Ride-sharing is About Agreeing on a Destination

A.K.M. Mustafizur Rahman Khan
University of Melbourne
khank@student.unimelb.edu.au

Oscar Correa
University of Melbourne
oscarcg@student.unimelb.edu.au

Egemen Tanin
University of Melbourne
etanin@unimelb.edu.au

Lars Kulik
University of Melbourne
lkulik@unimelb.edu.au

Kotagiri Ramamohanarao
University of Melbourne
kotagiri@unimelb.edu.au

ABSTRACT

Ride-sharing is rapidly becoming an alternative form of transportation mainly due to its economic benefits. Existing research on ride-sharing aims to optimally match trajectories between people with pre-selected destinations. In this paper, we show better ride-sharing arrangements are possible when users are presented with more destinations and agree on a common destination. Given a set of points of interest (POIs) and a set of users, our approach presents destination POIs and computes ride-sharing plans. Each arrangement for a subset of users that fit in a car can be presented as a minimum Steiner tree (MST) problem. An optimal solution of the overall problem minimizes the total length of all the MSTs. The problem is a version of the set cover problem and is NP-hard. We first develop a series of baseline methods which use a popular MST algorithm. Then, we propose our method which uses constraints on intermediary points where users can meet to share rides. These constraints reduce the time complexity significantly and our method is up to two orders of magnitude faster than the best baseline method. Since our algorithm finds the subsets of users and POIs for each arrangement, we define and solve a new type of MST problem as a first step. Our experiments show that our method can provide a fast and readily deployable solution for real world large city scenarios.

CCS CONCEPTS

• **Applied computing → Transportation**;

KEYWORDS

Ride-Sharing, Spatial Databases, Steiner Trees

ACM Reference format:
A.K.M. Mustafizur Rahman Khan, Oscar Correa, Egemen Tanin, Lars Kulik, and Kotagiri Ramamohanarao. 2017. Ride-sharing is About Agreeing on a Destination. In *Proceedings of ACM SIGSPATIAL Conference, Redondo Beach, California, USA, November 7–10, 2017 (SIGSPATIAL'17),* 10 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Situated between the use of private cars and public transport, ride-sharing combines their benefits, i.e., the flexibility of a private car with the low cost of public transport. Real-time ride-sharing, which matches drivers and passengers on the fly, requires passengers to provide, among other information, pick-up points and destinations [3–5]. We show that better ride-sharing arrangements are possible, when users agree on common destinations among a set of choices, rather than going to their pre-chosen destinations. For instance, consider the case of four people u_1, u_2, u_3 and u_4 who are about to choose among any of the three POIs p_1, p_2 and p_3 nearby. The locations of users and POIs are depicted in Figure 1. In the optimal solution, u_1 and u_2 would meet at q_1, leave one of their cars there and share one car to go to p_1; u_3 would pick up u_4 and go to p_3. Thick edges in Figure 1 represent the optimal travel plan. The total travel distance for all cars is $1+1+1+0.5+1.5 = 5$. There are many other sub-optimal plans. If the users do not share rides, they will likely go to their nearest POIs and the total travel distance would be $2 + 1.5 + 1.5 + 1.5 = 6.5$. Requesting to go to their nearest POIs without first agreeing on common destinations chosen from a set of options, e.g. u_3 wishes to go to p_2 and u_4 wishes to go to p_3 rather than agreeing to go together to p_3, will end up in sub-optimal plans.

In real-world example, a set of tourists may be interested in local restaurants. Since they may want to experience diffferent cuisines, our solution would present restaurants to which they may agree to ride-share by following the optimal computed routes. A mobile app that presents different options, divide users into groups that fit in a car and compute optimal ride-sharing plans would be useful. Going further, even existing platforms such as uberPool could use our solution. Suppose tourists want to go to Manhattan, where parking fares are very expensive and there are many close-by places to visit. They might not have a specific touristic attraction in mind to start with and a POI could be any of the non-visited attractions. An uberPool driver can pick-up these tourists and drop them off in the most convenient attraction(s). We create a setting where users can exchange information and agree on a destination and thus producing further benefits for ride-sharing.

In this paper, we develop a fast dynamic programming method to efficiently choose the destination POIs, divide the set of users into optimal subsets that fit in a car and compute the optimal travel plan for them. The optimal travel plan can be described as a set of MSTs, where the root of each tree is a POI and the leaves are users. An MST is the minimum cost subgraph that spans a subset of nodes in a graph. In our problem each subset is found by our algorithm,

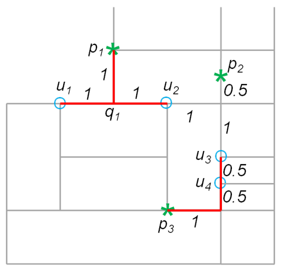

Figure 1: A simple road network. u_1**,** u_2**,** u_3 **and** u_4 **are users and** p_1**,** p_2 **and** p_3 **are POIs**

i.e., the subset of users who fit in a car and their corresponding POI, whereas in the classical MST problem such a subset is given. We define a new type of MST problem for this that we solve efficiently by the inclusion of constraints on the possible meeting points of users. We name this new problem "Variable Steiner Tree" (VST) as the POI is not set beforehand, and our solution to this problem VST-RS, where the suffix refers to "ride-sharing".

Our problem has many variations. In this paper, we show how VST-RS can be modified to address two of them, i.e., when users do not have a car and when they can meet at parking lots only, and discuss other related problems that could be studied in the future. The contributions of the paper can be summarized as follows:

- We generalize ride-sharing by allowing users to agree on common destinations from a set of possible options.
- We define a new MST problem we call "Variable Steiner Tree." We solve it in a novel way by imposing constraints on candidate meeting points thus significantly reducing computation time.
- We extend VST-RS to solve realistic variations of the problem.
- Our extensive experiments demonstrate that VST-RS is fast and can be deployed in real world applications.

2 RELATED WORK

2.1 Activity-based Ride-sharing

Recently, in transportation science, Wang et al.'s [8] work considers a choice set of POIs instead of fixed destinations which is similar to our goal. Their work shows that this paradigm improves the ride-sharing matching rate without presenting a viable algorithm for it. Our work shows that this paradigm can be solved efficiently as a new Steiner tree problem and through dynamic programming.

2.2 Car Pooling

Car pooling is a service organised by large companies which get employees pick up colleagues while driving to/from work [1]. Its purpose is to minimize the travel cost, which is typically the distance travelled by all cars. The problem is NP-hard [1]. Several exact and approximation algorithms were proposed to solve this problem. Recently, Oslen [7] considered a version of car pooling called "private park and ride" problem where the users meet at intermediary points and then share rides to a common destination. The author showed that the private park and ride problem is APX-complete even when the cost of all edges is 1. The author discussed about the complexity of the problem but proposed no algorithm to solve it. The car pooling problem appears to be similar to ours but in fact it is quite different. In car pooling there is a single fixed destination which is agreed on ahead of time, whereas in our problem there are multiple destinations and they are known ad-hoc, which makes heavy-weight optimization infeasible in our setting. In car pooling, the participants are commonly divided into servers and clients [1]. Servers start from their location, pick up clients and reach the destination. In our problem, a server may leave its car at an intermediary point and share a ride with other servers.

2.3 Minimum Steiner Trees

A minimum Steiner tree (MST[1]) is a connected sub-graph that spans a subset of nodes in a graph with the minimum cost.

