Author/s:
Correa Guerrero, Oscar Fabian

Title:
Effective Transportation Models for Sharing Economy Through Graph Theory

Date:
2020

Persistent Link:
https://hdl.handle.net/11343/249362

Effective Transportation Models for Sharing Economy Through Graph Theory

Oscar Correa

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

March 2020

Abstract

Ride-sharing and crowdsourced delivery are two sharing-economy transportation models that have been a trend in academic research and industry. Although large companies have popularized these terms through their platforms and the online services they offer, their versions do not follow the sharing-economy vision. Ride- and delivery-sourcing are the proper names for such services as these companies are outsourcing private drivers to serve transportation requests. As a consequence, instead of reducing the number of vehicles on the roads, the opposite effect has been observed. Congestion and pollution, in some US cities and Europe, have ride- and delivery-sourcing as their main contributors. Ride- and delivery-sourcing are convenient for drivers, as they are making a living from them, and for customers, as they pay less compared with a taxi (resp. a courier company). Therefore, this thesis proposes novel ride-sharing and crowdsourced delivery models that can compete with the convenience, trust, and incentives provided by ride- and delivery-sourcing. Our first model exploits the advantages that suitable meeting points have in ride-sharing. In our second model, congestion caused mainly by ride-sharing is considered. Our last model challenges the way retailers optimize their delivery process by regarding routing and pick-up store selection as part of the same optimization process. Optimal travel plans for instances within our novel models are equivalent to solving graph-theoretic problems. We contribute with novel graph problems related to well-known problems such as the Minimum Steiner Tree, Multicast Congestion and Hamiltonian Path. Our novel problems are solved efficiently through heuristic-based algorithms for which we show their effectiveness when compared to optimal solutions and state-of-the-art approaches.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Oscar Correa, March 2020

Acknowledgements

I would like to thank Anabelle for everything.
I see her strength and worth every day, every moment.

I thank Egemen for that email that ended in this, which changed my life. For the precious advice of trying to find the best of me and in my work, and tell it loudly.
I thank Rao for being my model, my mentor.
I thank Lars for always challenging me, correcting me, talking to me friendly, and smiling honestly.
I thank Data61, Australia's leading digital research network, for believing and helping me with extra funding.

I thank my parents for their unconditional love.

Preface

This thesis contains six chapters. The first two chapters provide an introduction to the problem, background and related work. The last chapter summarizes and concludes the thesis and proposes future research directions. The remaining chapters cover the core research topics. I am the primary author of these publications or under review papers:

- **Part of Chapter 2 is based on the following:**
  Correa, O., Tanin, E., Kulik, L. and Ramamohanarao, K., 2018, November. Activity-Based Ride-Sharing in Action (Demo Paper). In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (pp. 608-611). ACM.

- **Chapter 3 is based on the following:**
  Correa, O., Ramamohanarao, K., Tanin, E. and Kulik, L., 2017, November. 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* (pp. 136-145). ACM.

- **Chapter 4 is based on the following:**
  Correa, O., Khan, A.M.R., Tanin, E., Kulik, L. and Ramamohanarao, K., 2019. Congestion-Aware Ride-Sharing. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 5(1), pp.1-33.

- **Chapter 5 is based on the following:**
  Correa, O., Tanin, E., Ramamohanarao, K., Kulik, L. and Zaslavsky, A., 2020. Cost-Effective Crowdsourced Delivery Through Concurrent Routing and Pick-Up Store

Selection. *Proceedings of the VLDB Endowment*. Under Review.

In addition, I contributed to the following additional publication as a co-author while completing my PhD:

- Khan, A.K.M., Correa, O., Tanin, E., Kulik, L. and Ramamohanarao, K., 2017, November. Ride-Sharing is About Agreeing on a Destination. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (p. 6). ACM.

This thesis was prepared in LaTeX. The algorithms included were written in Python and run on Windows 7 operating systems.

To my beloved family, Anabelle, Manuela, and Daniel.

Contents

List of Figures

List of Tables

Acronyms

AV Autonomous Vehicle

C-VST Constrained Variable Steiner Tree

GHPPCP Group Hamiltonian Path with Precedence Constraints Problem

HPPCP Hamiltonian Path with Precedence Constraints Problem

ST Minimum Steiner Tree

MC Multicast Congestion

PDP Pickup and Delivery Problem

POI Point of Interest

TSP Traveling Salesman Problem

VST Variable Steiner Tree

VST-CA VST-based Congestion-Aware

VST-RS VST-based Ride-Sharing

Chapter 1
Introduction

SHARING economy, also known as peer-to-peer, gig, collaborative or access economy, is an economic model that has become ubiquitous. Although the idea of sharing idle assets is not new, it has recently become a trend due to the Internet and, in particular, to online social networks. Online platforms nowadays enable individuals to offer spare bedrooms, vehicle seats, peer-to-peer lending or even knowledge to other individuals. Companies such as AirBnB and Uber were arguably the first which enabled individuals to connect and trade services through their Internet platforms. AirBnB connects homeowners who have spare bedrooms with individuals who need a place to stay. Uber dispatches drivers in their personal vehicles to serve transport requests. Since then, many other sharing-economy companies have appeared in both the same and different business domains, given the success of these pioneer companies. Sharing economy is ubiquitous nowadays in a variety of domains, and its massive adoption has been facilitated by online social platforms.

The success of this model is based on different factors. First, products and services are generally cheaper compared with the traditional model. This is because the individuals who provide the products and services face less costs than companies. Moreover, individuals can rent products on demand rather than buy them and deal with the ownership costs. Secondly, providers have extra income and the flexibility of being their own bosses. Thirdly, the participants in a transaction can collaborate and create social bonds, e.g., when individuals share a vehicle ride. Lastly, there are environmental benefits as a consequence of the more efficient use of idle assets. For instance, vehicle seats, which otherwise would be empty, are occupied and thus, fewer vehicles transit and less pollution

is generated.

