Activity-based Ride-sharing with constraint on travel distance

Student ID: 1F10170117
Student name: NGUYEN TUAN HAI Dang
Supervisor: ASANO Yasuhito

Falcuty of Information Networking for Innovation and Design,
Toyo University

Abstract

Ride-sharing is an environmentally friendly and cost-effective means of transportation. While existing research mainly focuses on trip-based ride-sharing, a new ride-sharing paradigm, activity-based ride-sharing, offers a clear advantage over the former paradigm. In this work, we add the travel distance constraint to the activity-based ride-sharing problem. Given a set of users and a set of points of interest, the objective is to find an optimal partition of users into multiple ride-sharing groups. Each group can fit into a car and users in each group do not travel a longer distance than they expected. Our problem is a version of the minimum weight set cover problem and an extension of the Minimum Steiner Tree problem, thus it is NP-hard. Later, we propose two versions of the solution, an exact solution and an approximation solution, based on dynamic programming. Our experiments on real road network data highlight the benefit of activity-based ride-sharing than other methods. Despite the complexity of the problem and the travel distance constraint, the approximation solution is fast, stable, and scalable, thus can be applied to the real-time ride-sharing system.

Contents

1 Introduction

Ride-sharing is a mix of public transports and private cars, thus it inherits the advantages of both methods. It provides numerous benefits to both individuals and society. As for individuals, ride-sharing reduces travel costs, travel time, and stress. While for society, it mitigates congestion, pollution, and conserves energy. All of these benefits arise from the principal objective of a ride-sharing system, which is to reduce the total travel cost. The total travel cost is the total travel distance for all vehicles participating in the system. For example, in Figure 1, two users u_1 and u_2 travel to p_3 by car. Without ride-sharing, the travel distance of each car is $2+4=6$, then the total travel cost is $6+6=12$. If u_1 and u_2 meet at q_1 and both users go to p_3 in u_1's car, the travel distance of u_1's car is $2+4=6$ while the travel distance of u_2's car is only 2. Then, the total travel cost in the latter case is $6+2=8$. In ride-sharing, because only one car goes from q_1 to p_3, energy consumption decreases, carbon emission also decreases. Due to its great advantages, ride-sharing draws a lot of attention to researchers. Trip-based ride-sharing, which means the destination must be fixed in advance, has been an active research field due to its potential application in existing ride-sharing systems. Well-known ride-hailing mobile apps, such as Uber or Lyft, are based on the trip-based ride-sharing paradigm. In these apps, users send requests consisting of the origin and the destination of their trips, then the system schedules proper drivers to pick up them. More recently trip-based ride-sharing approach which can receive and respond to requests in real-time is called real-time taxi-sharing. Many works [5, 6, 8] have been conducted to improve the efficiency of real-time taxi-sharing with a trip-based ride-sharing setting. Meanwhile, a new ride-sharing paradigm, activity-based ride-sharing, receives much less interest. In activity-based ride-sharing, people can select the destination from a set of points of interest (POIs) that serves the same purpose. It has been shown that activity-based ride-sharing increases the matching rate considerably compared to trip-based ride-sharing [10]. Besides, the choice of destination improves the efficiency of ride-sharing by reducing the total travel cost for all vehicles [3]. It also turns out that activity-based ride-sharing is close to many real-world scenarios. Several people in the same city, for instance, want to go to the park to have a picnic. Each person travels in a private car. Although there are several parks in the city, one naturally travels to his/her nearest park to minimize the travel distance. Imagine that a few people among them accept the extra travel distance and agree on a common park. Then, they can meet at a confluence point, leave every car except one, and ride-share to the chosen destination. One noticeable difference from the cases of with and without ride-sharing is that the destination cannot be pre-determined beforehand. Instead, the destination of a ride-sharing group should be chosen in such a way that the total travel cost of the whole group is minimized.

The idea in which multiple people meet at confluence points and ride-share, also known as confluent routing, shows several practical applications including ride-sharing, delivery routing, and pedestrian navigation [11]. Ride-sharing with confluent routing reduces the total travel cost of all vehicles in comparison with the greedy approach for each person, i.e., each person travels to his/her nearest destination. In a real-life situation, however, people only accept the extra travel distance to some extent, which is hereafter mentioned as the **extra ratio**. Therefore, it restricts the freedom of grouping people and the choice of a common destination. We will use the term **travel distance constraint** to describe the condition of accepting ride-sharing. Previous work on ride-sharing with confluent routing [3, 9] does not consider the constraint on travel distance, thus their proposals cannot be applied.

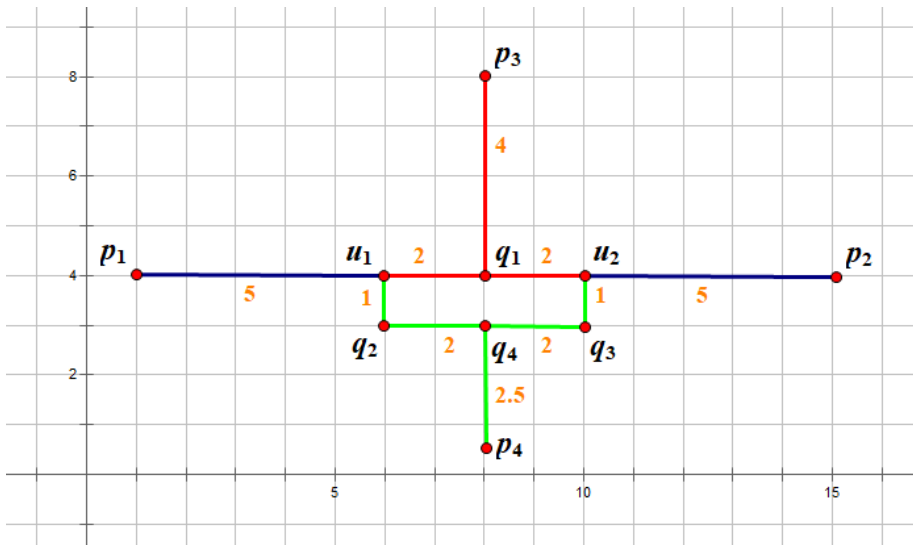


Figure 1: An example of the optimal travel plan that is obtained by the VST-RS method violates the travel distance constraint.

In Figure 1, there are two users u_1, u_2, and four POIs p_1, p_2, p_3, p_4. Assume that each user travels by car with 4 seats. When two or more users meet at a point, they leave every car there except one and travel together in that car. The nearest POIs to u_1 and u_2 are p_1 and p_2, respectively. Without ride-sharing, each user travel to the corresponding nearest POI (the greedy approach). In this example, the total travel cost of the greedy approach is $u_1p_1 + u_2p_2 = 5+5 = 10$. Assume that the extra ratio is 0.1 for both users. In the presence of ride-sharing, the maximum travel distance that one can accept is $(1 + 0.1) * 5 = 5.5$. In this case, the VST-RS method [3] gives the optimal travel plan as follows: u_1 and u_2 meet at q_1, then travel together to p_3. The total travel cost is only $2+2+4 = 8$. However, each user has to travel a distance of $2 + 4 = 6 > 5.5$, violates the travel distance constraint. The optimal travel plan satisfying the travel distance constraint is as follows: u_1 and u_2 meet at q_4, then ride-share to p_4. The total travel cost is $3+3+2.5 = 8.5$, which is slightly higher than the one obtained from the VST-RS method but still smaller than the greedy approach's one. In addition, the travel distance of each user is only $3 + 2.5 = 5.5$, which satisfies the travel distance constraint. The example illustrates the fact that the travel plan that minimizes the total travel cost may not guarantee the travel distance constraint. This is usually the case in a real-life situation when people find their ride-sharing partners. Therefore, we need to improve the VST-RS method to solve the problem of activity-based ride-sharing with a travel distance constraint. The main contributions of our work are as follows.

- We formulate the problem of activity-based ride-sharing with a constraint on travel distance. To the best of our knowledge, this is the first work to consider the travel distance constraint in activity-based ride-sharing.
- We propose an exact solution and an approximation solution to the problem.
- Our experiments on real road network data show that ride-sharing reduces the total travel costs. Also, having a choice set of destination improve the efficiency of ride-sharing in comparison with the pre-defined destination. Furthermore, the approximation solution is fast enough to be employed in the real-time ride-sharing system.

2 Related work

2.1 Activity-based Ride-sharing

Activity-based ride-sharing allows users to choose the destination among a set of choices, thus increases the flexibility of matching. In 2016, Wang et al. [10] proposed a new ride-sharing paradigm, activity-based ride-sharing. Their objective was to create users' daily schedules with chains of activities to maximize the matching rate. They provided an approximation solution based on an efficient space-time filter. They also proved that global optimization can be achieved by binary linear programming. There are two main differences between Wang's activity-based ride-sharing problem and ours. Our problem does not consider the chain of activities and its objective is to minimize the total travel cost. Later, Khan et al. [3] showed that the activity-based ride-sharing problem can be solved efficiently as a new Steiner tree problem via dynamic programming. However, they did not consider the travel distance constraints of users, which is solved efficiently in our problem.

2.2 Ride-sharing with confluent routing

In ride-sharing with confluent routing, multiple users meet at confluence points along the way then ride-share to the common destination. There are no roles of servers and clients in ride-sharing with confluent routing; therefore, this method prefers private cars to taxis. Takise et al. [9] formulated the problem of ride-sharing with confluent routing and proposed an exact solution in the case of multi-user and a single destination. Later, Khan et al. [3] considered the setting for multi-user and multi-destination. Based on a heuristic, they divided the set of users into multiple equally sized groups to improve the computation speed. Our solution employs their Variable Steiner Tree Ride-sharing (VST-RS) method but modifies the division step. Furthermore, we add the users' constraint on travel distance to make the problem more practical.

2.3 Minimum Steiner Tree (MST)

Minimum Steiner Tree is the problem of finding a tree connecting all terminals in the graph with the lowest cost. In our problem, terminals are locations of people in the same group and their common destination while Steiner nodes are confluence points. An MST is the optimal travel plan for people in the group to the common destination. A difference in the setting of our problem from the MST problem is that the common destination is not known in advance but determined by the algorithm. The MST problem is NP-hard. Dreyfus and Wagner [2] developed a fast solution which has been used as a baseline till now. Our solution resembles their solution but improves the computation speed by limiting it to a smaller set of Steiner nodes instead of the whole set of nodes in the graph.