In our case, when a group of users fit in a single car, their optimal routing to a given location is an MST. Since a car has limited seats, in the overall optimal travel plan for all users one has to divide them into groups so that each group can fit into a car. Therefore, the optimal travel plan consists of multiple MSTs. To determine the optimal travel plan we compute the travel cost of all possible groups and then select the groups so that all users are selected and the total cost is minimized. In addition, POIs have to be chosen during the execution of the algorithm and are not predetermined.

The MST problem is NP-hard. Dreyfus et al. [2] developed a fast solution which has been the best for the last four decades [6]. Recently, algorithms were proposed to improve the computation speed by decreasing the dependency on the number of terminals with the cost of increasing dependency on the number of nodes in the network. In our scenario, the number of nodes in the network is much larger than the number of terminals (users). Hence, Dreyfus et al.'s continues to perform better in our scenario. We refer to this algorithm regularly as we use it to build the baseline methods.

3 PROBLEM DEFINITION

Let U and P be the set of users and POIs, respectively. Let z be the same number of seats in each passenger car in our simplest setting. Our query finds interconnecting routes from all members of U to one or more elements of P that minimizes the total cost of the edges on the routes. For the sake of simplicity we consider the users and POIs are located at the nodes of the road network.

Given a network $G = (V, E)$ and a set of nodes $D \subseteq V$, $Cost(D)$ denotes the cost, i.e., the sum of the edge costs, of the MST that spans at least[2] D. We call the problem of finding the optimal routing for a group of users U' to the optimal POI assuming the users can fit in a car as *"Variable Steiner Tree" problem.* It can be defined as:

Given: An undirected network $G = (V, E)$, locations of users $L \subseteq V$ and locations of POIs $P \subseteq V$.

Find: A POI $p \in P$ and the corresponding MST such that $Cost(L \cup \{p\}) = \min_{p' \in P} Cost(L \cup \{p'\})$.

Let $OptCost(U')$ be the travel cost of users U' to the optimal POI.

For the overall problem, we need to divide all users into groups so that they can fit into multiple cars. Since more than z users cannot share a car, the maximum size of a group is z. Let $U = U_1' \cup U_2' \cup U_3' ... \cup U_t'$, where $|U_i'| \leq z$ and $U_i' \cap U_j' = \phi$ and $1 \leq i \leq$

[1]We abbreviate this problem as MST. Readers should not be confused with the minimum spanning tree problem.
[2]Additional nodes, called Steiner nodes, are also spanned

$t, 1 \leq j \leq t, i \neq j$. We need to find out the optimal division of U and the corresponding MSTs so that $\sum_{i=1}^{t} OptCost(U_i^{'})$ is minimized.

Solving this problem requires solving the set cover problem. In the set cover problem, there is a universal set and multiple subsets with associated costs. A solution is a collection of subsets so that their union is equal to the universal set and the total cost is minimized. In our problem the universal set is the set of users and the subsets are all possible subsets of less or equal to z users. The cost of a subset is the travel cost of the users in this subset to the optimal POI according to the optimal travel plan. The set cover problem is known to be NP-hard, hence our problem is also NP-hard.

4 SOLUTIONS

4.1 A Brute Force Approach

Dreyfus et al. [2] proposed an algorithm to compute an MST over a network. An MST spans a subset of nodes D in a graph and, in order to minimize its cost, additional nodes are also spanned. These additional nodes are called Steiner nodes and they correspond to the meeting points within our ride-sharing model. At the beginning, Dreyfus et al.'s algorithm builds Steiner subtrees for every combination of 2 users and one Steiner node, and stores the best Steiner node with its corresponding combination. Then, the same process is performed for every combination of 3 users based on the results for 2 users and so on, until the number of users for each combination is $|D| - 1$. In our scenario, the last node is the location of a POI and the remaining nodes ($|D| - 1$) are users' locations. Then, an MST is computed connecting the users to the POI. When computing an MST connecting the users to another POI, most of the computation done for the first POI can be reused. We can compute an optimal travel plan for a given set of users by computing the MSTs connecting different POIs and finding the MST with minimum length.

As a solution of our overall problem where all users cannot fit into a car, we can solve a version of the set cover problem to find the optimal division of the users into subsets so that the total cost is minimized. The cost of a subset is the cost of the MST which connects the users in the subset to a POI. We can search all subset combinations to find an optimal solution of the overall problem. The set cover problem is an NP-hard problem and thus this solution will incur prohibitive computation cost even if the cost of each subset is given. This solution will not scale even for a small number of users and POIs.

We later give a more pragmatic baseline solution using existing best MST practices. We show that even such approximate and seemingly efficient solution is much slower than ours in section 5.

4.2 Our Solution: VST-RS

At the core of our approach, there is the need to restrict the MST search to a reduced set of nodes, by first being able to divide a large city into smaller parts for an approximate overall solution and then, by finding optimal solutions within each part of the city. Our approach is based on two key observations for this. First, when people ride-share, the closer they are and the longer the route they share, the better trade-off we get from ride-sharing. Thus, people's locations is a sensible heuristic to divide the whole set of users as people who are far away will most likely not ride-share. Second, the whole road network is not of concern when computing the optimal travel plan for such sharers. By the inclusion of novel constraints on the meeting points, the search space can be further shrunk.

At high level, VST-RS has two steps. In the first step, based on users' locations, for a very large city, we divide users into groups of at most S users so that the computation time for each group is fast. (Later, we experimentally show that even a small value of S gives good ride-sharing arrangements.) We divide the users into groups starting with their nearest POIs. For computing the nearest POI to a user we find the corresponding Voronoi cell [3] for the user. If there are more than S users in a Voronoi cell we divide the users into multiple groups using the distances between the users. If there are less than S users in a Voronoi cell, we combine users from multiple Voronoi cells into a group. It is worth mentioning that this initial division does not force the users to ride-share to these POIs. We use the POI concept to create pockets in a large city with S users each.

The second step considers each group ($\leq S$) and has two sub-stages. In the first sub-stage, we compute the travel plan and cost for each subgroup ($\leq z$) by solving our new proposed Steiner tree problem where the whole set of terminals is not pre-defined. We use constraints to find the meeting points. We also develop an efficient algorithm built on Dijkstra's algorithm to compute the next possible intermediary meeting points. As a result, the optimal travel plan for each combination of z users that can fit in a car is computed by considering only a very small part of the network and thus further decreasing the computation time significantly. *Hence, at the core of our contributions we find a faster MST algorithm for our setting.*

Once the optimal travel plan for each combination of z users was computed in the first sub-stage, the second sub-stage finds the optimal division of all users into subgroups so that each subgroup fits in a car and the total cost for the group of S users is minimized.

In short, we reduce the need for a large network inspection for solving an MST problem by (i) realizing that different parts of the city can be solved independently, and (ii) for each part, MST search can also be efficient by choosing meeting points carefully.

In the rest of this section, we first derive the constraints on the meeting points and describe the process of finding them in section 4.2.1. Then, the computation of the optimal plan for each subgroup is presented in section 4.2.2. These two sections correspond to the first sub-stage. Section 4.2.3 shows the division of users into subgroups (second sub-stage,) concluding the second step of the overall algorithm. In section 4.2.4, we detail the first step of the overall algorithm to enable our work to run efficiently and accurately.

4.2.1 Finding the Intermediary Meeting Points. In order to compute the optimal travel plan for a subgroup of users, we need to find the intermediary meeting points where users meet each other and start sharing rides. We define the following keywords and properties of the optimal travel plan which we use to detect candidate intermediary meeting points of users.

Intermediary meeting point (IMP): An IMP for a set of users $U^{'}$ is a node where all users in $U^{'}$ can meet. The users leave redundant cars at an IMP and proceed to share rides to complete their trip.

