From Ride-Sourcing to Ride-Sharing through Hot-Spots

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

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

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

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

ABSTRACT

Smartphones have allowed us to make ad-hoc travel arrangements. Ride-sharing is emerging as one of the new types of transportation enabled by smartphone revolution. Ride-sharing aims to alleviate current environmental, social and economical issues many big cities are facing due to low vehicle occupancy rates. Although ride-sharing companies have millions of users around the world, some of them do not offer true ride-sharing but a similar service called *ride-sourcing* where private car owners provide for-hire rides. Ride-sharing uptake has not been wide due to lack of convenience and incentives. We propose an enhanced ride-sharing model through the inclusion of proper places to meet that we call *hot-spots*. Hot-spots are shown to increase the convenience by solving the round-trip ride-sharing problem. As we represent our enhanced model through graphs, we introduce a new graph problem that we call *Constrained Variable Steiner Tree*, which is NP-hard. An effective and readily deployable heuristic solution to this problem is presented which is up to two orders of magnitude faster than the state-of-the-art solution as combinatorial explosion is avoided by the usage of a novel monotonic nondecreasing function.

CCS CONCEPTS

• **Applied computing** → **Transportation**;

KEYWORDS

Ride-sharing, Computational Transportation Science, Spatial Databases

ACM Reference format:
Oscar Correa, Kotagiri Ramamohanarao, Egemen Tanin, and Lars Kulik. 2017. From Ride-Sourcing to Ride-Sharing through Hot-Spots. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Melbourne, VIC, Australia, November 7–10, 2017 (MobiQuitous 2017),* 10 pages.
DOI: 10.1145/3144457.3144483

1 INTRODUCTION

Smartphones have fundamentally changed the way we do things in our daily lives. One of these changes is the way we approach transportation. The ubiquitous nature of access to services and data via smartphones has paved the way to ride-sharing as an ad-hoc form of carpooling as transportation.

Although ride-sharing companies have millions of users around the world, many of them do not offer true ride-sharing services. Instead, these companies offer a similar mobility service called **ride-sourcing** where private car owners provide for-hire rides. In ride-sourcing, car owners do not drive their passengers as part of their trips, thus ride-sharing benefits vanish.

Chaube et al. in [3] conclude that people would participate more in ride-sharing if issues such as convenience, incentives and trust are resolved. This paper proposes to enhance ride-sharing and make it more convenient, attractive and safer by the inclusion of special meeting points which we call **hot-spots**. We envision hot-spots as places where users find it easier to leave their cars and take someone else's car for shared travel. These places may be equipped with video surveillance and other facilities so that hot-spots can be thought as a parallel infrastructure that is similar to train stations but designed for ride-sharing.

Users would drive to hot-spots and leave their cars there instead of either (a) meeting at other places (to which we refer as *model A*) where finding a parking spot is perhaps difficult, or (b) being picked up from their initial locations such as their home address (to which we refer as *model B*) revealing their locations to strangers and causing larger detours to drivers.

However, we argue that hot-spots are also important because round-trip ride-sharing arrangements are significantly more likely than in the abovementioned ride-sharing models, i.e., models A and B. As hot-spots are a subset of the set of candidate meeting points in a road network, more people start their travel from a hot-spot than from a normal meeting point. Consequently, there may be higher demand to come back to hot-spots than to normal spots. This provides a higher certainty to ride-sharers regarding their return trips. Our scheme with hot-spots not only makes ride-sharing (a) convenient, e.g., round-trips are more likely, users can drive their cars as they will find parking and drivers do not need to make large detours, (b) attractive, e.g., hot-spots gather more people thus drivers will find more people who share costs with, and (c) trustworthy, e.g., privacy is achieved as users do not need to provide precise home/work locations and safety is increased through video surveillance; but also keeps a comparable car occupancy rate to model A, which in turn has been proved to increase the occupancy rate compared with model B [2, 5, 9].

Our proposed model can be built on top of the classical paradigm where the origins and destinations of the users are established, namely *trip-based* ride-sharing. Nonetheless, since a more recent

MobiQuitous 2017, Melbourne, VIC, Australia

DOI: 10.1145/3144457.3144483

type of ride-sharing called *activity-based* ride-sharing increases the matching rate [8, 11], we decided to build our model based on this paradigm. In activity-based ride-sharing, places are seen not only as geographical locations, but also as venues where people perform activities so that users can be offered a set of destination choices to ride-share to. Therefore, we argue that an activity-based ride-sharing scheme where the meeting points are constrained to be hot-spots only, can be even more convenient.

An activity-based ride-sharing problem instance consists of a set of users who ride-share to any of the points of interest (POIs) where they can perform a common activity, e.g., shopping. Thus, an approach to solve this problem instance would be first, to find an optimal division of the set of users into subsets who ride-share to their most convenient common POI; and second, to find an optimal shared route to this POI for the users in each subset. Rahman et al. in [8], proposed such an approach. The authors observed that if the road network is modelled as a graph and the users, POIs and meeting points are vertices in the graph, the shared route for each subset of users would be a Minimum Steiner Tree (MST[1]). In each MST, the leaves are the users (terminals), the intermediate nodes are the meeting points (Steiner vertices) and the root is the POI. That is, Rahman et al. modelled the solution to an activity-based ride-sharing problem instance as a forest of MSTs. The authors call this new problem on graphs the *Variable Steiner Tree problem* (VST), which is different from the classical Minimum Directed Steiner Forest (MDSF) because the roots in each MST are chosen by Rahman et al.'s algorithm, whereas in the MDSF problem disjoint subsets of terminals with their corresponding roots are given.

In this paper, we include a new constraint on the feasible set of Steiner vertices, i.e., meeting points must be hot-spots only. We redefine VST as *Constrained Variable Steiner Tree problem* (C-VST) and we show that even with such a constraint, this problem is NP-hard. Therefore, we introduce an approximate solution which maximises a heuristic set function that we call *Gain ratio*. This function is associated with a candidate Steiner vertex and the vertices that may connect to it. The larger its value, the more attractive such Steiner vertex is for meeting. Since the size of the domain L of this function, i.e., the set of vertices that may connect to it, is 2^L, combinatorial explosion occurs when it is maximized naïvely. We show that our heuristic function is nondecreasing under a specific update rule. Such update rule maximizes our heuristic function in linear time with respect to the number of users (terminals) for each candidate Steiner vertex. Thus, we construct a nearly optimal solution to the C-VST problem that can be deployed as a real-time service.

Our research hypothesis is that, our approximate solution to the C-VST problem is more efficient and even more effective, under realistic city conditions, than the state-of-the-art approach presented by Rahman et al. in [8], while, at the same time our proposed ride-sharing model makes round-trips significantly more likely and keeps car occupancy rates comparable to model A.

Our contributions are summarized as follows:

- We propose a new ride-sharing scheme constrained to hot-spots that increases convenience, incentives and trust.
- We define a new class of problem on graphs called *Constrained Variable Steiner Tree problem* (C-VST) and prove it is NP-hard.
- For the C-VST problem, we present a fast approximate algorithm that allows the development of a readily deployable solution for activity-based ride-sharing with hot-spots. A monotonic nondecreasing function that constructs nearly-optimal Steiner trees without combinatorial explosion is introduced.
- We show empirically that our solution is more efficient (up to two orders of magnitude faster) and more effective than the state-of-the-art solution introduced by Rahman et al. [8].