However, sharing economy has also disadvantages. Among them, we can mention safety concerns. For example, criminal records and other status checks of the drivers are not required in every country where Uber operates. Privacy is also compromised when an Uber passenger informs her location to the driver, or when a stranger enters the host's house in AirBnB. Individuals who offer the service cannot count on a stable income. Furthermore, these individuals do not receive labor benefits as they are considered "micro-entrepreneurs" rather than employees. Although sharing-economy companies are associated with disruptive innovation and thus, creation of new markets, they are indeed causing problems to well-established activities such as hospitality and transportation. Lack of regulation, especially in these business sectors, put hotels and taxi drivers in a disadvantaged position.

1.1 Motivation

Within the sharing-economy spectrum, we focus on transportation, and in particular, on *ride-sharing* and *crowdsourced delivery*. Both transportation modes rely on private drivers to serve transportation requests. Ride-sharing drivers transport people, whereas in crowdsourced delivery, drivers transport goods from stores to customers. Our work proposes new ride-sharing and crowdsourced delivery models, and the reasons are as follows:

1. *Transportation* has a paramount role in many aspects of our societies; however, it is still inefficiently managed. People get connected, economic growth is propelled, and mobility is enabled by efficient transportation. Yet, we have been observing the opposite to efficient transportation in the last decades. For example, Figure 1.1 shows some statistics in Australia. In the top-left chart, vehicle ownership has been increasing steadily. Therefore, the more owned vehicles, the less vehicle occupancy rate, as shown in the top-right chart. There are environmental consequences reflected in the increment of the amount of carbon emissions generated by transportation (bottom-left). This vicious chain: more vehicles, then more congestion, then more pollution, affects the economy as well. In the bottom-right, avoidable

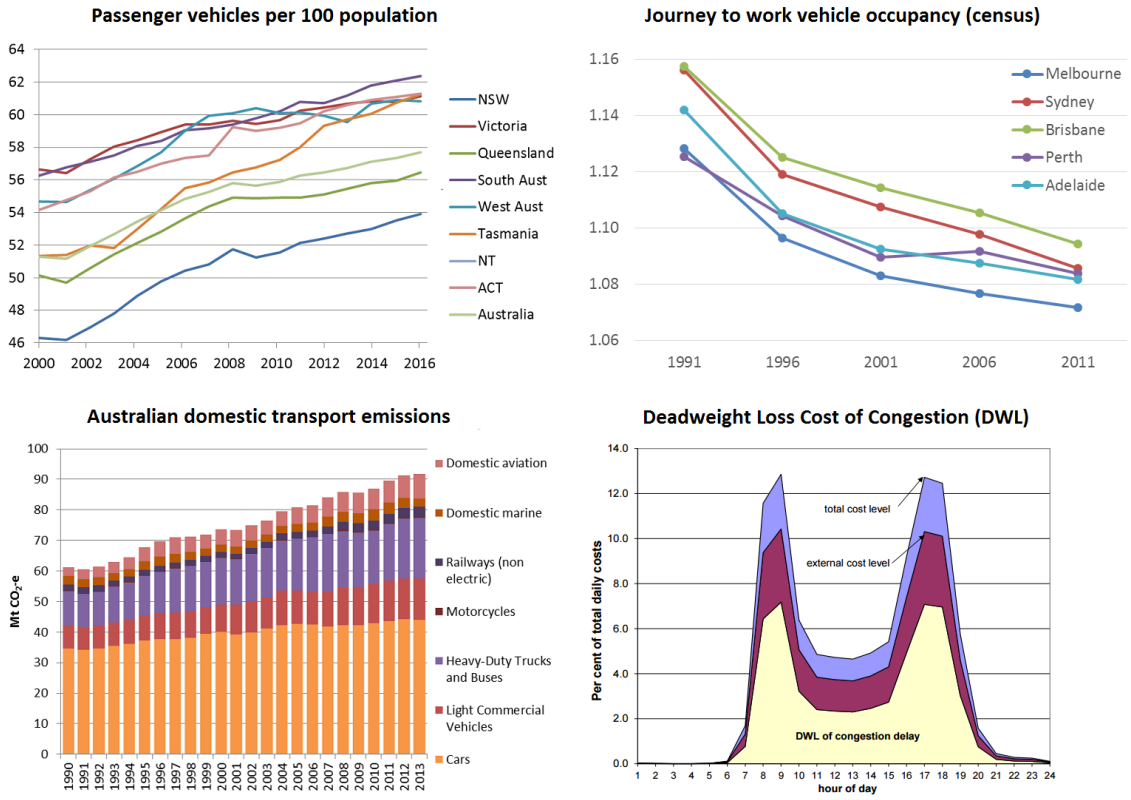

Figure 1.1: **Top-left.** Vehicle ownership: (passenger vehicles) / (population). **Top-right.** Vehicle occupancy: (vehicle drivers + vehicle passengers) / (vehicle drivers). **Bottom-left.** Transport was responsible for 17.2% of total Australian emissions in the year to September 2014, an increment from around 15% in 2002. **Bottom-right.** Deadweight Loss Cost of Congestion: the increment in net social benefit if appropriate traffic management schemes were introduced and optimal traffic levels were obtained. *(Bureau of Infrastructure, Transport and Regional Economics, 2016).*

costs throughout the day due to congestion are shown. This translated into about 16.5 billion of avoidable costs for the 8 Australian capitals until 2015 [13].