[3]In a Voronoi cell, the distance of a node, which belongs to this cell, to the center of this cell is shorter than its distance to any of the other centers of the Voronoi cells.

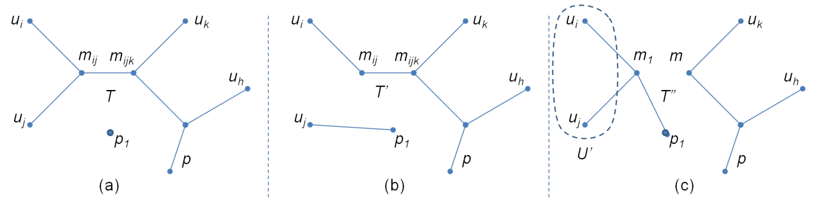

Figure 2: (a) An optimal travel plan, (b) a travel plan with reduced cost if distance between u_j and p_1 is less than distance between u_j and m_{ij} (c) a travel plan with reduced cost if cost of travelling $U^{'}$ to p_1 is less than the cost of travelling $U^{'}$ to m.

Next intermediary meeting point (NIMP): A NIMP for users $U^{'}$ is a node where users in $U^{'}$ can meet other users not in $U^{'}$.

Constraints on NIMPs: Firstly, for a single user.

Theorem 4.1. *Let u_j meet another user at node m_{ij} to share a ride in the optimal travel plan. The distance between m_{ij} and u_j must be smaller than the distance of the nearest POI from u_j.*

Proof. If the distance between u_j and m_{ij} is larger than the distance between u_j and the nearest POI from u_j, u_j can directly go to this POI and the travel cost can be reduced. In Figure 2(a), T represents an optimal travel plan. In the figure, the nearest POI from u_j is p_1. If the distance between u_j and m_{ij} is larger than the distance between u_j and p_1, u_j will go to p_1 and the rest of the users go to p as shown by the graph $T^{'}$ in Figure 2(b), and in this way the travel cost can be reduced. But T is the optimal travel plan and a travel plan with lower cost is not possible. Thus, in an optimal travel plan the distance between u_j and m_{ij} is smaller than the distance between u_j and the nearest POI from u_j. □

We can expand the idea of NIMP to consider multiple users. NIMP of a set of users is a node where these users may meet others.

Theorem 4.2. *Let $Cost(U^{'} \cup \{m\})$ be the travel cost of $U^{'}$ to m. If m is a NIMP of set of users $U^{'}$, $Cost(U^{'} \cup \{m\}) \leq \min_{p \in P} Cost(U^{'} \cup \{p\})$.*

Proof. Suppose a set of users $U^{'}$ have met at an intermediary meeting point m_1 and then they travel to NIMP m to meet other user(s). In the optimal travel plan T, the cost of $U^{'}$ to travel to m must be smaller than the cost of $U^{'}$ to travel to any POI. Suppose there is a POI p_1 for which $Cost(U^{'} \cup \{p_1\}) < Cost(U^{'} \cup \{m\})$. In this case, the sub-tree which connects $U^{'}$ to m will be replaced by the optimal route that connects $U^{'}$ to p_1 as shown in Figure 2(c) and the new travel plan $T^{''}$ will cost less than T. But T is the optimal travel plan. Thus, for any $p \in P$, $Cost(U^{'} \cup \{p\}) > Cost(U^{'} \cup \{m\})$. □

Constraints on IMPs: Let X_i be the set of NIMPs for user u_i. At first, we consider constraints on IMPs of a set of two users. If u_i meets other users at m in the optimal travel plan, m must be a NIMP of u_i and thus $m \in X_i$. Hence, if two users u_i and u_j meet at point m_{ij}, $m_{ij} \in M_{ij} = X_i \cap X_j$.

For any IMP m_{ij}, $dist(u_i, u_j) \geq dist(u_i, m_{ij})$ and $dist(u_i, u_j) \geq dist(u_j, m_{ij})$. If $dist(u_i, u_j) < dist(u_i, m_{ij})$, u_i can travel to u_j and share a ride from there and the travel cost can be reduced. Hence, in an optimal travel plan $dist(u_i, u_j) \geq dist(u_i, m_{ij})$. Similarly, we can show that $dist(u_i, u_j) \geq dist(u_j, m_{ij})$.

When two sets of users meet, we can compute constraints on the IMPs. The IMP must be a NIMP for both sets. In the optimal travel plan, if $U^{'}$ meet the users $U^{''}$ at m, the cost of $U^{'}$ to travel to m must be smaller than the cost of $U^{'}$ to travel to any $u \in U^{''}$.

Algorithm to compute NIMPs: It is based on Dijkstra's shortest path algorithm and is a function we name *FindNIMP*.

In Dijkstra's algorithm, given a source o, a destination d and a graph G, it incrementally searches for the shortest path from o to d. During the search, the algorithm finds all nodes of G whose distances from o are smaller than $dist(o, d)$. Thus, given a user's location *FindNIMP* finds all nodes of G whose distances from the user's location are smaller than the distance from this location to the nearest POI. For instance, in Figure 3, u_1, u_2, u_3 and u_4 are users and p_1, p_2 and p_3 are POIs. If we run *FindNIMP* taking u_1 as source and the POIs as destinations or boundary nodes, the NIMPs of u_1 are the nodes whose distances from u_1 are smaller than 2 because p_1, the nearest POI to u_1, is located at 2 units. X_1 includes all NIMPs of u_1, so $X_1 = \{(2,4),(3,4),(3,3),(4,4)\}$. The cost of a NIMP is its distance from u_1. Similarly, we compute X_2, X_3 and X_4, NIMP sets of u_2, u_3 and u_4. Thus, $X_2 = \{(4,4),(5,4),(6,4),(5,3)\}$, $X_3 = \{(6,4),(6,3),(6,2),(7,3)\}$ and $X_4 = \{(6,3),(6,2)\}$.

On the other hand, when a set of IMPs for a set of users is given, *FindNIMP* finds the NIMPs with other users. Suppose we have a set of IMPs M and the corresponding costs for a set of users $U^{'}$. The cost of an IMP $m \in M$ is the total travel cost for all users to reach m. The cost of users in $U^{'}$ to travel to a node $m^{'}$ after meeting at m is the summation of the cost of m and $dist(m, m^{'})$. In our example, in Figure 3(a), if u_1 meets other users, the IMP must be a node in X_1. Similarly, an IMP for u_2 must be in X_2. Thus, if u_1 and u_2 meet at an IMP m, $m \in X_1 \cap X_2$. Hence, the set of possible IMPs for u_1 and u_2 is $M_{12} = X_1 \cap X_2 = \{(4,4)\}$, as in Figure 3(b). The cost of IMP (4,4) is its total distance from the two users and thus it is 2.