The rest of the paper is organised as following: Section 2 defines the problem and proves it is NP-hard. Section 3 presents related work. Section 4 shows the state-of-the-art and our solution to the C-VST problem. Experiments and results are presented in Section 5. The paper ends in Section 6 with discussions and conclusions.

2 PROBLEM DEFINITION

Trip-based ride-sharing systems require users to inform about their origins and destinations. In our activity-based scheme, users submit their requests informing their origins and the activity they are willing to carry out. An instance of the C-VST problem is formed per activity and contains users and one or more POIs where such activity can be performed. The system can then compute an optimal travel plan for each instance in parallel as inter-instance arrangements are not possible (users in different instances perform different activities). In a C-VST problem instance, users are also required to meet at hot-spots. Once users have met at a hot-spot, the system determines either they can ride-share to another hot-spot to meet with more users or they must go to the optimal POI.

Let a directed graph $G = (V, A, c)$ represent the road network where the road intersections are the set of vertices V, the segments between two intersections are the set of arcs A and the cost function $c : A \rightarrow \mathbb{R}_+$ is defined as $c(A) = \sum_{a \in A}[\text{cost of arc } a]$. Since users, POIs and hot-spots can be located across the segments between road intersections, for the sake of simplicity and without loss of generality, we assign their locations to the closest intersections. If U, P and H correspond to the set of user, POI and hot-spot locations, respectively, then, $U \subset V$, $P \subset V$ and $H \subset V$. Therefore, an instance of the C-VST problem can be defined as:

Given: A directed network $G = (V, A, c)$, the set of user locations U, the set of POI locations P and the set of hot-spots H.

Find: An MDSF where the Steiner trees have as leaves disjoint subsets of U, the Steiner vertices are a subset of H, and each tree T has as its root $p = \arg\min_{p' \in P} c(T)$.

Since C-VST is a generalization of the classical MDSF problem and no special graph is considered, C-VST is NP-hard even with the additional constraint on the feasible set of Steiner vertices. To prove that C-VST is NP-hard, we introduce the decision version of this problem, namely *kC-VST*. In this version, the algorithm must determine if a graph contains a C-VST with cost of at most k.

Theorem 2.1. *kC-VST is NP-complete.*

Proof. To prove that kC-VST is NP-complete we must prove that it is in NP and there exists a polynomial time reduction from

[1]We abbreviate this problem as MST. Readers should not be confused with the minimum spanning tree problem.

an NP-complete problem. As the decision version of MDSF, that is, the directed Steiner Forest DSF with cost at most k is NP-complete, we construct a reduction from an instance of this problem.

The first part is straightforward because we only need a certificate *cert* of cost k'. A polynomial time algorithm can verify whether (a) every terminal in *cert* is connected with at most one root, (b) $k' \leq k$, and (c) the Steiner vertices only belong to H.

To prove the second part, we build a reduction from a graph $G = (V, A)$. In this reduction, we construct a metric closure of G, namely $G' = (V', A')$, induced by the vertices in $U \subset V$, $P \subset V$ and $H \subset V$, that is, $V' = U \cup P \cup H$. A' corresponds to the set of shortest paths in G between every pair of vertices in V'. To prove this reduction works, we must show G' contains a DSF of cost at most k if and only if G contains a kC-VST of cost at most k.

($\Rightarrow$) If G' contains a DSF F' of cost at most k, each arc (v, w) of F' corresponds to the shortest path between vertices v and w in G which altogether form a forest F in G. F will have Steiner vertices that belong to H only, as F' does not include other vertices. $c(F) = c(F') \leq k$ because the cost of each arc of F' is the minimum pair cost (see Definition 4.1) and each arc of F is the shortest path between the same vertices. Hence, F is a kC-VST in G.

($\Leftarrow$) If G contains a kC-VST F of cost at most k, every pair of vertices that are joined in F will have its corresponding arc in G' as G' is complete. These arcs will form a forest F' in G' whose cost $c(F') = c(F) <= k$. Hence, F' is a DSF in G'.

Therefore, kC-VST is NP-complete. □

Corollary 2.2. *C-VST is NP-hard.*

Proof. The optimization version of the kC-VST problem is the C-VST problem. A certificate of C-VST is not verifiable in polynomial time, thus C-VST is not in NP. Hence, C-VST is NP-hard. □

3 RELATED WORK

3.1 Intermediate meeting points

As an alternative to users to be picked up from their original locations, they may agree to meet at intermediate spots. When users travel to intermediate locations, drivers do not need to make large detours, hence ride-sharing becomes more convenient. Intermediate locations can be any intersection within the city road network as the following related works consider. We refer to this alternative as *model A*. Our proposed model uses intermediate meeting points with the additional constraint of being hot-spots only.

Aissat et al. in [2] present two approaches to find optimal intermediate meeting points based on a detour factor. However, the authors consider only pairs of drivers and riders in a scenario where both converge at a starting intermediate location and diverge later at an ending intermediate location. Stiglic et al. in [9] show the benefits of including intermediate meeting points to ride-sharing. Their approach maximizes two objectives hierarchically. Their problem definition also considers starting and ending intermediate locations but with the possiblity of matching some riders with one driver.

Goel et al. in [5] presented the benefits of intermediate meeting points from the perspective of privacy. Their approach offers a Pareto front of optimal intermediate meeting points after two objectives were maximized, namely, coverage and *k-anonymity*. Based on estimated population density, coverage circles with different radii are centered at intermediate meeting points.

3.2 Activity-based ride-sharing

Activity-based travel planning regards places not only as spatial points but also as points where people go to perform an activity. Therefore, more than one destination may be appealing to riders as long as such destinations allow the users to carry out their desired activities. Getting users be flexible upon their destinations should increase the ride-sharing matching rates.

In [11], the authors design *ABRA*, an activity-based ride-sharing algorithm that expands the set of destinations by including a space-time filter. ABRA is an algorithm that computes the global optimum of all feasible matches. No heuristic besides space-time filter is applied, hence it is quite computationally expensive. There is no notion of meeting points and passengers do not necessarily go to the same destination. As its goal is to show the increment in terms of ride-sharing matching rate with respect to trip-based algorithms, it does not aim to offer a readily deployable solution.

On the other hand, Rahman et al. [8] introduced a fast algorithm, namely *VST-RS*, where users can meet at intermediate points and have a common destination. VST-RS chooses from a candidate set of POIs, the most convenient one for a subset of users. Although VST-RS shrinks the candidate meeting points space by the inclusion of novel constraints, it still regards every intersection in the road network as part of such search space (model A). As we will show later, the time complexity can be reduced by considering hot-spots only, without sacrificing cost savings. Indeed, in many realistic city scenarios, our solutions are less costly than VST-RS's. As our experiments use VST-RS as a baseline solution, we describe this algorithm in more detail in Section 4.

3.3 Steiner trees and ride-sharing

Olsen in [6] focus on computing an optimal plan for users who count on their own cars which they are willing to drive to a "park-and-ride" lot and ride-share afterwards to a common destination. Although this problem seems very similar to ours, it is a version of the *car pooling* problem. In car pooling, users ride-share to a common fixed destination on a daily basis, e.g. people going to their workplace, whereas in our problem there are multiple destinations and they are not known much earlier than the travel time. The author models this problem as an MST. His contribution is demonstrating that this problem is APX-complete when it is restricted to cars with capacity 4 and unweighted undirected graphs. The author does not propose any algorithm to solve it efficiently.