3 Problem definition

3.1 Assumptions

For the sake of simplicity, there are the following two assumptions. Firstly, the locations of users and POIs are at the nodes of the graph. In real-life, the nodes are intersections of multiple roads while the locations usually sit somewhere in the middle of the road. If we move the locations to the intersections, it will not affect the travel plan much. Secondly, we assume that each user has a car and the number of seats is the same (4 seats) for all passenger cars. In the problem of activity-based ride-sharing, the means of transport is private cars. In order to assign each car the same role in the problem, we simply consider that each car allows up to 4 users.

3.2 Notations and Definitions

Here, we introduce some notations and definitions that will be used in the problem and the solution. Let the road network be an undirected graph $G = (V, E)$, where the nodes V represent locations, and the edges E represent road segments. Let U and P be the set of users and POIs, respectively. Let $d(s, t)$ be the shortest distance between two nodes s and t. Given a set of nodes $D \subset V$, $Cost(D)$ is the sum of all edges of the MST that spans D. We denote N_v the nearest POI of node v if the following conditions are satisfied $v \in V, N_v \in P$ and $d(v, N_v) \coloneqq \min_{p \in P} d(v, p)$. For each user $u \in U$, the starting location $u.s$ and the extra ratio $u.\epsilon$ is known in advance. Thus, the maximum travel distance that u can accept is $(1+u.\epsilon)d(u, N_u)$. Let z be the number of seats in each passenger car. We use the symbol U' to denote a group of users that can fit into a car, i.e., $U' \subset U, |U'| \leq z$. For each user $u \in U'$

and POI $p \in P$, let $c_{U'}(u, p)$ be the travel distance of user u from the starting location to p when u travels together with other users in U' to p. We denote $P_{U'}$ as the subset of P that satisfies the travel distance constraints for all users in U'. Thus, the following inequality holds: $c_{U'}(u, p) \leq (1 + u.\epsilon)d(u, N_u) \forall u \in U', p \in P_{U'}$. The optimal POI is defined as the POI that satisfies the travel distance constraints with the least total travel cost for the ride-sharing group. The optimal travel cost of group U' is the total travel distance for all users' cars in U' to the optimal POI. We denote $OptPOI(U')$ and $OptCost(U')$ as the optimal POI and the optimal travel cost of U'. In other words, we have the following two equations, $OptPOI(U') := \arg\min_{p \in P_{U'}} Cost(U' \cup \{p\}), OptCost(U') := \min_{p \in P_{U'}} Cost(U' \cup \{p\})$. The optimal travel cost of U' is different from the sum of the travel distance of all users in group U'. Because there are few edges that multiple users share cars along those edges. Therefore, those edges are counted multiple times in the sum of the travel distance of all users while they are counted only once in the optimal travel cost of the whole group U'. Finally, a meeting point is a point where two or more users ride-share to the common destination. It is worth mentioning that the meeting point we used here has a broader meaning than a confluence point. A confluent point is a point when two or more users meet for the first time and start to ride-share. Every node in the road network that they travel together from the confluent point to the common destination is considered as the meeting point. With the aforementioned notations and definitions, our problem is formulated as follows.

3.3 Problem

Input

- Road network $G(V, E)$.
- Set of users U and POIs P
- Pair-wise shortest distance between two nodes in V
- The nearest POI to each node in V

Output

- Partition of U into t groups U_i such that U_i can fit into a car, i.e., $|U_i| \leq z(1 \leq i \leq t)$
- For each U_i, the optimal POI and MST (travel plan)

Objective

- Minimize $\sum_{i=1}^{t} OptCost(U_i)$

Constraint

- Travel distance constraint $c_{U_i}(u, OptPOI(U_i)) \leq (1 + u.\epsilon)d(u, N_u) \forall u \in U_i, 1 \leq i \leq t$

3.4 Complexity analysis

Our problem can be interpreted as the minimum weight set cover problem. The minimum weight set cover problem can be stated as follows [4]. There are a universal set $\mathcal{U}$ and a collection of m subsets $\mathcal{S} = \{S_i\}_{1 \leq i \leq m}$ where $S_i \subseteq \mathcal{U}$ and $\bigcup_{1 \leq i \leq m} S_i = \mathcal{U}$. Each subset S_i has a positive weight w_i. The task is to find a subset $\mathcal{R} \subseteq \{1, 2, \ldots, m\}$ that minimizes $\sum_{i \in \mathcal{R}} w_i$ subject to $\bigcup_{i \in \mathcal{R}} S_i = \mathcal{U}$. In our problem, the universal set is the set of all users (U) and the collection of subsets is the collection of all possible subsets of less than or equal to z users. The weight of a subset is the optimal travel cost of that subset. Because the minimum weight set cover problem is NP-hard [4], so is our problem. Furthermore, finding the optimal travel cost of a subset requires solving the MST problem, which is also NP-hard.

4 Solution

4.1 Outline of the solution

In this section, we propose two versions of the solution, an exact solution and an approximation solution. The approximation solution has two steps. In the first step, the set of all users is divided into small groups. Each group has the same number of users, denoted as S (the last group may have less than S users). A user only ride-share with other users in the same group. In the second step, we solve the optimal partition problem for each divided group in the first step. Because the time complexity of the algorithms in the second step increases exponentially by the number of users. By dividing into smaller groups in the first step, the second step runs much faster. The second step includes two sub-stages. The first sub-stage computes the optimal travel plan for all possible combinations of less than or equal z users. The second sub-stage finds the optimal division of an S-user group into subgroups so that each subgroup has at most z users and the total travel cost of all subgroups is minimized. As for the exact solution, also referred to as the optimal solution in the Experiments section, it includes only the second step of the approximation solution. In other words, the exact solution solves the optimal partition problem for the whole set of users (U).

4.2 Algorithms

4.2.1 Step 1

The first step of our solution do the same task as the first step in the VST-RS method. The first step in the VST-RS method is summarized as follows [3]. The target was to divide all users into S-user groups. First, they divided users into Voronoi cells with respect to their nearest POIs. Because, a cell may contain less than or more than S users, they split or merged cells into a group that has exactly S users. The author did not mention convincing reasons for using Voronoi cells. In addition, users may ride-share to a POI which is different from the nearest POI of their group (the generator point of Voronoi cell). Therefore, the nearest POI of each group does not play any role in the latter step of the VST-RS method. On the other hand, the first step can be interpreted as the problem of clustering into the same size clusters. In clustering into the same size clusters, we cluster multiple points into groups of similar size while optimizing the quality of the clusters. In [7], the nearest neighbor approach (maxD variant) showed a good performance in terms of several metrics. Although the nearest neighbor approach has a bottleneck in computing the pair-wise shortest distance, we utilize the precompute pair-wise shortest distance in the input of our problem. Using the nearest neighbor approach (maxD variant), the first step of our solution is summarized as follows.

1. Select a user, which has the largest total distance to the others, as the center user.
2. Find the $S - 1$ closest users to the center user chosen in 1.

3. The center user and $S-1$ users selected in 2 form a group. Remove this group from the set of all users.

4. Repeat 1-3 until there are no users to form a new group.

Algorithm 1: Find nearest neighbors.

```
Input: users $\boldsymbol{U}$, center user $\boldsymbol{s}$, number of neighbors $\boldsymbol{k}$.
1 create node set $Q$
2 foreach node $u$ in $U$ do
3     $Q$.insert($u, d(s,u)$)
4 $T \leftarrow \emptyset$
5 for $i = 1 \rightarrow k$ do
6     $u, distance \leftarrow Q$.ExtractMin()
7     $T \leftarrow T \cup u$
Output: nearest neighbors $\boldsymbol{T}$.
```

Algorithm 1 finds the k nearest neighbors of the center user (including the center user itself). The task of finding the k nearest neighbor can be simply done through two steps: first sort the list of users in the ascending order by their distances to the center user, then select the first k points from the list. The time complexity of this approach is $O(n \log n)$ where n is the number of users. This approach is not efficient when the number of users is much larger than the required number of neighbors. This is the usual scenario in the real-time ride-sharing system in which there are a lot of participants. To make the system real-time, we need to divide the set of all users into small-sized groups so that the system runs fast enough. Therefore, Algorithm 1 employs another approach using the priority queue implemented using the binary heap. The loop in lines 2-3 builds a binary heap of users with the shortest distance from each user to the center user as the priority of that user. The loop in lines 5-7 finds the k nearest neighbor to the center user (including the center user).

Algorithm 2 divides all users into equal-size groups based on the nearest neighbor approach mentioned above. In line 2, the set of remaining users X is first set to the set of all users U. Because the group size S is determined, the number of groups is also determined. The while loop in line 4 repeats until all groups are formed. The for loop in lines 7-13 calculates the total distance from each user to the others, then finds the center user which has the largest sum. Because the number of users $|U|$ may not be divisible by

Algorithm 2: Divide all users into groups of equal size.

```
Input: users $\boldsymbol{U}$, group size $\boldsymbol{S}$.
1  $nCluster \leftarrow \left\lceil \frac{|U|}{S} \right\rceil$
2  $X \leftarrow U$
3  $label \leftarrow 0$
4  while $label < nCluster$ do
5      $center \leftarrow null$
6      $maxDist \leftarrow 0$
7      foreach $u$ in $X$ do
8          $totalDist[u] \leftarrow 0$
9          foreach $v$ in $X$ do
10             $totalDist[u] \leftarrow totalDist[u] + d(u, v)$
11         if $totalDist[u] > maxDist$ then
12             $center \leftarrow u$
13             $maxDist \leftarrow totalDist[u]$
14     if $|X| > S$ then
15         $k \leftarrow S$
16     else
17         $k \leftarrow |X|$
18     $groups[label] \leftarrow FindNearestNeigbors(X, center, k)$
19     $X \leftarrow X \setminus groups[label]$
20     $label \leftarrow label + 1$
Output: travel groups $\boldsymbol{groups}$, number of groups $\boldsymbol{nCluster}$.
```

the group size S, the number of neighbors is set to S for all groups except for the last group. In the last group, the number of users is set to the number of remaining users which can be less than S. Line 18 finds the k nearest neighbors of the center user using Algorithm 1. Line 19 excludes the group found in line 18 from the set of remaining users.