To compute the cost of the NIMPs, we create a virtual node I which has an edge connected to each meeting point $m \in M$. The cost of an edge is the cost of the connected m. We use I as source and POIs as destinations in *FindNIMP*. In our example, suppose u_1 and u_2 have met at one of the nodes in M_{12} and they travel further to meet other users. We create a virtual node I_{12} that is connected to each node in M_{12} as shown in Figure 3(c). The cost of an edge between $m \in M_{12}$ and I_{12} is the total distance of m from u_1 and u_2. These costs are in Table 1. Then, we run *FindNIMP* with I_{12} as source and the POIs as boundaries to find NIMPs for the combination of u_1 and u_2. The distance of m from I_{12} represents

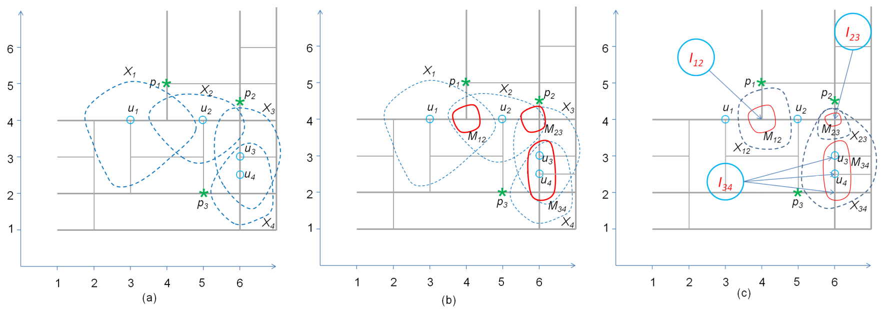

Figure 3: (a) NIMPs of the users (b) Intersections of each pair of sets of NIMPs (c) Computing NIMPs for each pair of users

the total travel cost of u_1 and u_2 to m when they ride-share. The nearest POI from I_{12} is p_1 whose distance is 3 from I_{12}.

Algorithm 1 finds the NIMP nodes

```
1: function FindNIMP($M$, $cost$, $B$, $G$)        ▷ $M$ : source nodes, $cost$ :
   source costs, $B$ : boundary nodes, $G$ : graph.
2:     create node set $Q$
3:     for each node $m$ in $M$ do
4:         Q.insert($m$, $cost[m]$)
5:         $J[m] \leftarrow m$
6:     $X \leftarrow \emptyset$
7:     while $Q$ is not empty do
8:         $u \leftarrow$ Q.ExtractMin()                    ▷ find the next node
9:         $X \leftarrow X \cup \{u\}$
10:        if $u \in B$ then
11:            $p \leftarrow u$
12:            $minCost \leftarrow cost[u]$
13:            break                ▷ terminate if a boundary node is found
14:        for each neighbor $v$ of $u$ that is still in Q do
15:            if $v$ not visited before then
16:                $cost[v] \leftarrow$ infinity
17:            $temp \leftarrow cost[u]$+length($u$, $v$)
18:            if $cost[v] > temp$ then
19:                $cost[v] \leftarrow temp$
20:                $J[v] \leftarrow J[u]$
21:                Q.insert($v$, $cost[v]$)
22:    return $X$, $cost$, $J$, $p$, $minCost$ ▷ $X$ : NIMPs, $cost$ : NIMPs costs,
   $J$ : meeting points, $p$ : nearest POI, $minCost$ : cost of $p$.
23: end function
```

When any node p from P is found, the algorithm ends. Due to the incremental property of *FindNIMP*, the cost of p is the minimum for any member of P. The costs of all nodes found before p are not larger than that of p. Let $X_{U'}$ be the set of these nodes. According to Theorem 4.1, $X_{U'}$ is the set of NIMPs where U' can meet other users from $U - U'$. Similarly, we can compute the NIMPs $X_{U''}$ of another set of users U'', as the set of the possible points where U'' can meet other users from $U - U''$. If $U' \cap U'' = \emptyset$, $X_{U'} \cap X_{U''}$ is the set of nodes where U' and U'' can meet.

Algorithm 1 finds NIMPs. Line 4 sets the cost of the nodes in M to mimic the case where the nodes are connected to a virtual node, which is the source. $J[v]$ records the node $m \in M$ which connects v with the virtual node in the shortest path. In other words, if the users travel to v they will meet at m and share a car to v.

Table 1: IMPs and costs for each pair of users

IMP set	Node	Total cost
$M_{12} = X_1 \cap X_2$	(4,4)	2
$M_{13} = X_1 \cap X_3$	$\emptyset$	-
$M_{14} = X_1 \cap X_4$	$\emptyset$	-
$M_{23} = X_2 \cap X_3$	(6,4)	2
$M_{24} = X_2 \cap X_4$	$\emptyset$	-
$M_{34} = X_3 \cap X_4$	(6,3)	0.5
"	(6,2.5)	0.5
"	(6,2)	1.5

Table 2: NIMPs and costs for each pair of two users

NIMP set	Node	Total cost
X_{12}	(4,4)	2
X_{13}	$\emptyset$	-
X_{14}	$\emptyset$	-
X_{23}	(6,4)	2
X_{24}	$\emptyset$	-
X_{34}	(6,4)	1.5
"	(6,3)	0.5
"	(6,2.5)	0.5
"	(6,2)	1.5
"	(7,3)	1.5
"	(7,2.5)	1.5

4.2.2 *Optimal Plan for Each Possible Subgroup.*

Suppose the number of seats in each car is three in the previous example. Thus, the size of a valid subgroup must be less or equal to three.

In the first step, we consider each user individually. In Figure 3(a), the nearest POI from u_1 is p_1 and the distance is 2. Thus, the optimal travel cost for u_1 is $OptCost_1 = 2$. If u_1 does not ride-share with others, she can go to p_1 by travelling 2 units. According to Theorem 1, u_1 will not travel more than 2 units to meet another user. The optimal travel cost for u_2, u_3 and u_4 when they do not ride-share are the distances from their nearest POIs. Therefore, $OptCost_2 = 1.5$, $OptCost_3 = 1.5$ and $OptCost_4 = 1.5$

In the second step, we consider each pair of users. For each pair of users we find the IMPs where they can meet each other and the NIMPs where they can meet other users. The set of IMPs for each pair of users is shown in Table 1. The optimal POI for u_1 and u_2 when they ride-share is p_1 and the travel cost is 3. Since $OptCost_1 + OptCost_2 = 2 + 1.5 > 3$, u_1 and u_2 should ride-share if possible and the cost of their travel plan is $OptCost_{12} = 3$.

According to Theorem 2, u_1 and u_2 will travel to m to meet other users if the distance of m from I_{12} is smaller than 3. Thus, the NIMP set of u_1 and u_2 is $X_{12} = \{(4,4)\}$ and the cost is 2. Similarly, we compute NIMP sets for each pair of users using *FindNIMP*. Table 2 shows the NIMPs and their costs for each pair of users. $X_{23} = \{(6,4)\}$, $X_{34} = \{(6,4), (6,3), (7,3), (6,2.5), (7,2.5), (6,2)\}$, $OptCost_{23} = 2.5$ and $OptCost_{34} = 2$. Since $M_{13} = M_{14} = M_{24} = \emptyset$, $X_{13} = X_{14} = X_{24} = \emptyset$, users do not ride-share in these cases, and the minimum cost of each pair of users to any POI is the total distance from their nearest POIs. Thus, $OptCost_{13} = Opt_1 + Opt_3 = 2 + 1.5 = 3.5$, $OptCost_{14} = 2 + 1.5 = 3.5$ and $OptCost_{24} = 1.5 + 1.5 = 3$.