Zhang et al. [12] and Takise et al. [10] present ride-sharing as the MST problem. In [12] the authors propose an algorithm that solves to optimality instances of the problem up to 3 users arguing their problem is more difficult than the common ride-sharing problem where users meet only once in a particular travel. In [10], the algorithm resembles the seminal work of Dreyfus et. al [4] which computes the exact solution of the MST problem on graphs. Thus, their algorithm does not scale since its complexity is exponential on the number of users.

4 SOLUTIONS

This section presents our solution and a baseline solution to the C-VST problem. The baseline is a modified version of the algorithm VST-RS presented by Rahman et al. in [8]. We modified this algorithm to take into account the constraint on the feasible set of Steiner vertices and make it comparable to our solution.

We introduce the following definitions:

Definition 4.1. Minimum pair cost is a function $mp : V \times V \rightarrow \mathbb{R}_+$ defined for two vertices in a graph which returns the minimum cost between them.

Definition 4.2. Voronoi diagram in a graph $G = (V, A)$ for a subset of vertices $P = \{p_1, p_2, \ldots p_m\}$ is a set of cells, one for each vertex in P, such that each cell

$$C(p_i) = \{v | v \in V, mp(v, p_i) < mp(v, p_j), \forall p_j : p_j \neq p_i\}$$

The vertices in P are known as *medoids*. We define an auxiliary function $M(v)$ which returns the medoid corresponding to v. Within our ride-sharing context, the medoids correspond to the POIs.

4.1 VST-RS

VST-RS [8] is based on two important observations. Firstly, ride-sharers can maximize the travel cost savings if they meet sooner and their shared route to either a POI or another meeting point is longer. Thus, the algorithm begins by dividing the users into groups based on their initial locations. Users who are located closer to a common POI than to other POIs are grouped together. That is, the algorithm divides the graph, i.e., road network, into Voronoi cells where the medoids are the POIs. Intuitively, these users are going to meet sooner and then it is more likely for them to share a longer route. It is worth to mention that this initial division does not force the users to ride-share to their common medoid.

Second, the algorithm does not need to explore the whole road network to find good candidate meeting points. To that end, VST-RS imposes constraints on them. The assumption is that a user is willing to ride-share only if the meeting point is closer than going directly to her medoid, i.e., to her closest POI. These constraints are reflected in the following results:

Theorem 4.3. *In an optimal solution to the C-VST problem, two terminals x and y are connected to a Steiner vertex h only if $mp(x, h) < mp(x, M(x))$ and $mp(y, h) < mp(y, M(y))$ [8].*

Corollary 4.4. *In an optimal solution to the C-VST problem, a set of terminals which have already met at Steiner vertex h_1 and another set of terminals which have already met at Steiner vertex h_2 meet at Steiner vertex h only if $mp(h_1, h) < mp(h_1, M(h_1))$ and $mp(h_2, h) < mp(h_2, M(h_2))$ [8].*

If the number of users within a Voronoi cell exceeds a parameter S, VST-RS accommodates groups of at most S users within each cell. Each group of at most S users is further divided into subgroups of at most z users ($z < S$). z is a new parameter representing the capacity of a car. The division of users into groups of at most S users is the first stage of VST-RS.

In its second stage, VST-RS computes the optimal travel plan for each subgroup of z users. To compute the optimal travel plan, VST-RS computes the best plan for every combination of $2, 3, \ldots z$ users. This stage of VST-RS is based on Dreyfus et. al's algorithm [4] which computes the exact solution of the MST problem on graphs. However, VST-RS is able to deal with more than one possible POI, whereas Dreyfus et al.'s algorithm is not. In VST-RS, a terminal (set of terminals) chooses to connect to a Steiner vertex as long as Theorem 4.3 (Corollary 4.4) holds, otherwise it connects to its closest POI, which may result in choosing more than one POI per subgroup of z users. We modified this stage of VST-RS to include the constraint on the feasible set of Steiner vertices and make it comparable to our solution.

Let the power set $\mathcal{P}$ of users and POIs be the universe of a Set Cover problem. Let the cost of the MST of each subset of users and POI be the cost of each element in $\mathcal{P}$. Thus, the first stage of VST-RS computes an approximate solution to the Set Cover problem where the subsets in the solution must be disjoint. On the other hand, the second stage of VST-RS is an exact solution to the MST problem with the additional ability of choosing more than one POI.

4.2 Our Solution

Our approximate solution maximises a heuristic set function that we call *Gain ratio*. This function is associated with a candidate Steiner vertex and the vertices that may connect to it. The larger its value, the more attractive such Steiner vertex is for meeting. Since the size of the domain L of this function, i.e., the set of vertices that may connect to it, is 2^L, combinatorial explosion occurs when it is maximized naïvely. We show that our heuristic function is nondecreasing under a specific update rule. Such update rule maximizes this function in linear time with respect to the number of users for each candidate Steiner vertex. Thus, we construct a nearly optimal solution to the C-VST problem which is readily deployable.

Before describing our solution in detail, we define new functions, concepts and present more results. Recall that a road network is modelled with a graph $G = (V, A, c)$ where $U \subset V$, $P \subset V$ and $H \subset V$ correspond to the set of user, POI and hot-spot locations, respectively. Moreover, if we use MDSF's terminology, the users are represented as terminals, the POIs as roots and the hot-spots as Steiner vertices. Thus, we use these terms interchangeably.

Definition 4.5. Pseudo-terminal is a Steiner vertex which has been greedily chosen as part of the solution. After a Steiner vertex is chosen, it behaves like a terminal.

Definition 4.6. Subtree-SV is a tree of depth 1 where the root[2] is a pseudo-terminal or a candidate Steiner vertex and its leaves can be either terminals or other pseudo-terminals. A pseudo-terminal leaf is in turn the root of another Subtree-SV.

The function $Sv(h)$ returns the Subtree-SV with root h. The function $r(x)$ returns the root of the Subtree-SV x. The function $l(x)$ returns the set of leaves of the Subtree-SV x.

Definition 4.7. Independent cost is a function $Ic : (U \cup H) \rightarrow \mathbb{R}_+$ defined for a terminal and a pseudo-terminal as following:

$$Ic(t) = \begin{cases} mp(t, M(t)) & t \text{ is a terminal} \\ \sum_{w \in L} Ic(w) & t \text{ is a pseudo-terminal} \end{cases} \quad (1)$$

[2]Readers should not confuse the root of a Subtree-SV with a POI which may be the root of a Steiner tree in the solution.