4.2.2 Step 2

The first sub-stage of the second step is designed based on dynamic programming which is computed in the bottom-up fashion. Before going into the pseudo-code of the first sub-stage, we introduce an algorithm to find meeting points. In the first sub-stage, finding meeting points is an essential task to obtain the optimal travel plan for a group of users. The meeting point plays the role of the intermediate node or Steiner node in the MST problem. In the Dreyfus-Wagner algorithm [2], every node in the graph is taken into account, this is the source of inefficiency. Khan et al. [3] proposed two constraints on meeting points, thus they reduced the set of potential meeting points to the set of all nodes to which the travel cost of a subset of users is less than or equal to the optimal travel cost of that subset. In other words, given a subset of users U', the set of potential meeting points of U' is defined as $X := \{m \in V | Cost(U' \cup \{m\}) \leq OptCost(U')\}$. Note that when the graph is large, the set of nodes satisfying two constraints is much smaller than the set of all nodes in the graph. Therefore, the VST-RS method is more efficient than the Dreyfus-Wagner algorithm. So, the question is "How to find meeting points?". Here, we utilize the idea of Dijkstra's algorithm to design an algorithm to find all potential meeting points. In Dijkstra's algorithm, given a source and destination node, the algorithm examines all nodes of which the distance to the source node is less than or equal to that of the destination. In our solution, a subset of users plays the role of the source node while the optimal POI plays the role of the destination node. In addition, the shortest distance from the source node to a node in the graph is replaced by the total travel cost of the subset of users to that node.

Algorithm 3 resembles the above-mentioned idea. However, the following three points should be considered. Firstly, because the optimal POI is not known in advance, neither is the destination node. Algorithm 3 searches until one POI is found. The found POI is the optimal POI because the algorithm searches in the ascending order of the total travel cost. Secondly, the source is a set of possible confluent points instead of a set of users. Because Algorithm 3 will be used inside a dynamic programming algorithm, when finding the optimal travel plan for a group of users, we partition it into two subsets and find the set of meeting points for two subsets. Two sets of meeting points for two subsets have already been computed in the corresponding sub-problem. Then, the set of possible confluent points for two subsets must be the intersection of these two sets. We find the set of possible meeting points starting from the set of possible confluent points. Thirdly, Algorithm 3 needs to check the travel distance constraint for meeting points. A travel plan of a group of users through a meeting point, which satisfies two constraints of meeting points, may violate the travel distance constraint of one user inside the group. For instance, consider a simple case of a group that includes only one user u_1 in Figure 1. The set of nodes that satisfy two constraints of meeting points is $\{q_2, q_1, q_4, u_2, p_1\}$. If the extra ratio is once again set to 0.1, travel plans through q_1 (to p_3) and u_2 (to p_2) violate the travel distance constraint.

Algorithm 3: Find the potential meeting points.

```
Input: confluent points $\boldsymbol{M}$, users $\boldsymbol{comb}$, travel costs $\boldsymbol{cost}$, user costs
       $\boldsymbol{travelDist}$, POIs $\boldsymbol{P}$.
1  create node set $Q$
2  foreach $node\ m\ in\ M$ do
3      $Q$.insert$(m, cost[m])$
4      $J[(m, cost[m])] \leftarrow m$
5      mark $m$ visited
6  $X \leftarrow \emptyset$
7  $prevCost \leftarrow -1$
8  $prevNodes \leftarrow \emptyset$
9  $p \leftarrow null$
10 $minCost \leftarrow null$
11 while $Q\ is\ not\ empty$ do
12     $u, currCost \leftarrow Q$.ExtractMin()                ▷ find the next node
13     if $currCost > prevCost$ then
14         if $prevCost \neq -1$ then
15             $X \leftarrow X \cup prevNodes$
16         $prevNodes \leftarrow \{u\}$
17     else
18         $prevNodes \leftarrow prevNodes \cup \{u\}$
19     $prevCost \leftarrow currCost$
20     if $u \in P$ then
21         $X \leftarrow X \cup \{u\}$        ▷ discard non-POI nodes with the same costs
22         $p \leftarrow u$
23         $minCost \leftarrow currCost$
24         break                        ▷ terminate if a boundary node is found
25     foreach $neighbor\ v\ of\ u\ and\ v \notin \{X \cup prevNodes\}$ do
26         if $v\ not\ visited\ before$ then
27             $cost[v] \leftarrow inf$
28             mark $v$ visited
29         $temp \leftarrow cost[u] + d(u, v)$
30         if $cost[v] > temp$ then
31             $violation \leftarrow False$
32             for $c \in comb$ do
33                 $travelDist[(c, v, temp)] \leftarrow travelDist[(c, u, currCost)] + d(u, v)$
34                 if $travelDist[(c, v, temp)] + d(v, N[v]) > (1 + \epsilon)d(c, N(c))$ then
35                     $violation \leftarrow True$            ▷ violate travel distance
                        constraint
36                     break
37             if $not\ violation$ then
38                 $cost[v] \leftarrow temp$
39                 $J[(v, cost[v])] \leftarrow J[(u, currCost)]$
40                 $Q$.insert$(v, cost[v])$
Output: meeting points $\boldsymbol{X}$, travel costs $\boldsymbol{cost}$, user costs $\boldsymbol{travelDist}$,
        confluent points $\boldsymbol{J}$, optimal POI $\boldsymbol{p}$, optimal travel cost $\boldsymbol{minCost}$.
```

Algorithm 3, which we named $FindMP$, finds all possible meeting points for a group of users. Again, a priority queue is utilized to store nodes, this is the same as the efficient implementation of Dijkstra's algorithm. The loop in lines 2-5 does three operations: inserts each confluent point into the queue, stores the information of the confluent point with the associated travel cost, and marks the nodes visited. The while loop in line 11 repeats until the queue is empty. Inside the loop, the algorithm pops the node u with the highest priority (lowest travel cost). In lines 20-24, if the node is a POI, assigns the appropriate values for the optimal POI and optimal travel cost, then terminates the loop. Otherwise, the algorithm loops through each adjacent node v of the popped node u. If a lower travel cost to v is obtained via u, proceeds to check the travel distance constraint of v in lines 31-36. To check the travel distance constraint of a node, the algorithm loops through each user and compares his/her travel distance to the nearest POI of this node to his/her maximum travel distance. If node v satisfies the travel distance constraint, updates the information of travel cost and the confluent point of v, then inserts it into the queue.

It is worth mentioning that the travel cost is necessary to identify the confluent point and the travel distance of a meeting point as there are multiple routes to a meeting point via different confluent points. The travel cost is calculated for a group of users, a lower travel cost of the whole group does not indicate a lower travel distance for every user in the group. Therefore, there is a possibility that a travel plan via a meeting point with a higher travel cost satisfies the travel distance constraint while another travel plan via the same meeting point with a lower travel cost violates the travel distance constraint. For instance, Figure 2 demonstrates a graph, which has three users $u_1 u_2, u_3$ and two POIs p_1, p_2. Assume that each user travels by car with 4 seats. When two or more users meet at a point, they leave every car there except one and travel together in that car. Besides, the extra ratio is set to 0.5 for all users. Consider the state where two subgroups $\{u_1, u_2\}$ and $\{u_3\}$ meet at q_3. It is obvious that u_3 travels directly to q_3 along the edge $u_3 q_3$. As for the subgroup $\{u_1, u_2\}$, they need to travel to q_3 via q_2. There are two possible travel plans to q_2 for $\{u_1, u_2\}$. In the travel plan with a lower travel cost, u_1 and u_2 meet at q_1, then ride-share to q_2. The total travel cost is $2.68 + 2.96 + 1.77 = 7.41$. In the travel plan with a higher travel cost, u_1 and u_2 meet at q_2 directly. The total travel cost is $2.68 + 1.77 + 3.22 = 7.67$.

Note that the maximum travel distance of u_2 is $7.03 * 1.5 = 10.545$. It is easy to check that both travel plans to q_2 satisfy the travel distance constraint of u_2. Moving to q_3, there are two corresponding possible travel plans for $\{u_1, u_2\}$. As for the travel plan with a lower travel cost, the travel distance of u_2 is $2.96 + 1.77 + 1.92 + 3.92 = 10.57 > 10.545$, which violates the travel distance constraint. While in the travel plan with a higher travel cost, the travel distance of u_2 is $3.22 + 1.92 + 3.92 = 9.06 < 10.545$, which satisfies the travel distance constraint. If at meeting point q_2, the travel plan with higher cost was discarded, q_3 would not be a meeting point of $\{u_1, u_2\}$. If this case happened, in the optimal travel plan for the group of three users $\{u_1, u_2, u_3\}$, u_3 would travel to p_2 via q_3 while u_1 and u_2 would meet at q_1, then travel together to p_2 via q_2, q_3. The total travel cost is $3.30 + 3.92 + 7.41 + 5.44 = 20.07$. However, the actual travel plan obtained from Algorithm 3 has a lower travel cost. In the obtained travel plan, u_3 still travels to p_2 via q_3 but u_1 and u_2 meet at q_2 instead, then travel together to p_2 via q_3. The total travel cost is $3.30 + 3.92 + 7.67 + 1.92 = 16.81$. Therefore, it is important to store every legit travel plan to a meeting point. Storing only the travel plan with the lowest cost at each meeting point may yield problems in the next meeting points.