2. *Ride-sharing*, the emblematic sharing-economy transportation model, is not widely adopted albeit its benefits. This may be surprising given that Uber, a disruptive self-claimed ride-sharing company, has millions of users around the world. What Uber offers through its most popular product UberX is not ride-sharing, it is instead ride-sourcing. Ride-sourcing is exactly what traditional taxi companies offer but with the difference that the driver is a private subcontractor. The drivers do this for a living and thus, they are roaming over the city waiting for requests. The advantages these

companies claim because they are supposedly under the umbrella of the sharing economy are misleading. On the other hand, true ride-sharing occurs when two or more people, whose routes generally overlap, share a ride on the driver's own vehicle. In ride-sharing, the driver's motivations are different ranging from being environmentally conscious to getting allowed into high-occupancy-vehicle lanes. Despite ride-sharing benefits, its adoption has been limited.

According to Chaube et al. [15], ride-sharing is lacking convenience, incentives and trust. Ride-sharing is inconvenient for drivers because their routes and passengers' routes do not generally overlap and thus, drivers may need to make large detours to pick up passengers. It is also inconvenient for passengers who do not easily find a ride-sharing trip when their destination is in a low-density populated zone, e.g., going back home. Ride-sharing does not have incentives for participants as there are not enough fellow passengers with whom share costs, for the same reason that ride-sharing is not popular. Moreover, the fact of sharing a ride with a stranger raises safety concerns, which is a rather important factor why people do not trust in this service.

3. *Crowdsourced delivery*, although is a trend among large retail companies around the world [6,19,71], it still can be incorporated in their daily operations more effectively. We move onto the realm of transportation of goods within the sharing economy as it has become a very important business strategy for large retailers such as Walmart. Efficient delivery means a competitive advantage over a rival. In 2018, Walmart partnered with Bringg, a delivery orchestration platform, to manage the backend of Walmart's crowdsourced program called Spark Delivery [12]. Integration between Walmart's delivery platform and Bringg enables the dispatching of orders which includes their assignment to drivers. However, the way this integration occurs still considers drivers to make deliveries for a living. As a consequence, since drivers do not have a defined destination, i.e., they roam waiting for delivery tasks, their routes are not part of the overall optimization performed by the retailer. Suboptimal routes are generally computed under this scenario, which hits back the retailer as delivery times may be compromised.

Governments around the world have agreed to considerably decrease their carbon emissions given the alarming environmental situation. Transportation is one of the largest CO_2 contributors, and although some other initiatives have been proposed to alleviate congestion and pollution, sharing-economy models are the best option to put a curb on the problem. However, both ride-sharing and crowdsourced delivery, in their conceived form, are not convenient nor provide incentives. On the other hand, ride-sourcing and delivery-sourcing are popular. An increasing number of drivers sign up to Uber and Spark Delivery given the benefits these companies offer to their participants. The problem is that these companies, which wrongly adopted the sharing-economy label, are increasing congestion and thus pollution [22, 45, 58, 62]. The research in this thesis can empower our society to transition to true sharing-economy transportation models. Such models must be more convenient, trustworthy and provide more incentives than they currently do if we expect them to be massively adopted and thus, observe environmental, economic and societal benefits.

1.2 Challenges

Sharing economy, in particular, ride-sharing and crowdsourced delivery, poses several challenges, societal as well as technical ones. This thesis sets out to address some computational challenges that need to be overcome to enable efficient and effective algorithms for true ride-sharing and crowdsourced delivery. The main challenges are the following:

- *Finding a balance between drivers, passengers and the service provider*. Such a balance is difficult to attain. A passenger prefers the shortest possible trip once they request a trip. A driver, however, also aims to have the shortest trip, which in turn means that the detour should be as small as possible. The service provider on the other hand—company, government or whatever entity is connecting the individuals—would like to maximize the utilization of their fleet in order to maximize their revenue. These three goals are often conflicting with each other.

- *Adoption of true ride-sharing is paramount*. Ride-sourcing is a well-established model run by large companies that is part of many people's lives. New ride-sharing mod-

els not only have to provide incentives for drivers and passengers but also for companies to enable rapid adoption. This is, again, about finding a balance between the participants, and in particular, between companies, which may want to continue using the profitable current models, and society—governments—which must aim for sustainable models.

- *Transportation is not exempt from huge complexity*. Computational models of transportation involve a large number of parameters. First, we need to decide between macroscopic, mesoscopic and microscopic traffic models. Equally important is the representation of the road network. We opted for a graph representation, yet there are other considerations to make. The most important are whether the graph is directed, weighted or its edges have capacities. In the case the graph is weighted, we should decide which aspect the weights are modeling, i.e., distance or travel time. Weights can also change throughout the day—as it does happen in real road networks when weights represent travel time—and with different traffic load. Are we modeling queue buildup and traffic lights at the intersections? Are the vehicles moving at constant speed?

- *Query response times are expected to be quick, and quality of travel plans, high*. Users expect real-time solutions or at least in no more than a few seconds. However, the assignment of transportation tasks to drivers as well as routing of drivers in both, ride-sharing and crowdsourced delivery, are rather difficult tasks. New algorithms must be able to cope with such a level of complexity and offer decent solutions in near real-time. Users will find the platforms less compelling otherwise.

- *True crowdsourced delivery has not materialized.* Large retailers expect high delivery rates at the minimum cost. High expectations also extend to online customers since "same-day" delivery is becoming a baseline for them. Similar to ride-sharing, crowdsourced delivery has been misconceived. It is thought to be about private drivers wandering while they wait for delivery tasks. The goal instead is to get crowds to deliver products as part of their daily routes, not for a living. Yet, at the same time, the drivers must satisfy the retailer delivery expectations.

1.3 Our Research Questions

Our research questions are presented below in chronological order. The first two are based on *activity-based* ride-sharing, a model introduced independently by both our previous work [44] and Wang et al. [73]. In activity-based ride-sharing, participants ride-share to a common destination where they can perform their desired activity. This common destination is computed by an algorithm, and it is the one that minimizes the system travel cost.