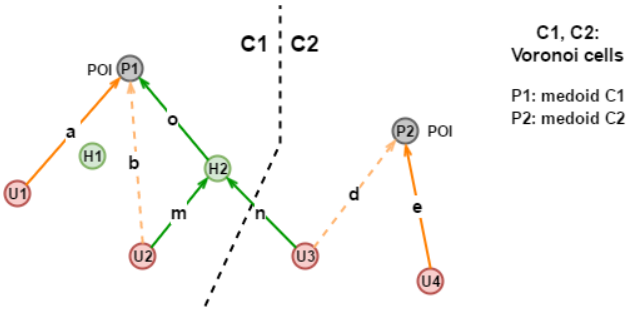

Figure 1: Let $U2$ and $U3$ be two terminals and $H2$ a hot-spot. $H2$ is the root of a Subtree-SV whose leaves are $U2$ and $U3$. $P1$ and $P2$ are the closest POIs to $H2, U2$ and $U3$, respectively (note the Voronoi cells $C1$ and $C2$). Thus, $Gr(\{U2, U3\}, H2) = \frac{b+d}{o+m+n}$, $Lr(U2, H2) = \frac{m}{b}$ and $Lr(U3, H2) = \frac{n}{d}$.

where $L = l(Sv(t))$

The independent cost can be understood as the minimum pair cost between a terminal and its closest POI, or the summation of the minimum pair costs between all its descendant terminals and their corresponding closest POIs in the case of a pseudo-terminal.

Definition 4.8. Cumulative loss is a function $Cl : (U \cup H) \rightarrow \mathbb{R}_+$ defined for a terminal and a pseudo-terminal as following:

$$Cl(t) = \begin{cases} 0 & t \text{ is a terminal} \\ \sum_{w \in L} Cl(w) + mp(w, t) & t \text{ is a pseudo-terminal} \end{cases} \quad (2)$$

where $L = l(Sv(t))$

Cumulative loss for a pseudo-terminal represents the cost all its descendants endure because of connecting to it. Thus, terminals do not have cumulative losses as they are leaves.

Definition 4.9. Gain ratio is a set function $Gr : 2^L \times H \rightarrow \mathbb{R}_+$ defined for the set of leaves of a Subtree-SV, whose root is a candidate Steiner vertex, as following:

$$Gr(L, h) = \frac{\sum_{w \in L} Ic(w)}{mp(h, M(h)) + \sum_{w \in L} Cl(w) + mp(w, h)} \quad (3)$$

where h is a candidate Steiner vertex.

Gain ratio of the set of leaves of a candidate Steiner vertex h can be seen as the proportion of cost savings that a set of terminals / pseudo-terminals would have if they connect to a POI through h, instead of them connecting independently to their closest POIs. If $Gr(L, h) > 1$ then, h is an appealing candidate. **Our approach must maximize** Gr (see Figure 1).

Definition 4.10. Loss ratio is a function $Lr : (U \cup H) \times H \rightarrow \mathbb{R}_+$ defined for a leaf of a Subtree-SV, whose root is a candidate Steiner vertex, as following:

$$Lr(t, h) = \begin{cases} 0 & t = h \\ \frac{Cl(t) + mp(t,h)}{Ic(t)} & t \neq h \end{cases} \quad (4)$$

where t can be either a terminal or a pseudo-terminal, and h is a candidate Steiner vertex. Since t can be a pseudo-terminal, equation (4) discerns whether t and h are the same or not.

Loss ratio $Lr(t, h)$ measures the proportion of cost increment of a terminal / pseudo-terminal t if it connects to a candidate Steiner vertex h instead of connecting directly to its closest POI. In the case of $t = h$, there is no cost increment as the pseudo-terminal is already "connected" to the candidate Steiner vertex. Generally, $Lr(t, h) > 1$. If $Lr(t, h) = 1$, then t and h have the same closest POI and h is part of the shortest path between t and this POI. $Lr(t, h) < 1$ can occur with pseudo-terminals only. **Our goal is to minimize** Lr (see Figure 1).

Definition 4.11. Well-formed Subtree-SV of a candidate Steiner vertex h is a Subtree-SV where its leaves

$$L = \{w | mp(w, h) < mp(w, M(w))\} \text{ and } |L| \geq 2$$

A well-formed Subtree-SV has at least 2 leaves since we are regarding Steiner vertices with degree ≥ 3 in our solution.

Definition 4.12. Committed terminal / pseudo-terminal
Let t be a committed terminal or pseudo-terminal then, $\forall h \in H : mp(t, M(t)) < mp(t, h)$.

Since our goal is to maximize Gr, we present an update rule, in Theorem 4.13, to attain a stationary point.

Theorem 4.13. *The set function $Gr(L, h)$ for a particular candidate Steiner vertex h is nondecreasing under the update rule*

$$L_{k+1} = L_k \backslash \{t\} \text{ only if } \frac{1}{Gr(L_k, h)} < Lr(t, h) \quad (5)$$

where $t = \arg\max_{w \in L_k} Lr(w, h)$

Proof. At iteration k:

$$Gr(L_k, h) = \frac{Ic(t) + \sum_{w \in W} Ic(w)}{mp(h, M(h)) + Cl(t) + mp(t, h) + \sum_{w \in W} Cl(w) + mp(w, h)} \quad (6)$$

where $W = L_k \backslash \{t\}$

And at iteration $k + 1$:

$$Gr(L_{k+1}, h) = \frac{\sum_{w \in W} Ic(w)}{mp(h, M(h)) + \sum_{w \in W} Cl(w) + mp(w, h)} \quad (7)$$

where $W = L_k \backslash \{t\}$

Let $A = \sum_{w \in W} Ic(w)$, $B = \sum_{w \in W} Cl(w) + mp(w, h)$, $a = Ic(t)$, $b = Cl(t) + mp(t, h)$, $s = mp(h, M(h))$. Thus, equations 6 and 7 become:

$$Gr(L_k, h) = \frac{a + A}{s + b + B} \qquad Gr(L_{k+1}, h) = \frac{A}{s + B} \quad (8)$$

Then, from equations in 8:

$$Gr(L_{k+1}, h) = Gr(L_k, h) + Gr(L_k, h)\frac{b}{s + B} - \frac{a}{s + B} \quad (9)$$

From equation 9 we observe that $Gr(L_{k+1}, h) > Gr(L_k, h)$ (non-decreasing) only if:

$$Gr(L_k, h)\frac{b}{s+B} > \frac{a}{s+B} \tag{10}$$

$$\frac{1}{Gr(L_k, h)} < \frac{b}{a} \tag{11}$$

$$\frac{1}{Gr(L_k, h)} < Lr(t, h) \tag{12}$$

When $\frac{1}{Gr(L_k,h)} > Lr(t, h)$, L_k is a stationary point because $\forall t' \in L_k \backslash \{t\} : Lr(t, h) > Lr(t', h)$ then, $\forall t' \in L_k \backslash \{t\} : \frac{1}{Gr(L_k,h)} > Lr(t', h)$. □

4.2.1 Algorithm. Our main algorithm is Algorithm 1. It computes and builds the approximate solution for the C-VST problem. The solution is built iteratively in a bottom-up manner. The first-level Subtree-SVs (see Definition 4.6), has only terminals as leaves. In the next iteration, new Subtree-SVs may be built on top of the Subtree-SVs of the previous iteration. That is, these new Subtree-SVs may have pseudo-terminals (see Definition 4.5) as leaves.

At each iteration, Subtree-SVs are chosen greedily based on how much gain their roots (candidate Steiner vertices) offer. The set function *Gr* (see Definition 4.9) can be seen as the proportion of cost savings that a set of terminals / pseudo-terminals would have if they connect to a POI through a particular candidate Steiner vertex instead of them connecting independently to their closest POIs. This is the intuition behind the optimization of *Gr* for a set of leaves (terminals / pseudo-terminals) of a particular candidate performed in Algorithm 2 and specifically in Algorithm 3.