Algorithm 4 computes optimal travel plans for all possible groups of less than z users using dynamic programming in the bottom-up fashion. The for loop in lines 1-2 computes the travel plans when the group size is 1. In the case of one user, the confluent point is solely the location of that user and the information of division and subgroup is unnecessary. Line 3 loops through every group size in ascending order. For each group size, the for loop in line 4-30 loops through every possible combination (subset of users) *comb* of that size. Lines 5-9 initiates some variables to their default values, these variables will be computed later. For each combination, the loop in lines 10-25 considers all possible divisions into 2 subgroups $comb_1$ and $comb_2$. Lines 11-14 computes the best travel plan in case that two subgroups will not ride-share. If the total travel cost of two subgroups $comb_1$ and $comb_2$ is less than the current optimal travel cost of the group *comb*, updates the information of the optimal travel cost and subgroup in lines 13-14. Lines 15-25 computes the set of possible confluent points, the travel cost, and the travel distance for the group *comb* in the current division. In line 15, the set of possible confluent points is set to the intersection of

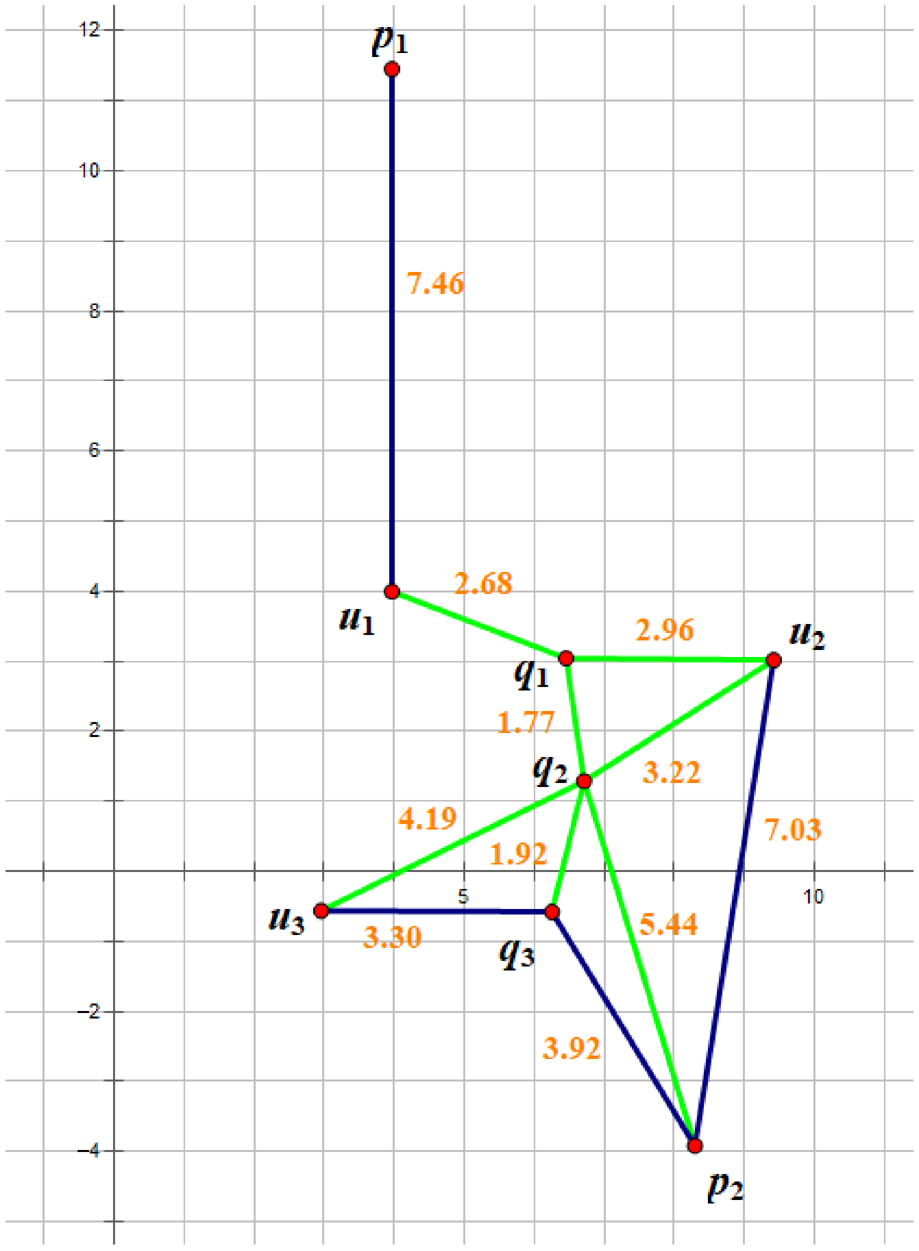


Figure 2: An example of a meeting point with two travel plans: the higher travel cost satisfies the travel distance but the lower one.

two sets of meeting points of two subgroups. The loop in lines 16-25 loops through every possible confluent point. Inside each loop, the travel cost of *comb* to the confluent point is compared to the total travel cost of two subgroups $comb_1$ and $comb_2$. If the current travel cost of the whole group *comb* is larger than the sum of two subgroups, updates the information of the travel cost, division, and the travel distance of each user in lines 20-25. Because two subgroups are disjoint, two for loops in lines 22-23 and lines 24-25

Algorithm 4: Compute optimal travel plans for all possible combinations of less than or equal z users.

```
   Input: users $\boldsymbol{U'}$, POIs $\boldsymbol{P}$.
1  foreach $user\ u \in U'$ do
2      $X[u], cost[u], travelDist[u], J[u], OptPOI[u], OptCost[u] \leftarrow$
         $FindMP(u, u, 0, 0, P, G)$
3  for $j = 2 \rightarrow z$ do
4      foreach $combination\ comb\ of\ j\ users\ from\ U'$ do
5          $M[comb] \leftarrow \emptyset$
6          $OptCost[comb] \leftarrow inf$
7          $OptPOI[comb] \leftarrow null$
8          foreach $v \in V$ do
9              $cost[comb][v] \leftarrow inf$
10         foreach $division\ of\ comb\ to\ comb_1\ and\ comb_2$ do
11             $temp \leftarrow OptCost[comb_1] + OptCost[comb_2]$
12             if $OptCost[comb] > temp$ then
13                 $OptCost[comb] \leftarrow temp$
14                 $bestSubgroupping[comb] \leftarrow [comb_1, comb_2]$
15             $XX \leftarrow X[comb_1] \cap X[comb_2]$
16             foreach $m \in XX$ do
17                 $M[comb] \leftarrow M[comb] \cup m$
18                 $temp = cost[comb_1][m] + cost[comb_2][m]$
19                 if $cost[comb][m] > temp$ then
20                     $cost[comb][m] \leftarrow temp$
21                     $bestDiv[comb][m] \leftarrow [comb_1, comb_2]$
22                     for $c \in comb_1$ do
23                         $travelDist[comb][(c, m, temp)] \leftarrow$
                             $travelDist[comb_1][(c, m, cost[comb_1][m])]$
24                     for $c \in comb_2$ do
25                         $travelDist[comb][(c, m, temp)] \leftarrow$
                             $travelDist[comb_2][(c, m, cost[comb_2][m])]$
26         $X[comb], cost[comb], travelDist[comb], J[comb], p, temp \leftarrow$
             $FindMP(M[comb], comb, cost[comb], travelDist[comb], P, G)$
27         if $temp\ is\ not\ null\ and\ OptCost[comb] > temp$ then
28             $OptCost[comb] \leftarrow temp$
29             $OptPOI[comb] \leftarrow p$
30             $bestSubgroupping[comb] \leftarrow [comb, null]$
   Output: optimal travel costs $\boldsymbol{OptCost}$, optimal POIs $\boldsymbol{OptPOI}$, travel plan
           $\{\boldsymbol{J}, \boldsymbol{bestDiv}, \boldsymbol{bestSubgroupping}\}$.
```

have no users in common. After looping through every division of the group *comb*, the algorithm has enough inputs to run Algorithm 3 for the group *comb*. If the optimal travel cost computed for the whole group *comb* is less than the previous computed optimal travel cost in the case of division, updates the information of the optimal travel cost, optimal POI, and subgroup in lines 28-30. The algorithm ends when optimal travel plans for all possible combinations of less than z users are calculated.

The second sub-stage computes the optimal partition for each S-user group, which is obtained in the first step. Dynamic programming is a suitable approach to utilize the optimal travel plans computed in Algorithm 4 to find the optimal travel plans when the group size increases from $z + 1$ to S. For example, when the group size is $z + 1$, the whole group does not fit into a car. It is necessary to consider all possible partitions of the group into two subgroups. For each partition, because both two subgroups have less than or equal to z users, the optimal travel plans for the two subgroups have already been calculated. Thus, the optimal travel cost of the group in each partition is obtained by taking the sum of the optimal travel costs of two subgroups. The partition with the lowest total travel cost of two subgroups is the optimal partition. By this method, the optimal partition for every group of size $z + 1$ is computed. Similarly, the algorithm finds the optimal partitions in cases of group size $z + 2, \ldots, S$.

Algorithm 5: Compute optimal division of users into subgroups.

```
Input: users $U'$, optimal travel costs $OptCost$.
1 for $j = z + 1$ to $S$ do
2     foreach combination comb of $j$ users from $U'$ do
3         $OptCost[comb] \leftarrow inf$
4         foreach division of comb to $comb_1$ and $comb_2$ do
5             $temp \leftarrow OptCost[comb_1] + OptCost[comb_2]$
6             if $OptCost[comb] > temp$ then
7                 $OptCost[comb] \leftarrow temp$
8                 $bestSubgroupping \leftarrow [comb_1, comb_2]$
Output: optimal travel costs $OptCost$, optimal division
        $bestSubgroupping$.
```

Algorithm 5 solves the second sub-stage of our solution. The for loop in lines 1-8 considers all group sizes from $z+1$ to S. For a given group size j, line 2 iterates through all possible subsets *comb* of j users. The for loop in lines 4-8 examines all possible divisions of *comb* into two subgroups $comb_1$ and $comb_2$. The optimal travel cost of *comb* in the current division is the sum of the optimal travel costs of two subgroups $comb_1$ and $comb_2$. If the optimal travel cost in the current division is smaller than the stored optimal travel cost, updates the information of the optimal travel cost and subgroup in lines 7-8. When the for loop in lines 1-8 finishes, the algorithm finds the optimal partition of group U'.

4.3 Analysis of time complexity

In Algorithm 1, the time complexity of building a binary heap of size n is $O(n)$ and that of extracting min k times is $O(k \log n)$. Therefore, the time complexity of Algorithm 1 is $O(n + k \log n)$.

In Algorithm 2, the while loop repeats $\frac{|U|}{S}$ times (suppose that $|U|$ is divisible by S for simplicity). For each loop, the time complexity of the for loop in lines 7-13 is $O(|X|^2)$, while the time complexity of finding k nearest neighbors in line 18 is $O(|X| + k \log |X|)$. Therefore, the time complexity of each loop is $O(|X|^2)$. X is the set of remaining users. Starting at $|U|$, the size of X reduces S after each loop until it reaches 0. In total, the time complexity of the whole while loop is $\sum_{1 \leq i \leq \frac{|U|}{S}} (iS)^2 = O((\frac{|U|}{S})^3 S^2) = O(\frac{|U|^3}{S})$, so is the time complexity of Algorithm 2.

