks. In: SIGSPATIAL. pp 56:1–56:4
8. Dai J, Liu C, Xu J, Ding Z (2016) On personalized and sequenced route planning. World Wide Web 19(4):679–705
9. Hashem T, Barua S, Ali ME, Kulik L, Tanin E (2015) Efficient computation of trips with friends and families. In: CIKM. pp 931–940
10. Hashem T, Hashem T, Ali ME, Kulik L (2013) Group trip planning queries in spatial databases. In: SSTD. pp 259–276
11. Jahan R, Hashem T, Barua S (2017) Group trip scheduling (GTS) queries in spatial databases. In: EDBT. pp 390–401
12. Lettich F, Nascimento MA, Anwar S (2020) Trade-off aware sequenced routing queries (or OSR queries when pois are not free). In: 21st IEEE International conference on mobile data management, MDM 2020, Versailles, France, June 30 - July 3, 2020. pp 59–68. IEEE
13. Levin R, Kanza Y, Safra E, Sagiv Y (2010) Interactive route search in the presence of order constraints. Proc. VLDB Endow. 3(1):117–128
14. Li F, Cheng D, Hadjieleftheriou M, Kollios G, Teng S (2005) On trip planning queries in spatial databases. In: SSTD. pp 273–290
15. Li J, Yang YD, Mamoulis N (2013) Optimal route queries with arbitrary order constraints. TKDE 25(5):1097–1110
16. Li W, Zhu H, Liu W, Yin J, Xu J (2021) Optimal sequenced route query with poi preferences. In: Dasffa. p. Accepted
17. Liu H, Jin C, Yang B, Zhou A (2018) Finding top-k optimal sequenced routes. In: ICDE. pp 569–580
18. Ohsawa Y, Htoo H, Sonehara N, Sakauchi M (2012) Sequenced route query in road network distance based on incremental euclidean restriction. In: DEXA. pp 484–491
19. Papadias D, Zhang J, Mamoulis N, Tao Y (2003) Query processing in spatial network databases. In: VLDB. pp 802–813
20. Rice MN, Tsotras VJ (2013) Engineering generalized shortest path queries. In: ICDE. pp 949–960
21. Samrose S, Hashem T, Barua S, Ali ME, Uddin MH, Mahmud MI (2015) Efficient computation of group optimal sequenced routes in road networks. In: MDM. pp 122–127
22. Sasaki Y, Ishikawa Y, Fujiwara Y, Onizuka M (2018) Sequenced route query with semantic hierarchy. In: Böhlen, M.H., Pichler, R., May, N., Rahm, E., Wu, S., Hose, K. (eds.) Proceedings of the 21st international conference on extending database technology, EDBT 2018, Vienna, Austria, March 26-29, 2018. pp 37–48. OpenProceedings.org
23. Sharifzadeh M, Kolahdouzan MR, Shahabi C (2008) The optimal sequenced route query. VLDB 17(4):765–787
24. Sharifzadeh M, Shahabi C (2008) Processing optimal sequenced route queries using voronoi diagrams. GeoInformatica 12(4):411–433
25. Yan D, Zhao Z, Ng W (2011) Efficient algorithms for finding optimal meeting point on road networks. PVLDB 4(11):968–979
26. Yan D, Zhao Z, Ng W (2015) Efficient processing of optimal meeting point queries in euclidean space and road networks. Knowl Inf Syst 42(2):319–351