Although *Gr* is a function over a set of leaves of a Subtree-SV, this function can also be seen as the ratio between the total of independent costs of a subset of terminals and the cost of the tree that spans the same subset of terminals whose root is the candidate Steiner vertex. Thus, Algorithm 2 repeatedly finds trees with good gain ratio, each spanning only a subset of terminals. Then, our solution obtains partial approximate solutions whose union yields the final solution.

In the following, we describe each of the algorithms.

Algorithm 1 Compute and build approximate C-VST.

```
1:  procedure C-VST(T, H)    ▷ T : set of terminals, H : set of hot-spots
2:      N = {n | n ∈ T, n is a non-connected terminal}
3:      S = {s | s = Sv(h), s : well-formed Subtree-SV, h ∈ H}
4:      while N ≠ ∅ and S ≠ ∅ do
5:          R = ChooseSubtreeSVs(S)
6:          T = T\{l(s') | s' ∈ R}
7:          T = T ∪ {r(s') | s' ∈ R}
8:          N = {n | n ∈ T, n is a non-connected terminal}
9:          S = {s | s = Sv(h), s : well-formed Subtree-SV, h ∈ H}
10: end procedure
```

Computing and building an approximate solution to C-VST. Committed terminals (see Definition 4.12) may exist within the set of terminals T given as parameter. Committed terminals have no option but to *connect* directly with their own closest POIs as these POIs are closer to these terminals than any candidate Steiner vertex. There will be no gain for a user u_i (terminal) to meet another user u_j (terminal) at a meeting point h_{ij} (candidate Steiner vertex) if they are committed terminals, as it is shown in Theorem 4.3. Therefore, such committed terminals are not included in set N, in line 2.

Line 3 initializes the set of well-formed Subtree-SVs (see Definition 4.11). Each Subtree-SV has as root a candidate Steiner vertex from the set of hot-spots H given as parameter. Here is where we constrain the feasible set of Steiner vertices to the set of hot-spots. Subtrees in S will likely share their leaves. Later on, in Algorithm 2, those shared links between Subtree-SVs will be dropped.

The loop from lines 4 through 9 is executed only if there are disconnected terminals and well-formed Subtree-SVs. These two sets are updated within the loop in lines 8 and 9, respectively.

Line 5 greedily chooses the Subtree-SVs which will be part of the solution (see Algorithm 2). The roots (Steiner vertices) of the set of chosen Subtree-SVs become pseudo-terminals. Thus, the set of terminals is adjusted. First, the terminals, which are leaves of the chosen Subtree-SVs, are removed (line 6). Second, the set of pseudo-terminals is added to this adjusted set (line 7). At this point, there may be committed pseudo-terminals, thus set N is updated accordingly in line 8. Committed pseudo-terminals will connect directly with their own closest POIs. There will be no gain for a set of users (pseudo-terminal) to meet other users (pseudo-terminal) at a meeting point (candidate Steiner vertex) if they are committed pseudo-terminals, as it is shown in Corollary 4.4.

Finally, a new set of well-formed Subtree-SVs is computed in line 9. Note that pseudo-terminals can now be included as leaves of these new Subtree-SVs. In line 4, if there are not disconnected terminals nor well-formed Subtree-SVs, the algorithm ends. The solution is the union of the Subtree-SVs chosen greedily in line 5.

Algorithm 2 Greedily choose Subtree-SVs.

```
1:  procedure ChooseSubtreeSVs(S)                    ▷ S : Subtree-SVs
2:      R = ∅
3:      while S ≠ ∅ do
4:          S' = {s' | s' ∈ S}
5:          for each s' ∈ S' do
6:              s' = MaximizeGainRatio(s')
7:          s_max = arg max_{s'∈S'} Gr(W, r(s'))          ▷ W : leaves of s'
8:          R = R ∪ {s_max}
9:          Drop shared leaves from Subtree-SVs ∈ S\{s_max}
10:         S = {s | s ∈ S, r(s) ∉ R, s : still well-formed}
11:     return R
12: end procedure
```

Choosing Subtree-SVs greedily. A set of well-formed Subtree-SVs is computed by Algorithm 1 and then it is passed as parameter S to Algorithm 2. In turn, Algorithm 2 chooses, from S, the Subtree-SVs that will be part of the solution. Recall that the solution is built in a bottom-up manner and as result of this algorithm, a new level of Subtree-SVs is added to the forest.

The loop from lines 3 through 10 is executed only if set S has elements. S is re-computed within this loop, in line 10.

The loop starts by creating a copy S' of set S. S' is needed because we do not want S to be affected by the sub-loop of lines 5 and 6. In this loop, each Subtree-SV $s' \in S'$ is pruned in such a way that

$Gr(l(s'), r(s'))$ is maximized (see Algorithm 3). Theorem 4.13 shows that Gr attains a stationary point by pruning the set of leaves of the corresponding Subtree-SV as long as a condition holds.

After the loop of lines 5 and 6, S' ends up having Subtree-SVs that attain an optimal Gr value. s_{max} is automatically chosen (line 8) as it is the Subtree-SV which offers the largest gain among the elements in S'.

Since there may be Subtree-SVs in S that share leaves with s_{max}, the connections between these shared leaves and the roots of the other Subtree-SVs in S are dropped (line 9). In line 10, the set of well-formed Subtree-SVs S is re-computed because after dropping the shared leaves in the previous line may cause some Subtree-SVs being non well-formed.

The algorithm ends when no well-formed Subtree-SVs are left in S. The set of chosen Subtree-SVs R is returned. As result, a new level of Subtree-SVs R with pseudo-terminals as roots is built.

Algorithm 3 Maximize Gain Ratio.

```
1:  procedure MaximizeGainRatio(s')                ▷ s' : Subtree-SV
2:      h = r(s')
3:      L_k = l(s')
4:      while |L_k| > 2 do
5:          t_max = arg max_{t∈L_k} Lr(t, h)
6:          if 1/Gr(L_k, h) < Lr(t_max, h) then
7:              L_k = L_k \ {t_max}
8:          else break
9:      return s'                                  ▷ s' has been pruned
10: end procedure
```

Maximizing Gr by pruning a Subtree-SV. Algorithm 3 prunes a Subtree-SV s' given as a parameter through the application of the update rule of Theorem 4.13. The loop from lines 4 through 8 represents such update rule. The additional condition in line 4 prevents s' from becoming a **non** well-formed Subtree-SV, i.e., a Subtree-SV with less than 2 leaves, as we are interested in maximize Gr only. Non well-formed Subtree-SVs are excluded in Algorithm 2, line 10.

Because of Theorem 4.13, we can safely stop the first time Gr decreases, i.e., when the condition specified in the theorem does not hold. In this way, we do not need to explore the whole domain (2^L) of Gr when searching for an optimal value. Indeed, the worst case time complexity when maximizing Gr for the leaves of a particular Subtree-SV is $|U|$.

4.2.2 Overall Time Complexity Analysis. The worst scenarios are: $|L| = |U|$ and $|S| = |H|$ where L is the set of leaves of a Subtree-SV, U is the set of user locations, S is the set of Subtree-SVs and H is the set of hot-spots.

If we start our analysis on the innermost algorithm (Algorithm 3), its complexity is $O(|U|)$ since it traverses all leaves of s' at most.