Because Algorithm 2 directly solves the first step of our solution, the time complexity of the first step of our solution is also $O(\frac{|U|^3}{S})$. In comparison with the first step in the VST-RS method, it depends on the number of POIs, thus the worst time complexity is $O(|U||V|^2)$. When the number of POIs is larger than the number of users, ours runs faster than that of the VST-RS method.

In Algorithm 3, the for loop in lines 2-5 builds a heap of size at most $|V|$. Thus, the time complexity of this loop is $O(|V|)$. The while loop repeats at most $|V|$ times. Inside each loop, the cost of extracting the node with the highest priority is $O(\log |V|)$. The for loop in lines 25-40 repeats at most $|E|$ times for all loops. Inside each loop, the for loop in lines 32-36 repeats at most $|comb|$ times while the time complexity of inserting into the priority queue in line 40 is $O(\log |V|)$. Therefore, the time complexity of the for loop in lines 25-40 is $O(|E|(|comb| + \log |V|)) = O(|E| \log |V|)$ as $|comb|$ is not larger than a constant z. In total, the time complexity of the while loop is $O(|V| \log |V| + |E| \log |V|)$. Therefore, the time complexity of Algorithm 3 is $O(|V| \log |V| + |E| \log |V|)$. Because the degree of a node in the real road network usually varies from 2 to 4, the number of edges $|E|$ is larger than $|V|$ but much smaller than $|V|^2$. Thus, the worst time complexity of Algorithm 3 is $O(|E| \log |V|)$, which is usually much smaller than $O(|V|^2 \log |V|)$. If we used list instead of binary heap in Algorithm 3, the time complexity would be $O(|V|^2)$. For simplicity, we will use $O(|V|^2)$ to compute the time complexity of other algorithms.

In Algorithm 4, the for loop in lines 1-2 repeats S times. Thus, the time complexity of this loop is $O(S|V|^2)$. For a given group size j, the loop in lines 4-30 repeats $\binom{S}{j}$ times as there are $\binom{S}{j}$ subsets of size j of a set of size S. The for loop in lines 8-9 has a time complexity of $O(|V|)$. There are $2^j - 2$ subsets of *comb* (here, *comb* and $\emptyset$ are excluded), thus there are $2^{j-1} - 1$ different divisions. The for loop in lines 16-25 repeats at most $|V|$ times. Two loops in lines 22-23 and 24-25 repeat in total j times. So, the time complexity of for loop in lines 10-25 is $O(2^{j-1}j|V|)$. The time complexity of line 26 is $O(|V|^2)$. In total, the time complexity of each group size j is $O(|V|+2^{j-1}j|V|+|V|^2) = O(2^{j-1}j|V|+|V|^2)$. Therefore, the time complexity of Algorithm 4 is $O(S|V|^2+\sum_{2\leq j\leq z}\binom{S}{j}(2^{j-1}j|V|+|V|^2)) = O(\sum_{0\leq j\leq z}\binom{S}{j}(2^{j-1}j|V|+|V|^2))$. Because $j \leq z$ and z is a constant in our solution, $O(\sum_{0\leq j\leq z}\binom{S}{j}(2^{j-1}j|V|+j|V|^2)) = O(\sum_{0\leq j\leq z}\binom{S}{j}(|V|+|V|^2) = O(|V|^2\sum_{0\leq j\leq z}\binom{S}{j}) = O(|V|^2S^z)$.

In Algorithm 5, the for loop in lines 2-8 repeats $\binom{S}{j}$ and the for loop in lines 4-8 repeats $2^{j-1} - 1$ times, which are similar to Algorithm 4. The code in lines 5-8 has the time complexity of $O(1)$. Therefore, the time complexity of Algorithm 5 is $O(\sum_{z+1\leq j\leq S}\binom{S}{j}2^{j-1}) = O(\sum_{0\leq j\leq S}\binom{S}{j}2^{j-1}) = O(3^S)$.

The time complexity of running the second step for a group of size S is $O(|V|^2S^z+3^S)$. Because there are, in total, $\frac{|U|}{S}$ groups, the time complexity of the approximation solution is $O(\frac{|U|}{S}(3^S+S^z|V|^2))$. In the exact solution, S equals to $|U|$, thus the time complexity of the exact solution is $O(3^{|U|}+|U|^z|V|^2)$.

The time complexity of the Dreyfus-Wagner algorithm is $O(3^{|U|-1}|V|+2^{|U|-1}|V|^2)$. Because $z \leq S < |U|$ in the approximation solution, our approximation solution runs much faster than the Dreyfus-Wagner algorithm. The exact solution also runs faster than the Dreyfus-Wagner algorithm.

The time complexity of the VST-RS method is $O(\frac{|U|}{S}(3^S+S^z|V|^2))$, which is the same as our approximation solution. However, our approximation solution, in general, runs slower than the VST-RS solution. Because in Algorithm 3 and 4, it is necessary to calculate the travel distance for each user in a subgroup and check the travel distance constraint for each meeting point.

5 Experiments

Source code is available at https://github.com/hsgser/ABRSWC.

To evaluate the efficiency of our solution, the travel plan was computed using four different methods. The first method is the greedy approach (referred to as "greedy" in figures) in which each user travels to the nearest POI. The second and third method is the approximation solution (referred to as "approx" in figures) and optimal solution (referred to as "optimal" in figures), respectively. The last method is the approximation solution without the choice of POI (referred to as "approx_woc" in figures). In the last method, users are restricted to ride-share to their nearest POI. In other words, users are divided into groups based on their nearest POI instead of using the first step in the solution. We used the following four different metrics to compare methods.

- Mean processing time, unit second: the time required to run the algorithms.
- Total travel cost, unit meter: the total travel distance by all cars.
- Approximation ratio: the ratio between the total travel cost of each method in comparison with the optimal solution.
- Ride-sharing efficiency: the ratio between the total travel cost when users do not ride-share (greedy approach) and the total cost of ride-sharing methods.

We run five experiments in total. The first experiment was to compare four different methods. The other experiments were to evaluate the effect of the group size, the number of users, the number of POIs, and the extra ratio, respectively. Because the optimal solution was much slower compared to the other methods. We run it only in the first experiment. Besides, the processing time for the greedy approach was usually 0 even if we measured it in microseconds. Therefore, we excluded the greedy approach from the mean processing time graphs.

5.1 Data and Settings

We used the road network data of Shinjuku city from OpenStreetMap. It was downloaded using the Python package OSMnx [1]. The obtained data was a directed multigraph with 5910 nodes and 14428 edges. We further simplified the graph to match the input format of the solutions via the following three operations (OSMnx does some graph simplification automatically). Firstly, we consolidated intersection because an intersection tends to be divided into 4 nodes in the real road network. Secondly, we converted the directed multigraph to the undirected (simple) graph. We kept the shortest edge among all parallel edges between two adjacent nodes and removed all loops in the graph. The graph up to this point can be passed into the solutions. Finally, the length of each edge (with the unit of meter) is rounded to the nearest integer. There are, of course, no problems working with edge lengths in the float format. However, keeping float format barely affects the overall result but increases the size of data. The final graph is an undirected (simple) graph with 3002 nodes and 5053 edges.

The shortest distance between two nodes in the graph was computed using Dijkstra's algorithm. The pair-wise shortest distance is an essential element when running the solutions. Due to the complexity, it should be computed only once and stored for later use. Both users and POIs were randomly generated using uniform distribution among the set of all nodes in the graph. The number of users varied from 8 to 64, while the number of POIs varied from 10 to 100. After selecting the set of POIs, the nearest POI to each node in the graph was computed. We changed the group size from 4 to 16 people. The number of seats for each passenger car is set to be 4. For simplicity, the extra ratio, which ranged from 0 to 1, is also the same for each user. Table 1 summarizes the range of parameters and their default values. For each set of parameters $(z, \epsilon, S, N_u, N_p)$, we run the experiment 10 times and take the average values of metrics.

Parameters	Notation	Range	Default
Number of seats	z	4	4
Extra ratio	ϵ	0 - 1	0.5
Group size	S	4 - 16	8
Number of users	N_u	8 - 64	16
Number of POIs	N_p	10 - 100	20