1. Previous work on ride-sharing with meeting points [2,32,60] has shown the benefits of introducing meeting points in ride-sharing arrangements. In our previous work [44], we also built our new model, activity-based ride-sharing, including meeting points. In all these works, meeting points do not have specific characteristics more than helping save travel distance. However, not every intersection in a city is a suitable meeting point, e.g., when ride-sharing participants want to park their vehicles in these places and board another vehicle as part of their ride-sharing trip. The key question is whether *suitable meeting points, besides being proper places to park, make ride-sharing more effective.* In particular, we investigate whether the availability of *hot-spots*, as we call these suitable meeting points, helps solve the round-trip problem. In the round-trip problem, passengers may find a one-way ride-sharing arrangement but finding a round-trip one becomes more difficult, especially when going back to a less populated zone.

   Furthermore, in our previous work [44], our new activity-based ride-sharing model was represented as a set of instances of a new graph-theoretic problem we called Variable Steiner Tree (VST), a variation of the classical Minimum Steiner Tree (ST) problem. In the ST problem, we must span a subset of vertices of a graph, called terminals. We may span additional vertices, called Steiner vertices, to minimize the cost of the tree. The goal in VST is again to find the minimum-cost tree that spans a set of fixed terminals—users—but a variable root vertex—a common activity-based destination. Therefore, by constraining the set of intermediate meeting points to be hot-spots only, we analyze which *effect this new constraint has on VST, and whether*

the resulting new graph problem is still NP-hard.

If the problem is as hard as VST, we investigate whether *our proposed algorithm in [44] could be extended to deal with the new constraint put on meeting points or designing a hot-spot-based algorithm is more effective and efficient*.

2. In our previous work [44], and even in the work with hot-spots, the travel plans computed with our algorithms were considered independent of each other. That is, we omitted the influence, in terms of travel time, each of them has over the others. In this research question, we set up a more realistic scenario where activity-based ride-sharing travel plans are inter-dependent. If travel plans are computed without considering the other contemporary plans, some road segments may end up congested. Therefore, we proposed a congestion-aware ride-sharing model and explore whether *our non-congestion-aware approaches can be extended to deal with congestion*. We argued that a *meta-algorithm that invokes our previous approaches can efficiently compute effective congestion-aware travel plans*.

   Finally, we investigated how this model can be represented as a graph. This is motivated by the fact that there is a related graph problem in telecommunication networks called the Multicast Congestion (MC) problem [10,39,40,69], where the goal is to find the set of STs that overlap the least. The aim is to find whether *our resulting graph representation generalizes this problem as VST does with ST*.

3. In this last research question, we focus on issues on crowdsourced delivery—the goods-based ride-sharing version. Large retailers employ private drivers to make deliveries to its online customers. Their pools of drivers are crowdsourced in the same way as in ride-sourcing. Thus, congestion, evidenced in ride-sourcing, is replicated in this pseudo-crowdsourced delivery model. This is exacerbated by the fact that retailers select the pick-up stores for each order without considering drivers' routes. Of course, retailers operate in this way since they do not have this information—drivers roam waiting for delivery tasks. Yet, retailers expect to optimize drivers' routes as they are directly related with delivery times. In this work, we investigate whether *a crowdsourced delivery model through concurrent optimization of pick-up stores*

and delivery routes, both based on drivers' routes information, can yield a multi-fold positive result: (a) drivers do not roam over the city, (b) further optimization of delivery times is achieved, and (c) delivery promises are met without resorting to many dedicated drivers, e.g., drivers hired from a fleet company.

In terms of graph representation, this new proposed crowdsourced delivery model can be considered as a new version of the Pickup and Delivery Problem (PDP). However, in PDP there is a single pick-up and a single delivery location for each order, whereas, in our model, due to the concurrent selection of the pick-up store and computation of the delivery route, the pick-up location for each order can be any store. We look into the *complexity of the resulting new Hamiltonian path problem and an approach to solve it to optimality*.

1.4 Contributions

Our main contributions are novel models within the context of *ride-sharing*, specifically in *activity-based* ride-sharing, and *crowdsourced delivery*. The changes proposed through these new models are effective as further savings in travel costs are achieved. Associated benefits in terms of convenience, safety and reduction of congestion and pollution are also observed. The representation of our models derived into new graph-theoretic problems. Well-known graph problems such as Steiner trees and Hamiltonian paths are related with our new problems, yet they are not able to abstract the complexities of our proposed models. We introduce efficient approximate algorithms to solve our graph problems and by doing so, near-optimal travel plans are found. Our contributions are presented below in chronological order.

1.4.1 Activity-Based Ride-Sharing Model with Hot-Spots

Activity-based ride-sharing with intermediate meeting points is a new model introduced in our previous work [44]. It considerably outperforms the traditional model of ride-sharing that we call *trip-based* ride-sharing—see Figures 1.2 and 1.3. However, in this activity-based model with meeting points, there is no distinction whether such meeting