In Algorithm 2: Algorithm 3 is executed for each Subtree-SV, that is, line 6 is executed $O(|H||U|)$ times. Line 9 is executed for each Subtree-SV except s_{max}, and for each shared leaf, that is, $O(|H||U|)$ times. Line 10 checks out, for each terminal / pseudo-terminal, whether it fulfills the well-formed Subtree-SV condition to be a leaf. This is verified for each hot-spot (root of a Subtree-SV). That is, line 10 is executed $O(|H||U|)$ times. Finally, the outer loop is executed $O(|H|)$ times. Therefore, Algorithm 2's complexity is $O(|H|^2|U|)$.

In Algorithm 1: lines 3 and 9 have the same complexity as line 10 in Algorithm 2, that is $O(|H||U|)$. Lines 2 and 8 have the same complexity and each is executed $O(|U|)$ times since it traverses, at most, all user locations. The outer loop is executed $\min(|H|, |U|)$.

Therefore, the overall complexity of our solution is

$$O(\min(|H|, |U|)\ |H|^2|U|)$$

4.2.3 Extension.

Limited number of terminals per Steiner tree (Cars with capacity z). Rahman et al. [8] further divided groups of S users into subgroups of at most z ($z < S$) users as their goal is to take into account the capacity of a car. Recall that each Steiner tree within the solution of the C-VST problem represents a ride-sharing arrangement of a subset of users (terminals) to a POI. Thus, it is plausible to include an upper bound z to the number of terminals per Steiner tree that represents the maximum number of people a car can accommodate.

In Algorithm 2, a subset of Subtree-SVs are chosen. The leaves of such subset of Subtree-SVs are pruned as a result of maximizing Gr. To reach the upper bound z of the number of terminals per Steiner tree, a constructive method is needed instead of a destructive one (pruning). Likewise Theorem 4.13, Theorem 4.14 shows that Gr is nondecreasing under an update rule. However, rather than pruning the leaves of the Subtree-SV in order to maximize Gr, Theorem 4.14 presents a constructive update rule. Now, the set of leaves of a Subtree-SV is constructed by including leaves iteratively as long as a condition holds.

Theorem 4.14. *The set function $Gr(L, h)$ for a particular candidate Steiner vertex h is nondecreasing under the update rule*

$$L_{k+1} = L_k \cup \{t\} \text{ only if } \frac{1}{Gr(L_k, h)} > Lr(t, h) \text{ or } L_k = \emptyset \qquad (13)$$

where $t = \arg\min_{w \in Y_k} Lr(w, h),\ Y_k = l(Sv(h)) \backslash L_k$

Proof. If $L_0 = \emptyset$ then, $0 = Gr(L_0, h) < Gr(L_1, h)$.
At iteration $k > 0$:

$$Gr(L_k, h) = \frac{\sum_{w \in L_k} Ic(w)}{mp(h, M(h)) + \sum_{w \in L_k} Cl(w) + mp(w, h)} \qquad (14)$$

And at iteration $k + 1$:

$$Gr(L_{k+1}, h) = \frac{Ic(t) + \sum_{w \in L_k} Ic(w)}{mp(h, M(h)) + Cl(t) + mp(t, h) + \sum_{w \in L_k} Cl(w) + mp(w, h)} \qquad (15)$$

Let $A = \sum_{w \in L_k} Ic(w)$, $B = \sum_{w \in L_k} Cl(w) + mp(w, h)$, $a = Ic(t)$, $b = Cl(t) + mp(t, h)$ and $s = mp(h, M(h))$. Thus, equations 14 and 15 become:

$$Gr(L_k, h) = \frac{A}{s + B} \qquad Gr(L_{k+1}, h) = \frac{a + A}{s + b + B} \qquad (16)$$

Then, from equations in 16:

$$Gr(L_{k+1}, h) = \left(\frac{a}{s + B} + Gr(L_k, h)\right)\frac{s + b}{s + b + B} \qquad (17)$$

From equation 17 we observe that $Gr(L_{k+1}, h) > Gr(L_k, h)$ (non-decreasing) only if:

$$\left(\frac{a}{s+B} + Gr(L_k, h)\right)\frac{s+b}{s+b+B} > Gr(L_k, h) \tag{18}$$

$$\frac{1}{Gr(L_k, h)} > \frac{b}{a} \tag{19}$$

$$\frac{1}{Gr(L_k, h)} > Lr(t, h) \tag{20}$$

When $\frac{1}{Gr(L_k,h)} < Lr(t, h)$, L_k is a stationary point because $\forall t' \in Y_k : Lr(t, h) < Lr(t', h)$ then, $\forall t' \in Y_k : \frac{1}{Gr(L_k,h)} < Lr(t', h)$. □

Definition 4.15. Spanned terminals is a function $span : (U \cup H) \rightarrow \mathbb{Z}_+$ defined for a terminal and a pseudo-terminal as following:

$$span(t) = \begin{cases} 1 & t \text{ is a terminal} \\ \sum_{w \in L} span(w) & t \text{ is a pseudo-terminal} \end{cases} \tag{21}$$

where $L = l(Sv(t))$

It returns the number of terminals that are spanned by the tree that has t as root.

Algorithm 4 builds a new Subtree-SV based on a given one (s') whose corresponding tree spans up to z terminals (see Definition 4.15). The set of leaves L_k of the new Subtree-SV is initialized with the leaf of minimum loss ratio among the set of leaves of s' sent as a parameter (line 4). Set Y_k, which has the rest of leaves of s', is the source from which the leaves are taken to construct such new Subtree-SV. Leaves are added one by one until either all leaves of Y_k has been analyzed, the upper bound z has been reached or the Theorem 4.14's condition does not hold (lines 7 and 8).

Algorithm 4 Accommodate z-leaves.

```
1:  procedure ACCOMMODATE Z-LEAVES(s', z)          ▷ s' : Subtree-SV
2:      h' = r(s')
3:      t_min = arg min_{t∈l(s')} Lr(t, h')
4:      L_k = {t_min}
5:      Y_k = l(s')\L_k
6:      Connect t_min with root h
7:      while Y_k ≠ ∅ do
8:          if 1/Gr(L_k,h') ≤ Lr(t_min, h') or span(h) = z then break
9:          t_min = arg min_{t∈Y_k} Lr(t, h')
10:         if span(h) + span(t_min) ≤ z then
11:             L_k = L_k ∪ {t_min}
12:             Y_k = l(s')\L_k
13:             Connect t_min with root h
14:     return Sv(h)
15: end procedure
```

Line 10 checks out whether adding the terminal / pseudo-terminal with minimum loss ratio from Y_k does not surpass the upper bound. If this is the case, such terminal / pseudo-terminal is added to the set of leaves L_k (line 11) and also connected to the root h of the new Subtree-SV (line 13). Moreover, Y_k is updated accordingly (line 12).

In the scenario of cars with limited capacity, the call to Algorithm 3 from Algorithm 2 in line 6 will be replaced by a call to Algorithm 4. Since time complexity of Algorithm 4 is $O(|U|)$ (it traverses all leaves in Y_k at most), the overall complexity of our solution stays the same.