Table 1: Parameters summary

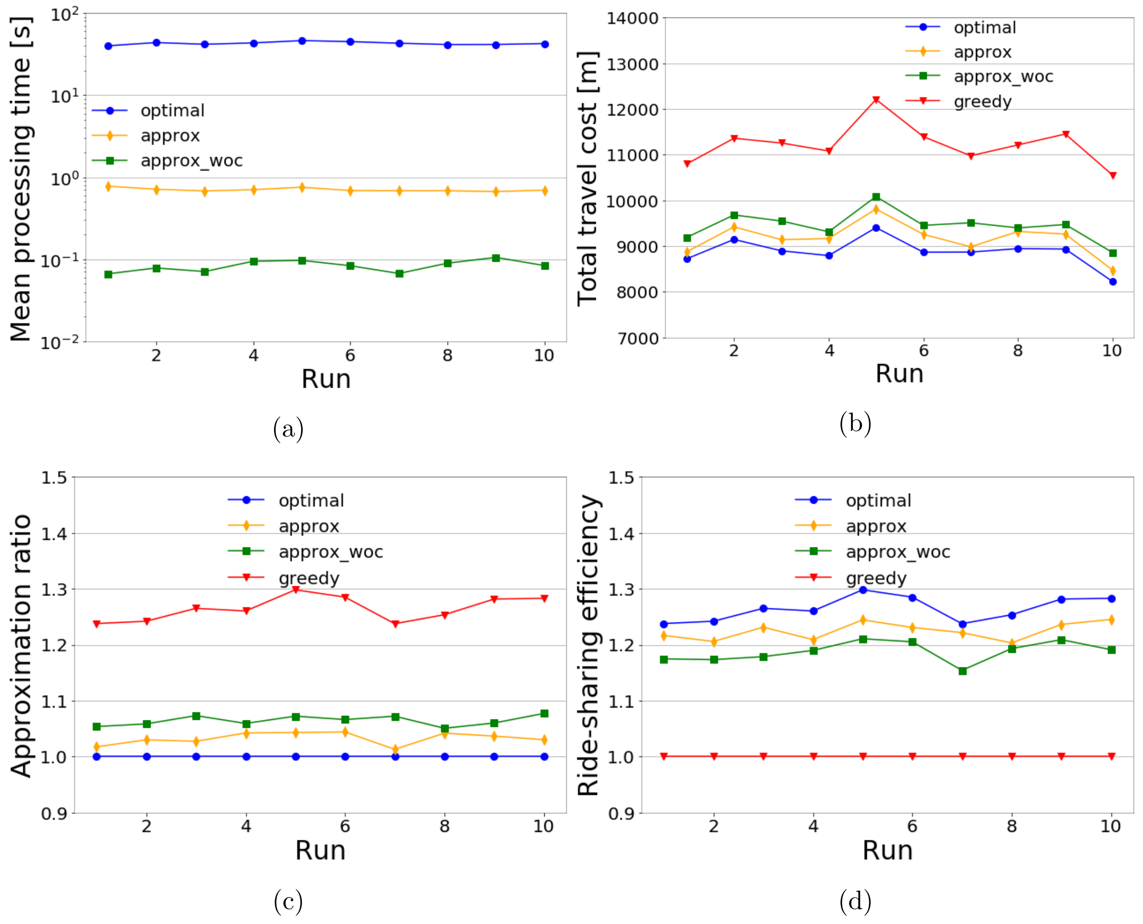
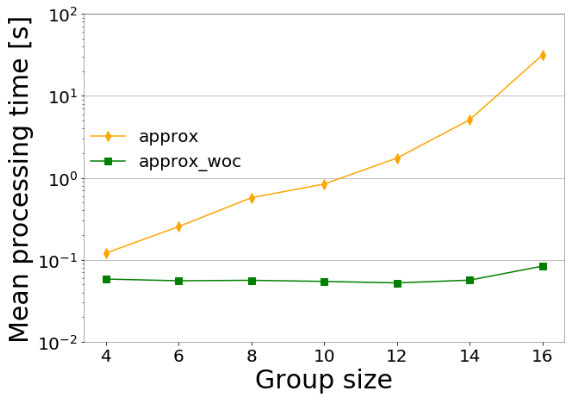


Figure 3: Compare (a) mean processing time, (b) total travel cost, (c) approximation ratio, and (d) ride-sharing efficiency between methods.

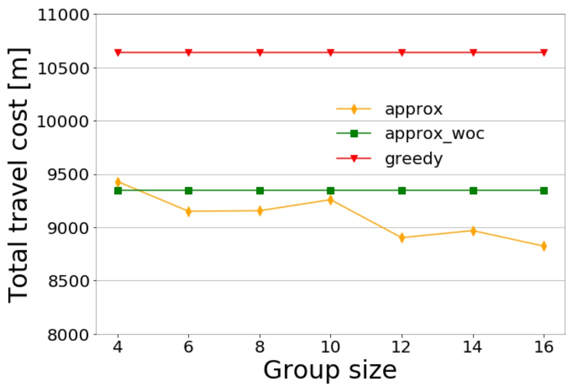
5.2 Methods comparison

Figure 3a shows that the optimal solution was more than one order of magnitude slower than the approximation solution. While the approximation solution without the choice of POI was ten times faster than the approximation solution. The main source of difference was the group size in the second step of the solution. As the group size increases, the run time of the solution increases (More details about the effect of the group size will be explained later). In the default setting, the group size of the approximation solution was only 8 while the group size of the optimal solution was 16. As for the approximation solution without the choice of POI, each group corresponded to a POI, thus had only $16/20 = 0.8$ users on average. Figure 3b shows that the greedy approach had the largest total travel cost as expected. This proved the efficiency of ride-sharing methods. The same conclusion can be drawn from Figure 3d. The ride-sharing efficiency was always larger than 1.1 for the approximation solution without the choice of POI while the ratios for two other ride-sharing methods were always larger than 1.2. Although the choice of POI increased the processing time, the approximation solution outperformed the approximation solution without the choice of POI in all runs. Intuitively, the fewer the number of groups is, the better the approximation is, the smaller the approximation ratio is. Because there were only two groups in the default setting, the approximation ratio of the approximation solution was at most 1.05 as seen in Figure 3c. While the approximation ratio of the approximation solution without the choice of POI fluctuated between 1.05 and 1.1. The actual values in Figure 3 are given in Appendix A.

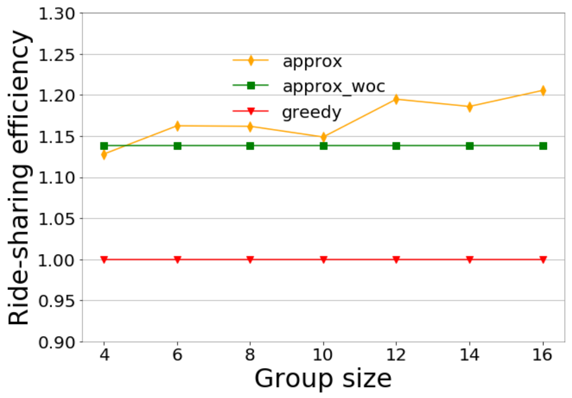
5.3 Effect of the group size



(a)



(b)



(c)

Figure 4: Effect of the group size on (a) mean processing time, (b) total travel cost, and (c) ride-sharing efficiency.

Figure 4 shows that the mean processing time, total travel cost, and ride-sharing efficiency were unchanged in the case of the approximation without the choice of POI and the greedy approach. Because the group size parameter only affected the approximation solution. In the first step of the approximation solution, the set of all users is divided into smaller groups to reduce the processing time, which comes at the expense of an increase in the total travel cost found in the second step. As the group size increases, the approximation solution gets closer to the optimal solution. This means that the mean processing time and the ride-sharing efficiency increases while the total travel cost decreases. The group size of 8 seems to be a good choice in this case considering both the processing time and the total travel cost. The processing time was less than 1 second, suggesting that it is suitable to deploy into a real-time ride-sharing system. The actual values in Figure 4 are given in Appendix B.

5.4 Effect of the number of users

Intuitively, increasing the number of users increases the processing time and the total travel cost for every method. Figure 5a shows that the processing time of the approximation solution increased as the increase of the number of users created more groups in the first step. As for the approximation solution without the choice of POI, the increase in the number of users did not increase the number of groups but increased the number of users in each group. In this method, the mean processing time also rose in almost all of the cases (except the case of 80 users). Besides, the approximation without the choice of POI showed a higher rate of increase compared to the other method. When the number of users was 72, its mean processing time was approximately 10 times of the competitor. Therefore, the approximation solution is more stable than the approximation solution without the choice of POI. Figure 5b shows that all methods experienced an increment in the total travel cost as expected. The increasing speed of the greedy approach was much higher than the other methods. When the number of users was 80, there was a huge gap, nearly 20km, between the greedy approach and two ride-sharing methods. The larger the number of users is, the higher the matching rate is, the more the ride-sharing methods save. Figure 5c shows that the approximation solution outperformed the approximation

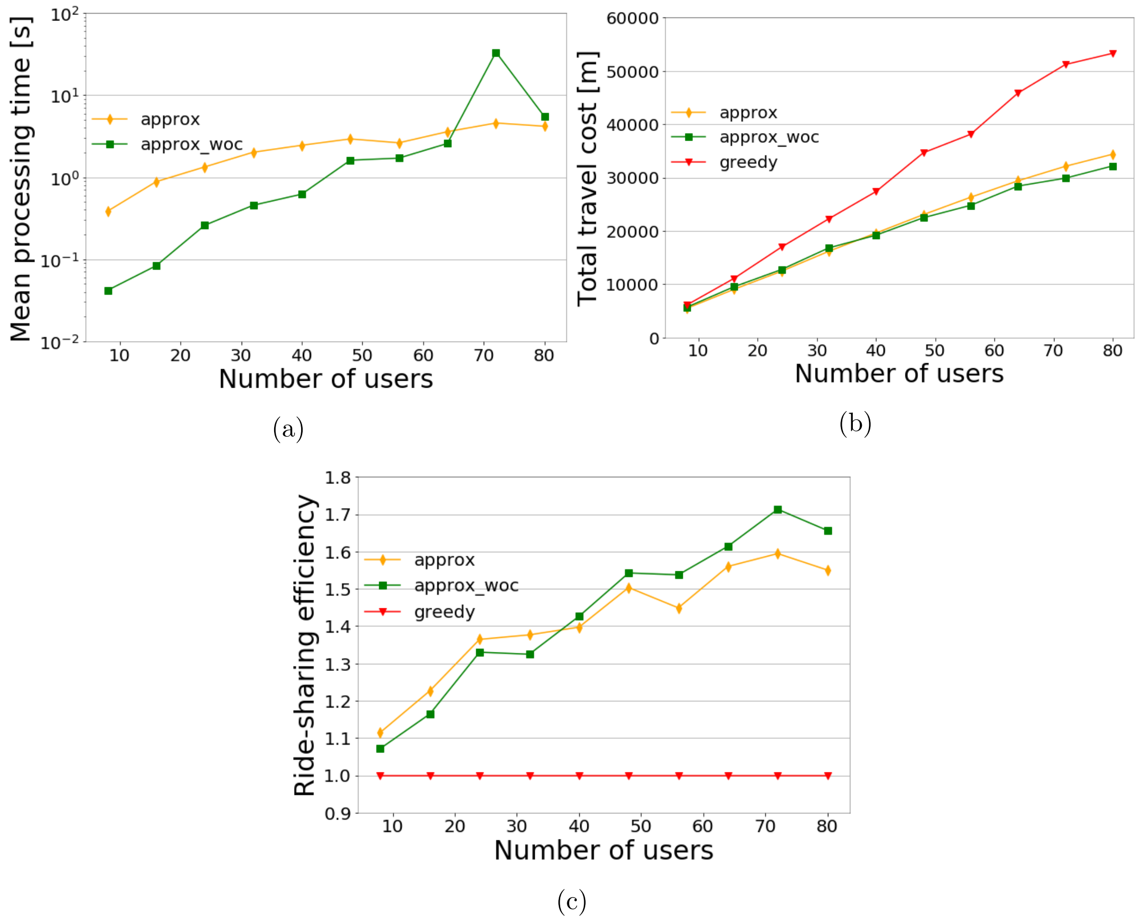
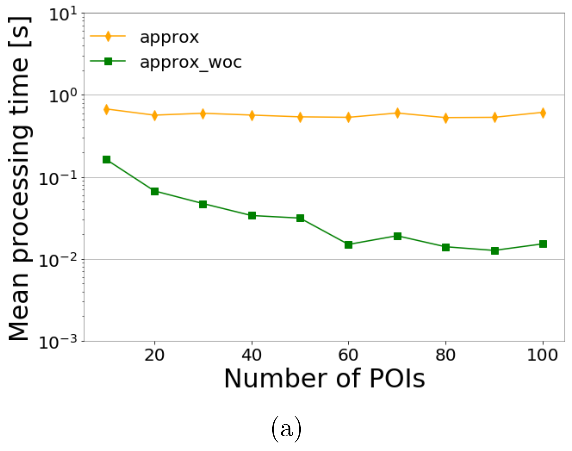