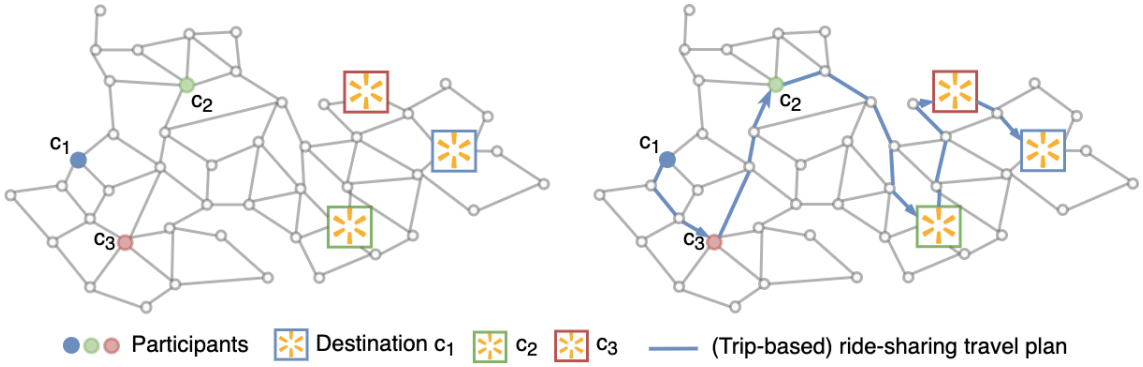

Figure 1.2: **Left.** Three ride-sharing participants c_1, c_2 and c_3 have informed their origins and target shopping destinations—indicated by the same border color. **Right.** Driver c_1's travel plan is depicted with blue bold edges. This setup is what we call *trip-based* ride-sharing since participants have defined their trips through origin and destination. Note how inconvenient it is for the driver due to large detours.

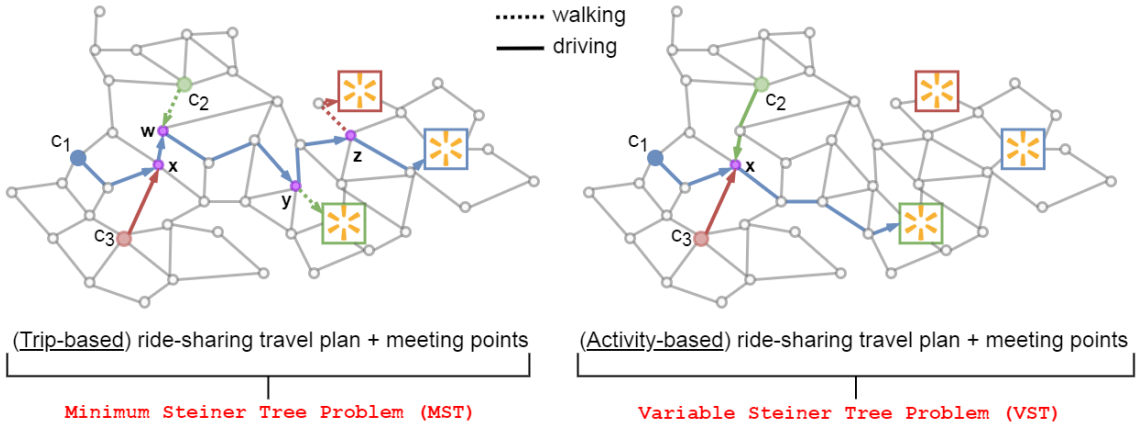

Figure 1.3: Based on the scenario in Figure 1.2, more convenient ride-sharing models are presented. **Left.** *Trip-based ride-sharing + meeting points:* Participants agree to meet at intermediate points. Users c_1 and c_3 drive towards x where they meet, and c_3 boards c_1's vehicle. User c_2 walks towards w where she meets c_1 and c_3. Then, all travel together until y where c_1 drops off c_2. After that, c_3 is dropped off at z. From y and z, the passengers walk towards their target stores. Finally, c_1 drives towards his destination. In this plan, a balance between c_1's detours and passengers' inconvenience is reached. Finding the optimal travel plan now becomes an instance of the *ST* problem due to the inclusion of meeting points. **Right.** *Activity-based ride-sharing + meeting points:* All participants agree to rideshare to a common destination provided that they can perform their intended activity, namely shopping. This is our contribution presented in previous work [44]. Note how the travel cost and inconvenience have been reduced. Finding the optimal travel plan is an instance of a new version of the *ST* problem that we call *VST* problem.

points are suitable or not for parking. In this thesis, we propose a model that constrains the intermediate meeting points to be only suitable locations for parking. We call these suitable points, *hot-spots*—see Figure 1.4. Although it is a seemingly subtle change, we not only make ride-sharing more convenient, more trustworthy and provide more incentives, but also, we present a solution to the *round-trip* problem that passengers may experience in ride-sharing. The following contributions are related to this model:

- *Constrained Variable Steiner Tree (C-VST) Problem.* Finding an optimal travel plan under this model is equivalent to solving a new version of the VST problem, a graph-theoretic problem we proposed in our previous work [44]. In this new version, the set of intermediate meeting vertices is constrained. Although C-VST works on this reduced set, we prove it is still NP-hard.

- *An Approximate Algorithm to Solve the C-VST Problem.* Our algorithm's insight lies in how intermediate meeting points are chosen. A meeting point's "gain" is evaluated for a subset of users. The meeting point that offers the best gain is included in the solution. Such gain is a ratio function that we prove is monotonic nondecreasing under a specific update rule. Our algorithm is at least as effective as a modified version of the state-of-the-art solution presented in our previous work [44]. This state-of-the-art algorithm is modified to cope with the constraint put on the meeting points. Furthermore, our algorithm is two orders of magnitude faster than this modified algorithm, and this is due to the monotonic nondecreasing trait of our gain ratio function.