Parameter	Range	Default
Number of users	2 - 2048	256
Number of POIs	10 - 640	80
Percentage of hot-spots	3% - 20%	3%
Car capacity	4 - 10	4
Number of vertices	1250 - 80000	10000

Table 1: Parameter set-up in synthetic graphs.

5 EXPERIMENTS

The goal of the experiments is two-fold. First, we present empirical evidence of the superiority of our solution in terms of efficiency and effectiveness with respect to the modified version of the algorithm presented by Rahman et al. [8], i.e., a version which takes into account the constraint on the feasible set of Steiner vertices. Second, we compare our proposed model and model A[3] and show that (a) significant difference in occurrence of popular meeting points[4] exists, which means round-trip ride-sharing arrangements are more likely in our scheme; and (b) alternative placement of hot-spots, either uniformly or regarding the population distribution, decreases the car occupancy rate gap significantly between both models.

We refer to our solution as *Gr-based* as a short form of "Gain-ratio-based". Baseline solution VST-RS is referred as *mVST-RS* as a short form of "modified VST-RS."

5.1 Simulation setup

Experiments are run over synthetic and Melbourne-based graphs.

In the case of synthetic graphs, grid-based graphs with uniformly distributed arc costs are built. Grid structure is chosen because our goal is to simulate blocks in a city, where every corner of a block is a vertex in the graph. POIs, hot-spots and users vertices are chosen randomly. Our parameters are *number of users, number of POIs, percentage of graph vertices that are hot-spots, car capacity and number of vertices.* The number of users represent a set of users looking for ride-share at a given time interval. The range of values and defaults for these parameters are shown in Table 1. Fifty trials were run for each value of a parameter within its range, while keeping the others with their default.

In the case of Melbourne-based graphs, the City of Maribyrnong graph is built. Road intersections, POIs and hot-spots are extracted from *OpenStreetMap* [7]. In *OpenStreetMap*, information is classified either as nodes or ways and both are tagged with key-value pairs. Road intersections are nodes that belong to at least two ways whose tag-key is `highway`. POIs nodes are retrieved based on activities, e.g. if the activity at destination is *shopping centre*, the nodes within ways with tag pair `shop:mall` are retrieved. Hot-spots are the nodes within ways with tag pair `service:parking_aisle`.

Distribution of users is based on the estimated population of the suburbs within the City of Maribyrnong and the number of trips according to the Victorian Integrated Survey of Travel and Activity

[3]Users meet at intermediate points which are not hot-spots necessarily.
[4]We refer to popular meeting points to those where more than z users coincide in a time window of an hour.

(VISTA) 2009-2010 [1]. The number of trips were retrieved per departure hour as each simulation considers users who depart from their locations at the same hour (they are more likely to ride-share).

VISTA allows filtering the trips per activity at destination. The considered activities are *Shopping Centre, Supermarket, Food Store, Fast Food and Swimming Pool* as we believe these types of destinations make users more flexible when deciding where to go, i.e., this does not happen with a restaurant which is more likely to be specific to users tastes.

5.2 Dependent variables

The following variables are evaluated:

Processing time is the time elapsed without considering shortest paths as they can be pre-computed.

Cost is the summation of the distances travelled by all cars. That is, if the travel plan is represented by the MDSF F and each subset of users who ride-share is a tree T then, the cost $c(F) = \sum_{T \in F} c(T)$ where $c(A) = \sum_{a \in A}[\text{cost of arc } a]$.

Avg. occupancy ratio is the weighted average of the number of passengers per car. Let n_a be the number of passengers who ride-share in a car through arc a. Thus, the average occupancy ratio $o = \frac{1}{c(F)} \sum_{a \in F}[\text{cost of arc } a]n_a$.

5.3 Comparison with baseline in terms of efficiency and effectiveness

5.3.1 Processing time. Theoretically, our solution is less time complex than the baseline (see Section 4.2.2). However, we show that it is also significantly faster in terms of *processing time*.

Figure 2(a) shows how dependent the baseline's efficiency is on the number of users ($|U|$). Although its time complexity is exponential on the size S of the group of users, which we keep constant ($S = 8$), the number of groups increases when $|U|$ increases. Since an exact solution is computed within each group, exponential growth in processing time is observed for this baseline. On the contrary, our solution's complexity is polynomial (max. degree = 2) on U.

Figures 2(b, d) show that our solution's efficiency is not dependent on the number of POIs ($|P|$) nor the car capacity. Figure 2(b) shows that the baseline's efficiency is comparable to ours only when $|P|$ is large enough because, in this case, it searches small areas to find the possible meeting points, thus its processing time decreases.

When the graph size ($|V|$) increases, the number of hot-spots ($|H|$) also increases as $|H|$ is a proportion of $|V|$ in our setting. Figure 2(c) shows that our solution grows linearly in processing time when $|H|$ increases. However, counting on large proportions (> 10%) of hot-spots in a city is unreal. Figure 2(e) shows how the baseline grows exponentially when $|V|$ increases because it has to search much larger areas to find the possible meeting points, whereas our solution is scalable as it searches in $H \subset V$.

With respect to Melbourne-based graphs, Figure 3 shows that our solution is one to two orders of magnitude faster than the baseline for any activity throughout the day.

5.3.2 Cost. Although our solution is not significantly less *costly* than VST-RS, it is on average in every scenario on both kinds of graphs (see Figures 2(f) and 4). Our solution outperforms VST-RS due to the initial division of users carried out in the first stage in

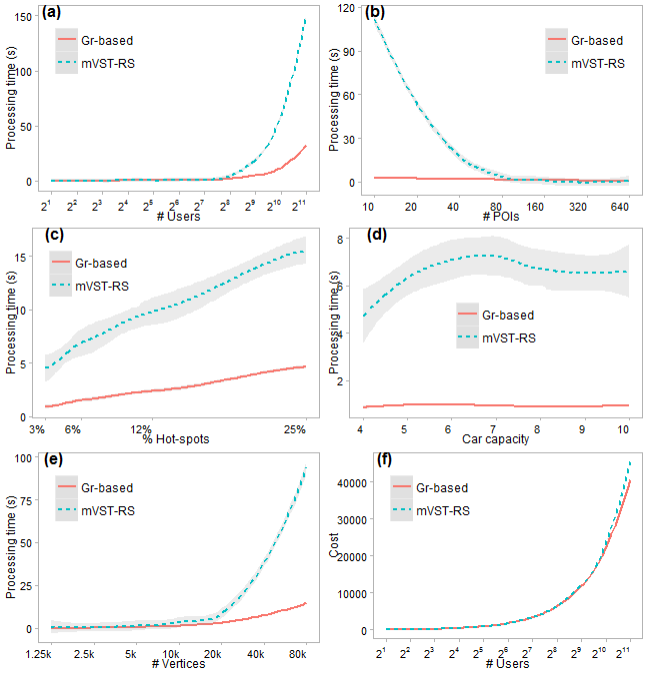

Figure 2: Comparison in terms of processing time and cost on synthetic graphs. (a) through (e) show processing time vs. our 5 parameters. (f) shows cost vs. number of users.