In the third step, we consider the combination of three users starting from (u_1, u_2, u_3). The possible plans for the users to reach a POI p are the combinations where two of them meet first and then the other joins them to go to p. We compute the possible meeting points and their costs for all plans. Now, consider the first plan: u_2 and u_3 meet at an IMP $m_1 \in M_{23}$ and then proceeds to a NIMP $m_2 \in X_{23}$ to meet u_1. On the other side u_1 comes to a NIMP $m_2 \in X_1$ to meet u_2 and u_3. Thus, the set of possible meeting points where u_1 can meet u_2 and u_3 is $M_{1|23} = X_1 \cap X_{23}$. In our example, $M_{1|23} = \emptyset$ which means that u_1 will not meet u_2 and u_3 in an optimal travel plan. Hence, the optimal travel cost for the first plan is $OptCost_{1|23} = OptCost_1 + OptCost_{23} = 2 + 2.5 = 4.5$. Similarly, $M_{2|13} = X_2 \cap X_{13} = \emptyset$, $OptCost_{2|13} = OptCost_2 + OptCost_{13} = 2 + 2.5 = 4.5$, $M_{3|12} = X_3 \cap X_{12} = \emptyset$ and $OptCost_{3|12} = OptCost_3 + OptCost_{12} = 1.5 + 3 = 4.5$. Thus, the set of IMPs for the three users is $M_{123} = M_{1|23} \cup M_{2|13} \cup M_{3|12} = \emptyset$ and $OptCost_{123} = min(OptCost_{1|23}, OptCost_{2|13}, OptCost_{3|12}) = 4.5$. Since $M_{123} = \emptyset$, u_1, u_2 and u_3 will not form a subgroup and go to a single POI. Similarly, we compute $M_{234} = \{(6, 4)\}$ and the cost of the IMP is the total distance from u_2 and I_{34} which is 2.5. In this manner, we compute $M_{124} = \emptyset$ and $M_{134} = \emptyset$. Next, we compute NIMPs for each combination of three users. Since $M_{123} = M_{124} = M_{134} = \emptyset$, these combinations are not valid to form a subgroup and thus $X_{123} = X_{124} = X_{134} = \emptyset$. We create a virtual node I_{234} which is connected to the node in M_{234} with an edge of cost 2.5. Then, we run *FindNIMP* considering I_{234} as source and the POIs as boundary. The nearest POI from I_{234} is p_2 and its distance is 3. Thus, the optimal POI for u_2, u_3 and u_4 is p_2 and the cost is $OptCost_{234} = 3$.

Algorithm to compute travel plan for each possible subgroup: We use dynamic programming to compute the optimal travel plan to the optimal POIs for all possible combinations where each combination can fit in a car. Although this algorithm resembles Dreyfus et al.'s, it has two crucial differences. First, the new constraints on the NIMPs reduce significantly the time complexity when computing the costs of candidate meeting points. Most nodes are not even taken into account. In Algorithm 2, the "for loop" in lines 17 through 22 only iterates IMPs resulting from the intersection of NIMPs of subgroups of users, whereas, in the original algorithm, every single node in the network is taken into account. Second, our new Steiner problem does not have a defined set of terminals, i.e., users and POIs. POIs are chosen by the algorithm.

Algorithm 2 finds the optimal travel plan and the optimal POIs for each combination of users where the number of users in each combination is less or equal to z. *FindNIMP* computes a list of NIMPs $X[comb]$ for users in *comb* where $cost[comb]$ is a list of the cost values of each node in $X[comb]$. For each node x in $X[comb]$ there is a node $j \in M[comb]$ listed in $J[comb]$. If the users in *comb* go to x they meet in j and then they ride-share to x.

In line 3, we compute the NIMPs for each user using *FindNIMP*. The function also computes the nearest POI and its distance from each user and records them in $OptPOI[u]$ and $OptCost[u]$. Next, we increase the number of users in each combination *comb* in the "for loop" in line 4. Line 6 to 10 initializes some variables which are used later. For each combination we consider each possible division into two subgroups $comb_1$ and $comb_2$ in the "for loop" of line 11. If the total cost decreases when $comb_1$ and $comb_2$ go to their respective optimal POIs, we update *OptCost* and *bestSubgrouping* in lines 12 to 15. We compute meeting points for each division by taking an intersection of the NIMPs of $comb_1$ and $comb_2$ in line 16. The "for loop" in line 17 iterates each node that is a NIMP of both $comb_1$ and $comb_2$. These are IMPs where $comb_1$ and $comb_2$ can meet. The cost of the IMPs are computed and updated in lines 19-21. Line 22 records the best division of *comb* into $comb_1$ and $comb_2$ for each node. We need this information to recover the optimal travel plan. After computing the IMPs and their costs for *comb*, *FindNIMP* in line 23 computes the NIMPs for *comb*. The function also computes the optimal POI if all users in *comb* ride-share to a common POI and the cost to the POI. If the cost is less than the previously computed $OptCost[comb]$, then *OptCost*, *OptPOI* and *bestSubgrouping* are updated in lines 24-27. Then, the "for loop" of line 5 considers the next combination. If all combinations are considered, then the size of the combination is increased. After considering all combinations of z users, the algorithm terminates.

Algorithm 2 computes optimal travel plans for all possible combinations of less than z users

```
1:  function ComputeCostEachCombination($U'$, $P$, $G$)  ▹ $U'$ : users, $P$ : POIs, $G$ : graph.
2:      for each user $u \in U'$ do
3:          $X[u]$, $cost[u]$, $J[u]$, $OptPOI[u]$, $OptCost[u]$ ← FindNIMP($u$, 0, $P$, $G$)
4:      for $j$ = 2 to $z$ do
5:          for each combination $comb$ of $j$ users from $U'$ do
6:              $M[comb] \leftarrow \emptyset$
7:              $OptCost[comb] \leftarrow$ inf
8:              $OptPOI[comb] \leftarrow null$
9:              for each $v \in V$ do
10:                 $cost[comb][v] \leftarrow$infinity
11:             for each division of $comb$ to $comb_1$ and $comb_2$ do
12:                 $temp \leftarrow OptCost[comb_1] + OptCost[comb_2]$
13:                 if $OptCost[comb] > temp$ then
14:                     $OptCost[comb] \leftarrow temp$
15:                     $bestSubgrouping[comb] \leftarrow [comb_1, comb_2]$
16:                 $XX \leftarrow X[comb_1] \cap X[comb_2]$
17:                 for each $m \in XX$ do
18:                     $M[comb] \leftarrow M[comb] \cup m$
19:                     $temp = cost[comb_1][m] + cost[comb_2][m]$
20:                     if $cost[comb][m] > temp$ then
21:                         $cost[comb][m] \leftarrow temp$
22:                         $bestDiv[comb][m] \leftarrow [comb_1, comb_2]$
23:             $X[comb]$, $cost[comb]$, $J[comb]$, $p$, $temp$ ← FindNIMP($M[comb]$, $cost[comb]$, $P$, $G$)
24:             if $OptCost[comb] > temp$ then
25:                 $OptCost[comb] \leftarrow temp$
26:                 $OptPOI[comb] \leftarrow p$
27:                 $bestSubgrouping[comb] \leftarrow [comb, null]$
28:     return $OptCost$, $OptPOI$, $J$, $bestDiv$, $bestSubgrouping$  ▹ $OptCost$ : optimal travel cost, $OptPOI$ : optimal POI for each combination, $\{J, bestDiv, bestSubgrouping\}$ : travel plan.
29: end function
```

Table 3: Division of a group of users into two sets

Division	Total cost
u_1, (u_2, u_3, u_4)	$Opt_1 + Opt_{234} = 2 + 3 = 5$
u_2, (u_1, u_3, u_4)	$Opt_2 + Opt_{134} = 1.5 + 4 = 5.5$
u_3, (u_1, u_2, u_4)	$Opt_3 + Opt_{124} = 1.5 + 4.5 = 6$
u_4, (u_1, u_2, u_3)	$Opt_4 + Opt_{123} = 1.5 + 4.5 = 6$
(u_1, u_2), (u_3, u_4)	$Opt_{12} + Opt_{34} = 3 + 2 = 5$
(u_1, u_3), (u_2, u_4)	$Opt_{13} + Opt_{24} = 3.5 + 3 = 6.5$
(u_1, u_4), (u_2, u_3)	$Opt_{14} + Opt_{23} = 3.5 + 2.5 = 6$

We recover the optimal travel plan in a recursive manner. Let p be $OptPOI[comb]$ for a given set of users $comb$. Let m be $J[comb][p]$. m records the node where all users in $comb$ meet before going to p. $bestDiv[comb][m]$ records the optimal division of users into two subsets $comb_1$ and $comb_2$ when $comb_1$ and $comb_2$ meet at m. Let m_1 be $J[comb_1][m]$. m_1 records the node where the users in $comb_1$ meet before going to m. $bestDiv[comb_1][m_1]$ records the two subgroups which meet at m_1 and forms $comb_1$. In this way, we find the meeting points and divisions of users prior to coming to the meeting points.