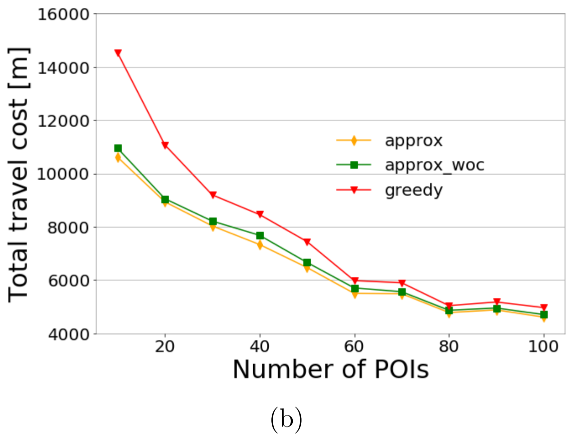
Figure 5: Effect of the number of users on (a) mean processing time, (b) total travel cost, and (c) ride-sharing efficiency.

solution without the choice of POI when the number of users is less than 40. As the set of users expands, the group size in the approximation solution without the choice of POI increases, thus the ride-sharing efficiency is larger. The actual values in Figure 5 are given in Appendix C.

5.5 Effect of the number of POIs

Figure 6a shows that the mean processing time of the approximation solution remained unchanged when the number of POIs increased while the mean processing time of the approximation solution without the choice of POI fell. Because the time complexity of the approximation solution does not depend on the number of POI. As for the approximation solution without the choice of POI, when the number of POI increases, the group size decreases, thus the run time decreases. Figure 6b shows that the total travel cost of





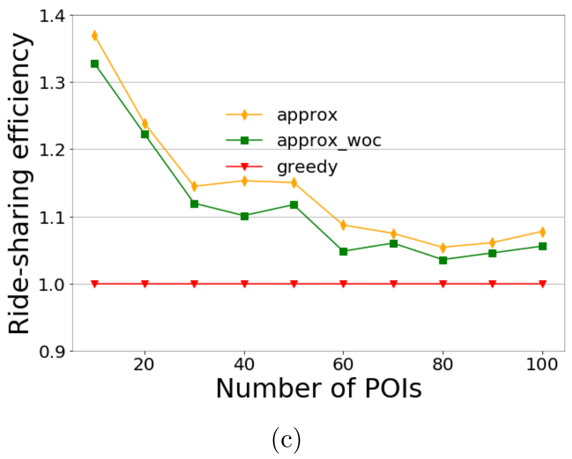


Figure 6: Effect of the number of POIs on (a) mean processing time, (b) total travel cost, and (c) ride-sharing efficiency.

all methods declined significantly when the number of POIs varied from 10 to 60, then decreased slightly when the number of POIs increased from 60 to 100. Because increasing the number of POIs reduces the distances from users to their nearest POIs, then reduces the distances to their optimal POIs. Figure 6c shows that the ride-sharing efficiency dropped significantly to less than 1.2 as the number of POIs varied from 10 to 30. As the distances between users and POIs decrease, ride-sharing matching is more likely to violate the travel distance constraint. As a result, in many cases, the users just travel to their nearest POIs as in the greedy approach. When the number of POIs is 100, ride-sharing save less than 10% of the total travel cost. The actual values in Figure 6 are given in Appendix D.

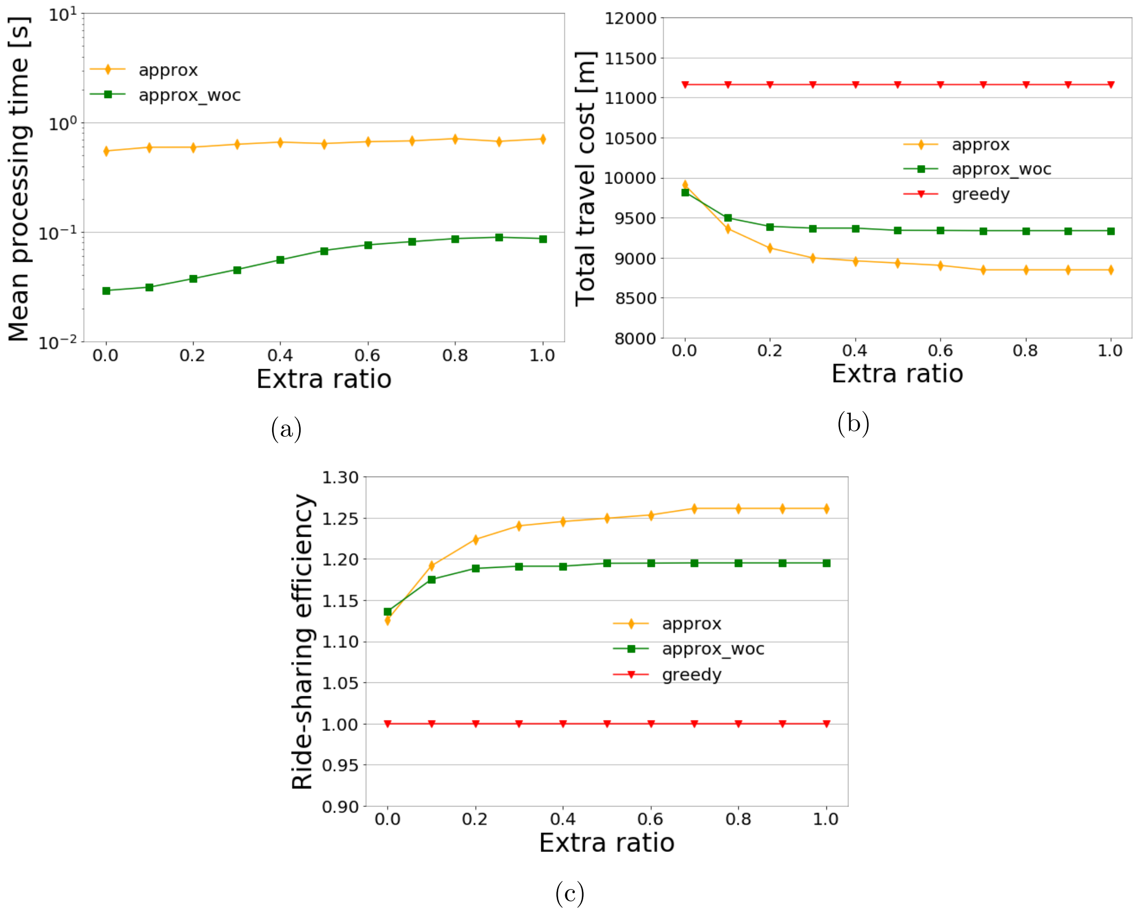


Figure 7: Effect of the extra ratio on (a) mean processing time, (b) total travel cost, and (c) ride-sharing efficiency.

5.6 Effect of the extra ratio

Figure 7a shows that the mean processing time of the approximation solution barely changed while that of the approximation solution without the choice of POI increased. Figure 7b shows that the total travel cost of the greedy approach remained unchanged while the travel cost of two ride-sharing methods decreased remarkably at the beginning, then declined slightly. Because the extra ratio does not involve in the greedy approach but it plays a crucial role in the ride-sharing decision. Intuitively, the larger the extra ratio is, the more flexible the matching is, the larger the ride-sharing efficiency is. Figure 7c shows that starting at around 1.13, the ride-sharing efficiency of the approximation solution increased up to 1.26 when the extra ratio increased from 0 to 1, while that of the approximation solution without the choice of POI only increased to 1.2 only. The actual values in Figure 7 are given in Appendix E.

6 Conclusion

In this paper, we generalized the previous work on activity-based ride-sharing by introducing the travel distance constraint. We present an exact solution and an approximation solution based on dynamic programming for the problem. Our experimental results on real road network data show that our solutions surpass other competing methods. Although the problem is NP-hard, our approximation solution is fast, stable, and scalable when the number of users increases. It would be wonderful if our solution will be integrated into a real-time ride-sharing system. For future work, we plan to consider the time constraint of users, i.e., users travel in different limited windows of time.

Acknowledgements

We would like to thank UEDA Masaya for the idea of using OSMnx to download OpenStreetMap data.

References

[1] Geoff Boeing. "OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks". In: *Computers, Environment and Urban Systems* Vol. 65, (2017), pp. 126–139.

[2] Stuart E Dreyfus and Robert A Wagner. "The Steiner problem in graphs". In: *Networks* Vol. 1, No. 3, (1971), pp. 195–207.

[3] A. K. M. Mustafizur Rahman Khan et al. "Ride-Sharing is About Agreeing on a Destination". In: *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '17. Redondo Beach, CA, USA: Association for Computing Machinery, 2017, pp. 6:1–6:10. ISBN: 9781450354905. DOI: 10.1145/3139958.3139972. URL: https://doi.org/10.1145/3139958.3139972.

[4] Bernhard Korte et al. *Combinatorial optimization*. Vol. 2. Springer, 2012, p. 378. ISBN: 9783662560389.

[5] S. Ma, Y. Zheng, and O. Wolfson. "T-share: A large-scale dynamic taxi ridesharing service". In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. 2013, pp. 410–421. DOI: 10.1109/ICDE.2013.6544843.

[6] Shuo Ma, Yu Zheng, and Ouri Wolfson. "Real-time city-scale taxi ridesharing". In: *IEEE Transactions on Knowledge and Data Engineering* Vol. 27, No. 7, (2014), pp. 1782–1795.