1.4.2 Congestion-Aware Activity-Based Ride-Sharing Model

In this model, we dropped the assumption of independence between travel plans that we made in our previous work [44]. Now, this model takes into account the effect, in terms of travel cost, each activity-based travel plan has on each other when they overlap—see Figure 1.5. This effect is captured by considering the travel cost of each road segment as a nonlinear and strictly increasing function of traffic load on the road segment. This model leads to the following contributions:

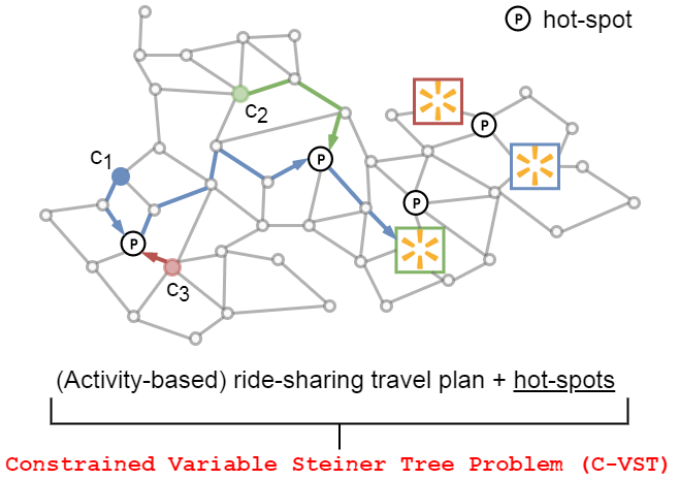

Figure 1.4: Based on the scenario in Figure 1.2, we present our first contribution. Our new model acknowledges that, in the model presented in our previous work [44], not every intersection is a suitable place to park a vehicle—intersections with parking ability are denoted by Ⓟ. Thus, point x in Figure 1.3 is not a proper parking space and our new model requires meeting points to be only suitable (parking) places. We call them *hot-spots*. Although the travel costs for this particular plan are increased compared with the plain activity-based + meeting points model's solution in Figure 1.3 (right), this new model offers more convenience, more incentives, and safety. Also, the round-trip ride-sharing problem is more likely to be solved. Finding an optimal travel plan in this model is equivalent to solving an instance of our new proposed graph problem *C-VST*.

- *A Generalization of the Multicast Congestion (MC) Problem.* The MC problem is a similar problem in telecommunication networks. When different sources multicast to their own sets of destination computers in the same network, congestion may be observed in some segments of this network. Thus, the goal of the MC problem is to find the set of STs that connect the computers within each set with their correspondent source with the minimum overlapping. Since the VST problem is a generalization of the ST problem and because our goal in this model is to find the set of VSTs that overlap the least, we are generalizing the MC problem.

- *A Congestion-Aware Approximate Meta-Algorithm.* Increasing the cost of the most demanded products is generally an effective way to gain money as a seller but also has the side-effect of getting your customers to think of another provider. This analogy explains the intuition behind our meta-algorithm. We increase the cost of the most used edges according to a nonlinear strictly increasing function such that in the next iterations, these busiest edges start losing their appeal. Then, alternative routes start emerging.

1.4.3 Adopting Activity-Based Ride-Sharing in Crowdsourced Delivery

Currently, large retailers choose the pick-up store for each online order without taking into account the drivers' routes. Private drivers pick up the orders from the already chosen stores and deliver them to the online customers. In our proposed crowdsourced delivery model, we adopt the idea of generalization from activity-based ride-sharing. Now, the private drivers, instead of picking up the orders from fixed—chosen by the retailer beforehand—stores, they have a set of stores to pick up each order from. In our novel model, the routing of the drivers and the selection of the pick-up stores are concurrent. Drivers provide their origins and destinations and thus, they do not wander on the roads waiting for delivery tasks. Drivers' optimal routing is sought in an exponential larger space—see Figure 1.6. This, in turn, guarantees cheaper delivery routes compared with the current crowdsourced delivery models. This model leads to the following contributions:

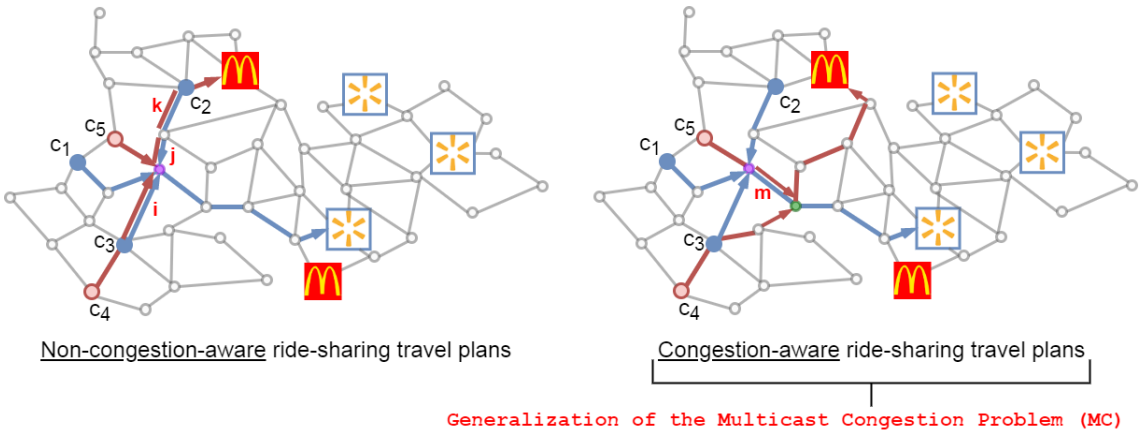

Figure 1.5: Two groups of participants categorized by their preferred activity want to share an activity-based ride. The first group has users c_1, c_2 and c_3 who want to do shopping, whereas users c_4 and c_5 want to have fast food. Their activity-based travel plans are shown in bold edges of different colors. **Left.** The plans were computed, each independently of the other, by using our VST-based Ride-Sharing (VST-RS) algorithm proposed in [44]. Edges i, j, and k are congested—we assume that congestion is reached with only two vehicles. This scheme is non-congestion-aware. **Right.** Weights of the most used edges are updated iteratively causing alternative travel plans to emerge. As a result, only edge m is congested. Although the travel plan seems more expensive—compared with the plan on the left—it is the opposite since the cost is a function of traffic load. Finding the set of activity-based travel plans that overlap the least is equivalent to solving a generalization of the *MC* problem.