Now, let $OptPOI[comb]$ be null. In this case, the users in $comb$ form multiple subgroups and the optimal travel plan for each subgroup needs to be recovered separately. $bestSubgrouping[comb]$ records the best division of $comb$ into two subgroups $comb_1$ and $comb_2$. If $OptPOI[comb_1]$ is not null, we recover the optimal travel plan for $comb_1$ using the method described in the previous paragraph. If $OptPOI[comb_1]$ is null, like the case of $comb$, we find the best division of $comb_1$ into two subgroups. This process continues recursively until a combination is found for which $OptPOI$ is not null. In this way, we recover the optimal travel plan.

4.2.3 *Dividing a Group of Users into Subgroups.*

In Section 4.2.2, we computed optimal travel plans for all possible combinations of users where each combination can fit in a car. Table 3 shows all possible divisions of all users into two subgroups, where the first and the fifth divisions minimize the cost and thus either of them can be used to compute the optimal plan.

In this section we propose an algorithm to divide a group of users into subgroups so that each subgroup can fit in a car and the total travel cost is minimized. At first, we consider each combination of $z + 1$ users. Since the users cannot fit into a car, we consider all possible divisions of $z + 1$ users into two sets where the size of each set is less or equal to z. For each division, we compute the travel cost by adding the optimal travel cost of the two sets. The travel cost of all possible sets of size less or equal to z have been computed previously by Algorithm 2. The division with the minimum cost is the optimal division and the cost is the optimal cost for this particular combination of $z + 1$ users. Similarly, we compute the optimal cost for each combination of $z + 1$ users. Next, we consider all combinations of $z + 2$ users and divide them into two sets where the size of each set is less or equal to $z + 1$. Similar to the combination of $z + 1$ users, we compute the optimal cost of all combinations of $z + 2$ users. In this way, we keep on increasing the number of users until all users are included in a combination and the optimal cost for all users in a group is computed.

Algorithm 3 computes the optimal division of the users into subgroups. The "for loop" in line 2 iterates from $z + 1$ to S. The "for loop" in line 3 iterates to compute the optimal cost for each combination given a particular size. The "for loop" in line 5 iterates the possible divisions into two subgroups of the current combination of users to find the optimal division. The optimal cost and optimal division into subgroups are updated in lines 6-9. After computing the optimal travel cost for a combination of users, the "for loop" of line 3 considers the next combination. If all combinations of a certain size are considered, the size is increased in the "for loop" of line 2. When the size reaches to S the algorithm terminates.

Algorithm 3 computes optimal division of users into subgroups.

```
1:  function ComputeOptimalDivision($U'$, $OptCost$)    ▷ $U'$ : users.
2:      for $j = z + 1$ to $S$ do
3:          for each combination $comb$ of $j$ users from $U'$ do
4:              $OptCost[comb] \leftarrow$ inf
5:              for each division of $comb$ to $comb_1$ and $comb_2$ do
6:                  $temp \leftarrow OptCost[comb_1] + OptCost[comb_2]$
7:                  if $OptCost[comb] > temp$ then
8:                      $OptCost[comb] \leftarrow temp$
9:                      $bestSubgrouping[comb] \leftarrow [comb_1, comb_2]$
10:     return $OptCost$, $bestSubgrouping$    ▷ $OptCost$ : updated
        optimal travel cost, $bestSubgrouping$ : optimal division.
11: end function
```

4.2.4 *Dividing All Users into Groups .*

We divide all users into groups so that the optimal travel plan for each group can be computed using Algorithm 3. Suppose we choose S to be the maximum size of a group. Intuitively, a larger S will produce fewer divisions and a better approximation. However, the computation time of Algorithm 3 is exponential to S.

We compute the Voronoi cell of POIs for each user. Then, we randomly select a POI p_1. Let U_1 be the set of users whose nearest POI is p_1. We find the furthest user $u_1 \in U_1$. If the number of users in U_1 is smaller or equal to S, we start a new group Gr_1 and assign all users to it. If $|U_1| > S$, we find the nearest $S - 1$ users to u_1 who are members of U_1 and complete Gr_1. Next, we find the furthest user u_1' from p_1 from the remaining users in U_1. Similar to u_1, u_1' forms a group with the remaining users in U_1. In this way, we divide the users in U_1 into groups. If a group is partially filled, we check if all users in a neighboring cell can fit into the group and we assign these users to the group.

4.2.5 *Complexity Analysis.*

Let N_u and N be the total number of users and the number of nodes in the network. Let S and z be the number of users in each group and the capacity of the cars. The worst case computation time complexities of the first and second steps are $O(N_uN^2)$ and $O\Big(\frac{N_u}{S}(S^zN^2 + 3^SN)\Big)$.

The time complexity of Dreyfus et al.'s algorithm is $O(2^{N_u}N^2 + 3^{N_u}N)$. In our algorithm, if $S = N_u$, i.e., there is just one group which contains all users, and z is big enough, its time complexity is comparable to Dreyfus et al.'s, i.e., it leads to combinatorial explosion. However, we show experimentally that the cost of the travel plan is stable when the group size is larger or equal to 8 and the capacity of a car is 5. Thus, $S = N_u$ does not hold when N_u is big enough. Also, the inclusion of the constraints in our algorithm yields to the term S^zN^2 instead of $2^{N_u}N^2$ as in Dreyfus et al.'s.

Consequently, in practice, our algortihm's performance is much better than the solutions based on Dreyfus et al.

4.3 Extensions

In real life, some users may not have a car and users might be allowed to leave their cars at designated parking lots only.

The users who do not have a car must be picked up from their locations. Suppose u_w has a car and u_{wo} does not. If u_w and u_{wo} meet and share a ride, the IMP must be the location of u_{wo}. Distance from u_{wo} to any node other than her own location is infinity since she cannot travel. Two users who do not have car cannot meet by their own. We modify *FindNIMP* so that the NIMP of a user who does not have a car is her own location.

If a user is allowed to leave her car at a parking lot only, other users must pick her up from that point. Thus, an intermediary meeting point must be a parking lot. First, we modify the road network by adding the parking lots as nodes where the connecting roads link them to the neighboring nodes. Line 9 in *FindNIMP*, Algorithm 1, can exclude the nodes that are not parking lots.

5 EXPERIMENTAL EVALUATION

As the brute force approach will not scale even for a small number of users and POIs, we propose two baseline methods.

The first approximate solution divides all users into groups using their locations and computes an optimal travel plan for each group using Dreyfus et al.'s algorithm. The size of each group is determined by the computational limit of the hardware. Although this baseline solution is better than the brute force approach, computing the cost for each subset of users is still costly. When the network is large, both time and memory consumption will be prohibitive. Our experiments show that this method requires about 200 seconds for 32 users when the number of intersections in the network is 10,000 and the group size is 8. Thus, we need a better solution.