[7] Jean Monlong. *Clustering into same size clusters*. June 2018. URL: http://jmonlong.github.io/Hippocamplus/2018/06/09/cluster-same-size/. accessed 24 October 2020.

[8] Kazuki Takise. “Reservable Real-time Taxi Ridesharing”. MA thesis. Yoshikawa-Ma Laboratory M2, Graduate School of Informatics, Kyoto University.

[9] Kazuki Takise, Yasuhito Asano, and Masatoshi Yoshikawa. “Multi-User Routing to Single Destination with Confluence”. In: *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.* SIGSPACIAL ’16. Burlingame, California: Association for Computing Machinery, 2016, pp. 72:1–72:4. ISBN: 9781450345897. DOI: 10.1145/2996913.2997018. URL: https://doi.org/10.1145/2996913.2997018.

[10] Yaoli Wang, Ronny Kutadinata, and Stephan Winter. “Activity-Based Ridesharing: Increasing Flexibility by Time Geography”. In: *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.* SIGSPACIAL ’16. Burlingame, California: Association for Computing Machinery, 2016, pp. 1:1–1:10. ISBN: 9781450345897. DOI: 10.1145/2996913.2997002. URL: https://doi.org/10.1145/2996913.2997002.

[11] X. Zhang, Y. Asano, and M. Yoshikawa. “Mutually Beneficial Confluent Routing”. In: *IEEE Transactions on Knowledge and Data Engineering* Vol. 28, No. 10, (2016), pp. 2681–2696. DOI: 10.1109/TKDE.2016.2590435.

Appendices

A Experiment 1: Method comparison

Run	greedy	optimal	approx	approx_woc
1	0.000000	39.972292	0.777624	0.066177
2	0.000000	43.826830	0.713174	0.078114
3	0.000097	41.751241	0.678946	0.070615
4	0.000000	43.381225	0.703977	0.094355
5	0.000000	46.299124	0.754335	0.096643
6	0.000097	45.003437	0.686456	0.083131
7	0.000198	43.027262	0.682173	0.066935
8	0.000100	41.444201	0.683003	0.089269
9	0.000100	41.509849	0.667716	0.105167
10	0.000100	42.606138	0.695376	0.083485

Table 2: Mean processing time with default settings, unit second

Run	greedy	optimal	approx	approx_woc
1	10795.6	8721.6	8874.3	9189.9
2	11359.4	9146.3	9420.8	9681.6
3	11255.3	8897.2	9140.8	9549.1
4	11081.1	8792.6	9164.7	9313.7
5	12207.9	9404.4	9810.7	10084.4
6	11394.7	8867.0	9257.6	9454.2
7	10975.9	8869.2	8984.8	9509.9
8	11213.4	8945.1	9320.2	9398.4
9	11452.0	8935.7	9263.2	9472.4
10	10550.6	8223.8	8473.3	8857.8

Table 3: Total travel cost with default settings, unit meter

Run	greedy	optimal	approx	approx_woc
1	1.237800	1	1.017508	1.053694
2	1.241967	1	1.030012	1.058526
3	1.265038	1	1.027379	1.073270
4	1.260276	1	1.042320	1.059266
5	1.298105	1	1.043203	1.072307
6	1.285068	1	1.044051	1.066223
7	1.237530	1	1.013034	1.072239
8	1.253580	1	1.041934	1.050676
9	1.281601	1	1.036651	1.060062
10	1.282935	1	1.030339	1.077093

Table 4: Approximation ratio with default settings

Run	greedy	optimal	approx	approx_woc
1	1	1.237800	1.216502	1.174724
2	1	1.241967	1.205779	1.173298
3	1	1.265038	1.231325	1.178677
4	1	1.260276	1.209107	1.189763
5	1	1.298105	1.244345	1.210573
6	1	1.285068	1.230848	1.205253
7	1	1.237530	1.221608	1.154155
8	1	1.253580	1.203129	1.193118
9	1	1.281601	1.236290	1.208986
10	1	1.282935	1.245158	1.191108

Table 5: Ride-sharing efficiency with default settings

B Experiment 2: Effect of the group size

Group size	greedy	approx	approx_woc
4	0.000000	0.120382	0.058184
6	0.000000	0.253731	0.055441
8	0.000000	0.571940	0.056162
10	0.000000	0.839824	0.054442
12	0.000004	1.739359	0.052221
14	0.000000	5.114401	0.056437
16	0.000000	31.669110	0.083628

Table 6: Mean processing time when varying the group size from 4 to 16, unit second

Group size	greedy	approx	approx_woc
4	10636.8	9429.1	9345.6
6	10636.8	9150.3	9345.6
8	10636.8	9154.8	9345.6
10	10636.8	9258.7	9345.6
12	10636.8	8902.2	9345.6
14	10636.8	8968.6	9345.6
16	10636.8	8823.4	9345.6

Table 7: Total travel cost when varying the group size from 4 to 16, unit meter

Group size	greedy	approx	approx_woc
4	1	1.128082	1.138161
6	1	1.162454	1.138161
8	1	1.161882	1.138161
10	1	1.148844	1.138161
12	1	1.194851	1.138161
14	1	1.186005	1.138161
16	1	1.205522	1.138161

Table 8: Ride-sharing efficiency when varying the group size from 4 to 16

C Experiment 3: Effect of the number of users

Number of users	greedy	approx	approx_woc
8	0.000100	0.384389	0.041667
16	0.000000	0.879962	0.083092
24	0.000100	1.332966	0.258742
32	0.000102	2.014182	0.454363
40	0.000000	2.449786	0.620761
48	0.000100	2.928618	1.609040
56	0.000000	2.623922	1.707706
64	0.000099	3.589229	2.576428
72	0.000100	4.578339	33.464726
80	0.000294	4.177905	5.457837

Table 9: Mean processing time when varying the number of users from 8 to 80, unit second

Number of users	greedy	approx	approx_woc
8	6088.4	5462.2	5682.0
16	11083.4	9036.5	9518.5
24	16939.5	12416.2	12737.3
32	22250.0	16165.0	16799.3
40	27372.2	19593.4	19194.5
48	34643.6	23045.7	22462.3
56	38112.7	26305.8	24789.9
64	45843.6	29388.4	28403.6
72	51185.6	32105.0	29879.7
80	53291.0	34375.7	32175.8

Table 10: Total travel cost when varying the number of users from 8 to 80, unit meter

Number of users	greedy	approx	approx_woc
8	1	1.114642	1.071524
16	1	1.226515	1.164406
24	1	1.364306	1.329913
32	1	1.376431	1.324460
40	1	1.397011	1.426044
48	1	1.503257	1.542300
56	1	1.448833	1.537429
64	1	1.559922	1.614007
72	1	1.594319	1.713056
80	1	1.550252	1.656245

Table 11: Ride-sharing efficiency when varying the number of users from 8 to 80

D Experiment 4: Effect of the number of POIs

Number of POIs	greedy	approx	approx_woc
10	0.00000	0.674476	0.164127
20	0.000000	0.565736	0.067346
30	0.000010	0.596385	0.047228
40	0.00000	0.567168	0.033729
50	0.000000	0.539597	0.031367
60	0.00000	0.532223	0.014971
70	0.000000	0.600393	0.019096
80	0.00008	0.528305	0.014018
90	0.00000	0.532323	0.012651
100	0.00000	0.612102	0.015191

Table 12: Mean processing time when varying the number of POIs from 10 to 100, unit second

Number of POIs	greedy	approx	approx_woc
10	14535.4	10613.6	10948.4
20	11073.0	8936.0	9052.7
30	9196.0	8032.0	8212.8
40	8457.5	7333.7	7680.0
50	7446.2	6472.5	6663.2
60	5978.9	5499.5	5703.9
70	5898.4	5487.3	5563.0
80	5039.5	4781.3	4865.7
90	5176.6	4879.3	4950.7
100	4966.9	4608.8	4705.0

Table 13: Total travel cost when varying the number of POIs from 10 to 100, unit meter

Number of POIs	greedy	approx	approx_woc
10	1	1.369507	1.327628
20	1	1.239145	1.223171
30	1	1.144920	1.119716
40	1	1.153238	1.101237
50	1	1.150436	1.117511
60	1	1.087172	1.048213
70	1	1.074918	1.060291
80	1	1.054002	1.035719
90	1	1.060931	1.045630
100	1	1.077699	1.055664

Table 14: Ride-sharing efficiency when varying the number of users from 10 to 100

E Experiment 5: Effect of the extra ratio

Extra ratio	greedy	approx	approx_woc
0.0	0.00000	0.549235	0.029045
0.1	0.000000	0.594008	0.031177
0.2	0.000100	0.595027	0.037295
0.3	0.000100	0.632041	0.045141
0.4	0.000000	0.662035	0.055452
0.5	0.000104	0.642279	0.067574
0.6	0.000099	0.667491	0.076005
0.7	0.00000	0.679244	0.081321
0.8	0.00000	0.712358	0.086783
0.9	0.00000	0.672469	0.089219
1.0	0.00000	0.708586	0.086908

Table 15: Mean processing time when varying the extra ratio from 0 to 1, unit second

Extra ratio	greedy	approx	approx_woc
0.0	11158.3	9909.5	9819.9
0.1	11158.3	9364.5	9495.9
0.2	11158.3	9119.1	9389.5
0.3	11158.3	8996.9	9367.9
0.4	11158.3	8959.2	9367.9
0.5	11158.3	8931.2	9340.2
0.6	11158.3	8902.6	9339.0
0.7	11158.3	8846.0	9336.5
0.8	11158.3	8846.0	9336.5
0.9	11158.3	8846.0	9336.5
1.0	11158.3	8846.0	9336.5

Table 16: Total travel cost when varying the extra ratio from 0 to 1, unit meter

Extra ratio	greedy	approx	approx_woc
0.0	1	1.126020	1.136295
0.1	1	1.191553	1.175065
0.2	1	1.223619	1.188381
0.3	1	1.240238	1.191121
0.4	1	1.245457	1.191121
0.5	1	1.249362	1.194653
0.6	1	1.253375	1.194807
0.7	1	1.261395	1.195127
0.8	1	1.261395	1.195127
0.9	1	1.261395	1.195127
1.0	1	1.261395	1.195127

Table 17: Ride-sharing efficiency when varying the extra ratio from 0 to 1