- *Group Hamiltonian Path with Precedence Constraints Problem (GHPPCP).* The classical Hamiltonian path problem is about finding a path which traverses every vertex in a graph once. In crowdsourced delivery, stores and customers are the vertices to be visited. Our new Hamiltonian path problem requires that the driver's path visits all her assigned customers and only the most optimal store that is chosen from a "group" of stores of a retailer. Precedence constraints must be enforced between vertices since a driver must visit a store before a customer. Our goal is to find the minimum-cost path under these conditions.

- *An approximate approach to multiple instances of the GHPPCP.* Computing the delivery route of a driver corresponds to solving only one instance of the GHPPCP. Generally, there are many requests—more than a single driver can serve. Therefore, before routing, customers must be assigned to drivers. All drivers' delivery routes are to be computed, which means solving multiple instances of our GHPPCP. Our two-stage approach first computes approximate assignments and then computes optimal delivery routes for each driver. The assignment stage is based on the intuition of proximity of customers to the original route of the driver. That is, a driver is assigned the closest customers to her route. In the routing stage, a personalized branch-and-bound algorithm is used to compute optimal delivery routes. This stage may be computationally prohibitive if the number of assigned customers per driver is large. We developed a technique of load balancing of assignments that is performed in such a case.

1.5 Outline

The rest of this thesis is organized as follows:

- **Chapter 2:** Concepts such as *activity-based ride-sharing*, *ST*, *VST*, *MC*, *Hamiltonian paths*, among others, used throughout this thesis are explained in this chapter. We also include key insights from previous work I was a co-author of [44], which is not

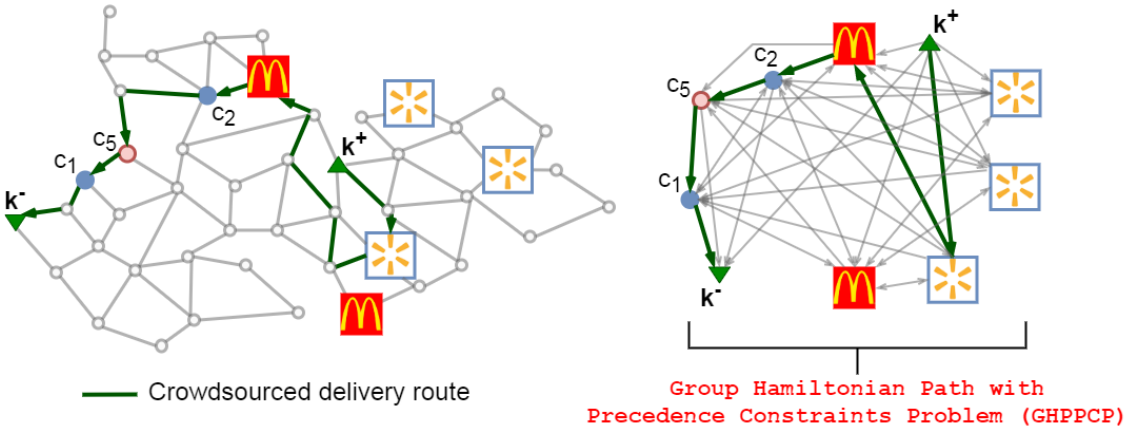

Figure 1.6: A private driver is on his way home from k^+ to k^-. There are two Walmart online customers c_1, c_2. There is also a McDonald's online customer c_5. The driver can pick up the orders from *any* Walmart and *any* McDonald's stores. **Left.** The whole road network and the optimal crowdsourced delivery route in bold edges are depicted. **Right.** A directed subgraph is constructed. Its vertices correspond to the union of customers, stores, and start and end locations of the driver. Its arc set satisfies a priori constraints of the problem, e.g., a driver cannot go to a customer before visiting the corresponding store. Finding an optimal delivery route in this model is equivalent to solving an instance of the *GHPPCP*. This subgraph is part of the input for the GHPPCP. The optimal delivery route in bold edges is also depicted in this subgraph.

regarded as a contribution chapter, yet the contributions presented in this thesis are based on it. Furthermore, along the concepts, relevant work is cited and explained.

- **Chapter 3:** Our *activity-based ride-sharing model with hot-spots, C-VST problem*, and its corresponding approximate algorithm *Gr-based*, are presented in this chapter.

- **Chapter 4:** Our *congestion-aware activity-based ride-sharing model, a generalization of the MC problem*, and the meta-algorithm *VST-based Congestion-Aware (VST-CA)*, are presented in this chapter.

- **Chapter 5:** Our *crowdsourced delivery model through concurrent routing and pick-up store selection, GHPPCP*, and our approximate approach $dist_{ra}$ – *BnB*, are presented in this chapter.

- **Chapter 6:** This chapter presents conclusions and future research directions in ride-sharing, crowdsourced delivery and related graph-theoretic problems.

1.6 Papers Resulted from This Thesis

- Correa, O., Ramamohanarao, K., Tanin, E. and Kulik, L., 2017, November. 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* (pp. 136-145). ACM.

- Correa, O., Tanin, E., Kulik, L. and Ramamohanarao, K., 2018, November. Activity-Based Ride-Sharing in Action (Demo Paper). In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (pp. 608-611). ACM.

- Correa, O., Khan, A.M.R., Tanin, E., Kulik, L. and Ramamohanarao, K., 2019. Congestion-Aware Ride-Sharing. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 5(1), pp.1-33.

- Correa, O., Tanin, E., Ramamohanarao, K., Kulik, L. and Zaslavsky, A., 2020. Cost-Effective Crowdsourced Delivery Through Concurrent Routing and Pick-Up Store Selection. *Proceedings of the VLDB Endowment*. Under Review.