The second approximate solution divides the users into subgroups using a heuristic so that each subgroup can fit into a car and go to a pre-chosen POI. Then, for each subgroup we compute the MST using Dreyfus et al.'s algorithm. This solution avoids the costly set cover problem and also minimizes the number of MST computations. We draw a Voronoi diagram for the POIs, the users in a Voronoi cell are grouped together. If the number of users is large and cannot fit in a car, we divide the cell into subgroups so that each subgroup can fit. We give priority to the farthest users to a POI to ride-share as they will be able to share a longer route with others and reduce the travel cost more compared to nearer users.

We refer to the first baseline method as Dreyfus-WC as a short form of "Dreyfus with choice" of POI. The second baseline method is referred as Dreyfus-WoC as a short form of "Dreyfus without choice" of POI. In order to verify the effect of the constraint on the IMPs on the computation time we present results when constraints are not used in VST-RS. VST-WoConst represents these results.

We used and generated a real and synthetic road networks. We use the road network of California[4] which has about 2 million nodes and 3 million edges. In each run, we randomly selected part of the network to consider different types of cities. We generated a grid-like road network where the length of each edge was set randomly

[4]https://snap.stanford.edu/data/roadNet-CA.html.

Table 4: Experimental set-up

Parameter	Range	Default
Number of users	2 - 1024	128
Number of POIs	10- 640	160
Network size	1250 - 80,000	10,000

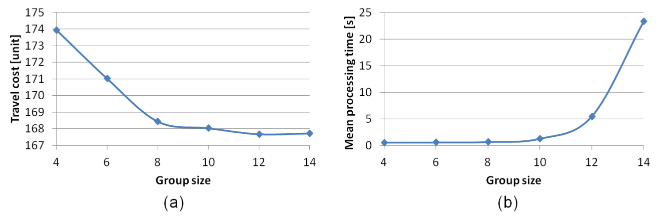

Figure 4: Effect of group size on (a) travel cost of the computed plan, (b) mean processing time of VST-RS.

using a uniform distribution between 0 and 1. The locations of users and POIs had a uniform random distribution. We also considered the special case when the POIs have a Zipfian distribution. In this case, there was a grid-like city and the POIs were located at one side of the city. Lengths of all edges were equal.

We varied the number of users from 2 to 1024. The number of POIs were varied from 10 to 640. We varied the network size from 1250 to 80000 nodes. In the default setup, there are 128 users and 160 POIs. If the system computes ride-sharing arrangements every two minutes, it will serve 3840 users per hour in the default case. If a POI serves an area of 10 km^2, 160 POIs serve a city of 1600 km^2. Since most passenger cars have comfortable seating arrangements for 4 passengers we take the default value of the number of users as 4. The baseline uses Dreyfus' MST algorithm and it could not compute an MST when the network size was larger than 10K. Therefore, the default value of the network size was set to 10K. We used the synthetic network as default to have better control on the network. The range of the parameters and their default values are shown in Table 4. We computed the mean processing time for all methods. For each experiment we processed 100 queries.

5.1 Effect of Group Size

We divide all users into groups and for each group we compute the optimal travel plan. Since we use heuristics to divide users into groups, VST-RS gives an approximate answer of the overall set cover problem. Intuitively, the larger and fewer the groups are, the better the approximation is. We studied the effect of group size on the cost of the computed travel plan by varying the number of users in a group. Figure 4(a) shows that the travel cost of the computed travel plan decreases at the beginning rapidly and then becomes stable when the number of users in a group increases. Since the users who are far away in space have little effect on the travel plan, the effect of the group size diminishes gradually when it increases. From Figure 4 we can say that the travel cost is stable when the group size is larger or equal to 8. Figure 4(b) shows that the mean processing time increases exponentially when the group size increases. Considering both travel cost and processing time, 8 -

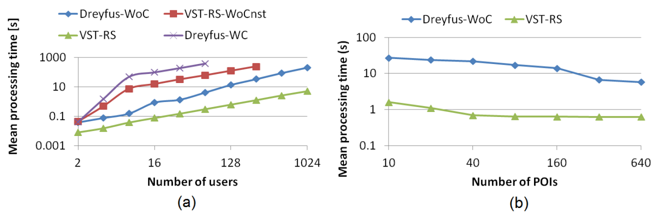

Figure 5: (a) Effect of number of users, and (b) Effect of number of POIs, on the mean processing time.

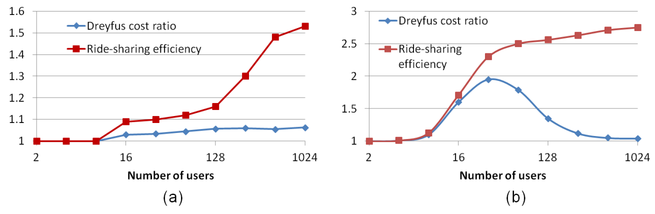

Figure 6: Effect of number of users on ride-sharing efficiency and the Dreyfus cost ratio when POIs have (a) uniform random distribution (b) Zipfian distribution.

10 is a good choice as the maximum group size. Thus, the maximum group size in VST-RS will be 8.

5.2 Effect of the Number of Users

We varied the number of users from 2 to 1024. Figure 5(a) shows that VST-RS is at least one order of magnitude faster than the other methods. The gap increases when the number of users increases. In our case, when the number of users increases, the distance between the users of a same group decreases. As a result, *FindNIMP* requires less effort to find the possible meeting points. Thus, the computation time of VST-RS does not increase as fast as that of the baselines. VST-WoCnst and Dreyfus-WC require prohibitive computation time, so we do not study them any further.

Figure 6(a) shows the *Dreyfus cost ratio* and *ride-sharing efficiency* when the POIs are uniformly distributed. *Dreyfus cost ratio* is the ratio between the costs of the travel plans computed by Drefus-WoC and VST-RS. It highlights the benefit on cost savings of having a choice set of POIs rather than going to a pre-defined destination. *Ride-sharing efficiency* is the ratio between the total cost when the users go to their nearest POIs individually, i.e., they do not ride-share, and the total cost computed by VST-RS. This value indicates the savings due to ride-sharing.

When there are only a few users, the users go to their nearest POIs individually since the users are far away from each other and they have to travel long distances to share rides. As a result, when the number of users is less or equal to 8, both Dreyfus cost ratio and ride-sharing efficiency is one. As the number of users increases, the locations of the users become close and the ride-sharing opportunity increases. As a result, the ride-sharing efficiency increases when the number of users increases. The Dreyfus cost ratio was stable around 1.05 indicating that the travel cost of the travel plan computed by the baseline was on average about 5% higher than that of VST-RS.

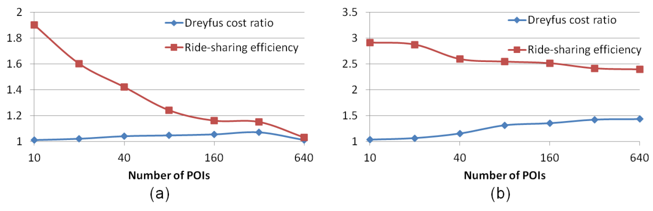

Figure 7: Effect of number of POIs on the Dreyfus cost ratio and ride-sharing efficiency when POIs have (a) uniform random distribution (b) Zipfian distribution.

However, this is true only if POIs are distributed uniformly and there is a POI close to ride-sharers.