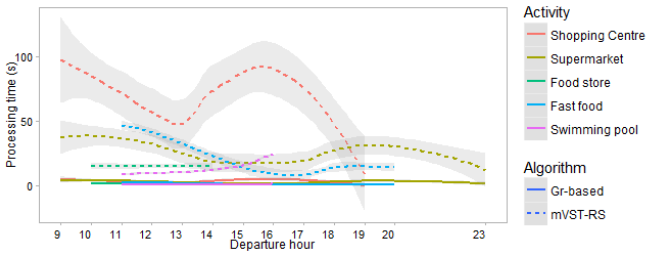

Figure 3: Comparison of Processing time on Melbourne-based graphs for different activities throughout the day.

VST-RS. Even though VST-RS may find that two or more users who belong to the same Voronoi cell could ride-share to a POI of a neighboring cell, VST-RS prevents two users of different cells from ride-sharing. In our approach, users are not grouped based on their mutual closeness but their closeness to candidate meeting points.

Comparison of both methods between cost and the other 4 parameters on synthetic graphs was also performed. These figures are not included because of lack of space and same conclusion obtained.

5.4 Comparison with model A in terms of round-trip likelihood and occupancy rate

In model A, users can meet anywhere, that is, *0% of people would dismiss a non-suitable intersection*, i.e., an intersection where parking is difficult. Whereas, in our model, *100% of people would dismiss such intersections*. We hypothesize that meeting at hot-spots offers (a)

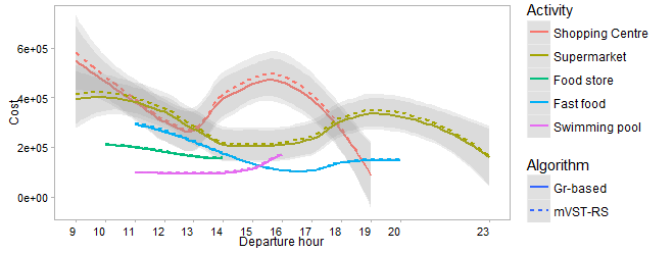

Figure 4: Comparison of Cost on Melbourne-based graphs for different activities throughout the day.

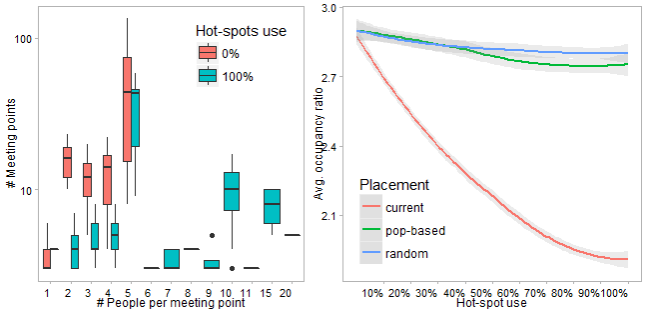

Figure 5: (a) Frequency of meeting points where a number of people (X axis) started their travel from. (b) Car occupancy ratio under three scenarios of placement of hot-spots at different levels of adoption of our model.

higher likelihood of finding a round-trip ride-sharing arrangement and (b) similar car occupancy rate, when comparing to model A.

5.4.1 Higher likelihood of a round-trip. Figure 5(a) shows the frequency of meeting points where X number of people start their travel from. Let $z = 5$ be the car capacity in these experiments. Let $d = \frac{X}{z}$ be the minimum number of drivers asked to come back to a particular starting meeting point. In both models, a user who started her travel from a meeting point where other 4 or fewer people ($X \leq 5$) met may find $d = 1$ drivers for coming back to such meeting point. On the other hand, **only in our model, a user may find $d > 1$ drivers for coming back** because $X \geq 5$ occurs with hot-spots only. The fact that X is always larger in our model than in model A confirms our hypothesis that more people met at hot-spots than at normal spots. Thus, having hot-spots as meeting points create more demand to come back to such starting points, which in turn creates more offer (higher round-trip likelihood).

5.4.2 Comparable occupancy ratio. Figure 5(b) shows how a gradual uptake of our model affects the car occupancy ratio of model A under three scenarios: (a) current parking infrastructure is used, (b) random parking placement, and (c) population-based parking placement. Population-based placement considers the number and location of hot-spots as a function of the distribution of the population. In the first scenario, as more people prefer using hot-spots only, occupancy ratio is more affected because hot-spots are not uniformly nor population-based distributed across Melbourne. Consequently, our algorithm would suggest going directly to the closest POI rather than meeting at a hot-spot that is not located within a convenient distance. Placement of hot-spots according to the population distribution, or even randomly, would keep car occupancy ratio at similar levels than model A's. **Model A's car occupancy ratio is an infeasible upper bound in all scenarios as it allows meeting at non-suitable points.**

6 DISCUSSION AND CONCLUSIONS

Our work proposes the inclusion of hot-spots into activity-based ride-sharing model as only-possible meeting points. We show that this model makes ride-sharing more convenient because round-trips are more likely. We also show that our model keeps a comparable car occupancy rate to model A, which in turn has been proved to increase the occupancy rate compared with model B [3, 6, 10].

We define a new problem on graphs that we call Constrained Variable Steiner Tree (C-VST). Although the parametric complexity of VST is reduced by constraining the set of feasible meeting points (Steiner vertices) to hot-spots, we prove that C-VST is still NP-hard. Our approximate solution, which exploits the monotonic nondecreasing property of our proposed *Gain ratio* function under a specific update rule, becomes a readily deployable solution for city-wide instances of the C-VST problem.

Our approach has a lower time complexity and is empirically faster (up to two orders of magnitude) than the state-of-the-art approach of Rahman et al. [8] on Melbourne-based graphs. The average processing time in a laptop is less than 10 seconds, on synthetic graphs with 1024 concurrent users who want to perform the same activity. Our solution is also less costly, on average, than this baseline as it was shown on both kinds of graphs.

REFERENCES

[1] Victorian Department of Transport. Victorian Integrated Survey of Travel and Activity (2009-2010), 2011.
[2] K. Aissat and A. Oulamara. Dynamic ridesharing with intermediate locations. In *Computational Intelligence in Vehicles and Transportation Systems (CIVTS), 2014 IEEE Symposium on*, pages 36–42. IEEE, 2014.
[3] V. Chaube, A. L. Kavanaugh, and M. A. Perez-Quinones. Leveraging social networks to embed trust in rideshare programs. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–8. IEEE, 2010.
[4] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1972.
[5] P. Goel, L. Kulik, and K. Ramamohanarao. Optimal pick up point selection for effective ride sharing. *IEEE Transactions on Big Data*, 2016.
[6] M. Olsen. On the complexity of computing optimal private park-and-ride plans. In *International Conference on Computational Logistics*, pages 73–82. Springer, 2013.
[7] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org, 2017.
[8] M. Rahman, O. Correa, E. Tanin, L. Kulik, and K. Ramamohanarao. Ride-sharing is about agreeing on a destination. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (in press)*. ACM, 2017.
[9] M. Stiglic, N. Agatz, M. Savelsbergh, and M. Gradisar. The benefits of meeting points in ride-sharing systems. *Transportation Research Part B: Methodological*, 82:36–53, 2015.
[10] K. Takise, Y. Asano, and M. Yoshikawa. Multi-user routing to single destination with confluence. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 72. ACM, 2016.
[11] Y. Wang, R. 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*, pages 1:1–1:10. ACM, 2016.
[12] X. Zhang, Y. Asano, and M. Yoshikawa. Mutually beneficial confluent routing. *IEEE Transactions on Knowledge and Data Engineering*, 28(10):2681–2696, 2016.