Figure 6(b) shows the Dreyfus cost ratio and ride-sharing efficiency when the POIs are Zipfian distributed. Figure 6(b) shows that the cost of the travel plan computed by VST-RS is much smaller than that of the baseline method when the number of users is between 16 and 128. In this case, the number of users in most Voronoi cells are one or two and the users of a neighboring cell can decrease the cost a lot by sharing rides and the destination POI is far away at the other side of the city. The overall ride-sharing efficiency is also higher in this scenario.

5.3 Effect of the Number of POIs

We varied the number of POIs from 10 to 640. The distance from a user to its nearest POI decreases if the number of POIs increases. As a result, our algorithm searches small areas to find the possible meeting points of users. Thus, the processing time of VST-RS decreases which is shown in Figure 5(b). The processing time of the baseline method also decreases. When the number of POIs increases the mean size of the Voronoi cells and the number of users in each of them decrease. The computation time of the MST algorithm is exponential to the number of users. Therefore, a small group requires much less computation time than a large group. Although the number of groups increases, due to smaller group size the computation time of the baseline decreases.

Figure 7(a) shows the effect of number of POIs on the Dreyfus cost ratio and ride-sharing efficiency when the locations of POIs have uniform random distribution. If the number of POIs increases, the average distance from a user to its nearest POI decreases and therefore the benefit from ride-sharing (ride-sharing efficiency) decreases. When there are 128 users and 640 POIs, almost all of the users go to their nearest POI independently without sharing rides. Thus, in this case, the ride-sharing efficiency is close to one. Similarly, Dreyfus cost ratio is also close to one in this case. When the number of POIs is much less than the number of users, i.e., for 10 and 20 POIs, each Voronoi cell has a good number of users and there is enough ride-sharing opportunity in each cell. Thus, the baseline method, which groups users based on Voronoi cells, can compute good ride-sharing plans. For the remaining cases, i.e., 40 to 320 POIs, the travel plan computed by the baseline method has about 4 ∼ 5% higher cost than that of VST-RS.

Figure 7(b) shows the effect of the number of POIs on the Dreyfus cost ratio and ride-sharing efficiency when the locations of POIs have Zipfian distribution. This setup represents the scenario

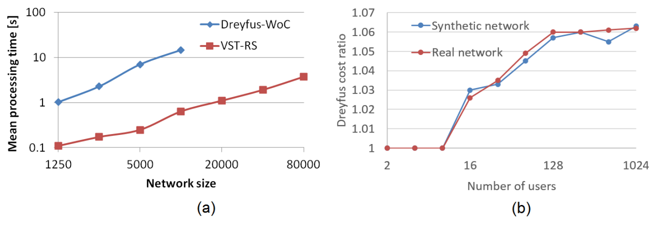

Figure 8: (a) Mean processing time of methods for different network sizes, (b) Real vs Synthetic networks.

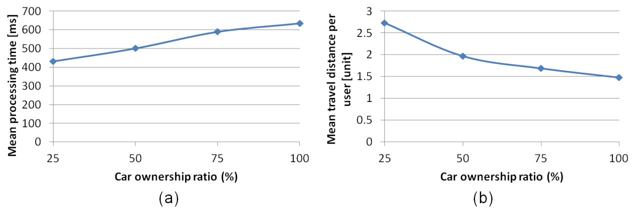

Figure 9: Effect of the number of car owners on (a) mean processing time, (b) travel cost.

where the POIs are located at one side of the city. The cost of the travel plan computed by VST-RS is much smaller than that of the baseline when the number of POIs is more than 40. In this case, the number of users in most Voronoi cells are one or two and the users of neighboring cells can decrease the cost a lot by sharing rides since the destination POI is far away at the other side of the city.

5.4 Effect of the Network Size

We varied the number of nodes in the synthetic network from 1250 to 80K. Figure 8(a) shows the mean processing time of VST-RS and the baseline method. VST-RS outperformed the baseline method by one order of magnitude. The baseline could not compute the optimal POI when the number of nodes was 20K or above.

5.5 Real vs Synthetic Networks

Figure 8(b) shows that the Dreyfus cost ratio for real and synthetic network was always close when the number of users increased from 2 to 1024. The computation time for both real and synthetic networks was also similar when we randomly select parts of the real network. The parts selected from the real network were of the same size as the synthetic counterparts.

5.6 Experiments on Extensions

The number of car owners was varied to study the effect of car ownership ratio. At first we considered car ownership ratio 100%. Then, we decreased it in three steps until 25%. If a user does not have a car she must be picked up from her location and thus, she has only one intermediary meeting point which is her location. Therefore, our algorithm checks less number of possible meeting points and the processing time decreases as shown in Figure 9(a). Figure 9(b) shows that the total travel distance increases when the number of car owners decreases.

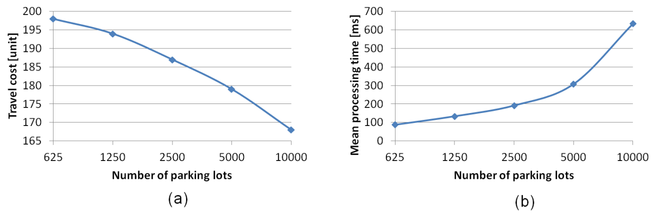

Figure 10: Effect of the number of parking lots on (a) travel cost, (b) mean processing time.

We studied the effect of the number of parking lots by increasing it from 625 to 10K. Figure 10(a) shows that when the number of parking lots increases, the processing time increases. Since more parking lot creates more options for intermediary meeting points, our algorithm needs to consider more options and thus the processing time increases. Figure 10(b) shows that the travel cost decreases when the number of parking lot increases. When there are fewer parking spots, the number of possible meeting points is less.

6 CONCLUSION

In this paper, we develop an enhanced ride-sharing system for a city to help users finding better ride-sharing arrangements based on agreements on POIs rather than matching pre-computed trajectories. We propose an efficient dynamic programming algorithm and a novel solution to a new Steiner tree problem.

Our experimental results show that our proposed method outperforms the baseline method, which is based on Dreyfus et al.'s MST algorithm, by two orders of magnitude. The processing time of VST-RS is small enough to be deployable in a real world scenario. In this paper, we also extended our proposed solution for two different variations of the problem, i.e., when users do not have a car and must meet at designated parking lots only. In the future, we plan to consider other variations of the problem such as, different car capacities to combine car travellers with van travellers, time constraints on travel periods, ride-sharing preferences based on spatial and non-spatial attributes, and using multiple hops, i.e., users may be dropped off and then picked up by another car later on.

REFERENCES

[1] R. Baldacci, V. Maniezzo, and A. Mingozzi. An exact method for the car pooling problem based on lagrangean column generation. *Operations Research*, 52(3):422–439, 2004.
[2] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1972.
[3] Y. Huang, F. Bastani, R. Jin, and X. S. Wang. Large scale real-time ridesharing with service guarantee on road networks. *VLDB*, 7(14):2017–2028, 2014.
[4] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*, pages 410–421, 2013.
[5] S. Ma, Y. Zheng, and O. Wolfson. Real-time city-scale taxi ridesharing. *Knowledge and Data Engineering, IEEE Transactions on*, 27(7):1782–1795, 2015.
[6] D. Mölle, S. Richter, and P. Rossmanith. A faster algorithm for the Steiner tree problem. *Lecture notes in computer science*, pages 561–570, 2006.
[7] M. Olsen. On the complexity of computing optimal private park-and-ride plans. In *Computational Logistics*, pages 73–82. 2013.
[8] Y. Wang, R. J. Kutadinata, and S. Winter. Activity-based ridesharing: increasing flexibility by time geography. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2016, Burlingame, California, USA, October 31 - November 3, 2016*, pages 1:1–1:10, 2016.