Chapter 2
Background Theory

In this chapter, we present and explain concepts used throughout the thesis. Along with the concepts, we survey relevant work in ride-sharing and crowdsourced delivery. We present a ride-sharing taxonomy different from what the literature presents. Activity-based, our new model introduced in our previous work, is by itself a novel ride-sharing category. All the studies in ride-sharing prior to our work are part of a category that we call trip-based ride-sharing. We still present dynamic ride-sharing as an orthogonal category, because it is currently the prevalent model. Other forms of ride-sharing, such as carpooling, slugging, among others, are mentioned as well. We present the most important insights from our previous work—activity-based ride-sharing with meeting points—as it is needed to fully understand the contributions presented in later chapters. Crowdsourced delivery is the second part of this chapter. Our novel ride-sharing and crowdsourced delivery models have been represented through new graph-theoretic problems since classical graph problems were not able to capture the complexities of our new models. Brief explanations of such classical problems are presented.

2.1 Ride-Sharing

RIDE-SHARING is a sharing-economy transportation service where two or more people share the driver's vehicle. The driver's motivations may be multiple ranging from being environmentally conscious to getting allowed into high-occupancy-vehicle lanes to receiving a small fare. It is also usual that ride-sharers' routes overlap, which prevents the driver from having large detours. In Figure 2.1, two ways of doing ride-sharing are shown. In both, a driver and two passengers are willing to ride-share. In the first, the driver picks the passengers up from their source locations and drops them off at their destinations. In the second, all participants agree to meet at intermediate points, which makes the driver's route shorter, yet this arrangement is still convenient for the passen-

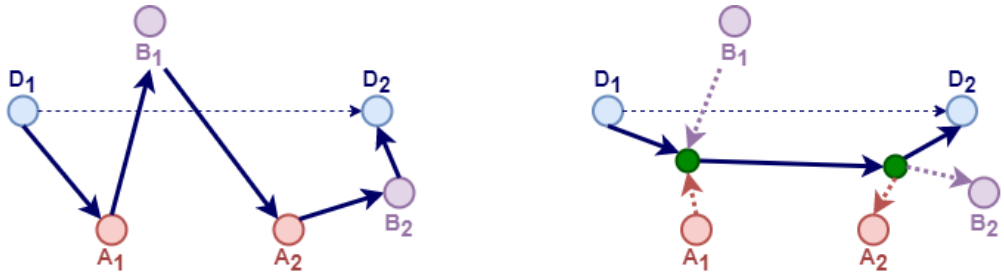

Figure 2.1: A driver D starts from D_1 towards D_2. Her original route is shown with a dashed line. Two more people A and B have similar routes as D, and they want to ride-share. Two ways of doing ride-sharing are presented. **Left.** *Classical ride-sharing*, where driver D picks them up from their source locations, A_1 and B_1, and drops them off at their destinations, A_2 and B_2. **Right.** *Ride-sharing with meeting points*, where users agree to meet at intermediate meeting points. Passengers walk to these points and when they are dropped off, they walk to their destinations. The driver does not detour as much as before.

gers. *The key in ride-sharing is not only that more than two people share the vehicle but also that the drivers do not wander across the network to serve riders for profit*.

Conceived in this way, ride-sharing does have the potential to be a real solution to congestion and pollution caused by vehicles. However, the ride-sharing concept has been distorted. Transportation Network Companies (TNCs) such as Uber, Lyft, and DiDi appear in the market as ride-sharing companies, however, they offer *ride-sourcing*. Ride-sourcing is a more accurate term because these companies outsource private drivers and dispatch them as taxis. Generally, they charge cheaper fares than actual taxis. With this incentive, ride-sourcing demand has steadily grown. This, in turn, has attracted many people to work as independent drivers. This vicious cycle has made ride-sourcing become the main source of congestion in San Francisco [22, 62], New York [58] and even in Europe [45].

TNCs do offer alternative products that encourage sharing, e.g., UberPool by Uber and Lyft Share by Lyft. These alternatives are not ride-sharing modes, though. They are technically called *ride-splitting* modes. According to the work presented by Santi et al. [56], taxi-sharing—equivalent to ride-splitting but with taxi companies—may enable massive savings. The authors analyzed the data of millions of taxi trips in New York City, and their empirical results show high percentages of shared trips ($\approx$ 100%) and saved

travel time (32%) if passengers are willing to increase their travel time up to 5 minutes and at most two trips are shared. To assess whether the results were particular to the well-known high taxi density of New York or they can be generalized to other low-taxi-density cities, the authors sub-sampled the dataset. The results were as good as with the whole dataset size. Tachet et al. [63] confirmed Santi et al.'s findings through a simple model that predicts the potential for taxi-sharing in cities with diverse sizes and traffic characteristics. Yet, taxi-sharing (resp. ride-splitting) still has drivers deadhead around the city and thus, it cannot be considered ride-sharing nor a solution to congestion and pollution.

On the contrary, the widespread use of ride-sharing has been restricted by several challenges. In ride-sharing, it is preferable to have overlapping routes. If routes are not overlapping, the driver is forced to make large detours, particularly within classical ride-sharing arrangements—see Figure 2.1 (left). This is not convenient for the driver since the driver does not generally charge a kilometer-based fare. Recall that, in true ride-sharing, drivers are not-for-profit. To alleviate this problem, researchers have started studying the benefits of intermediate meeting points in ride-sharing—see Figure 2.1 (right). All of them have concluded that more savings are achievable under this model compared with classical ride-sharing.

In ride-sharing with and without meeting points, passengers are required to inform their starting location and destination. This is what we call *trip-based* ride-sharing. Most of the work in ride-sharing is trip-based. In the following section, we survey this work. First, we present the classical trip-based ride-sharing mode. Then, we present work on its counterpart that includes meeting points. We do not constrain our survey to be within the true ride-sharing realm only as we also present work on taxi-sharing.

2.1.1 Trip-Based Ride-Sharing

Back in the 1970s, several ride-sharing methods emerged due to the oil crisis in the U.S. Private companies such as 3M and Chrysler were the first to introduce *vanpooling* programs—these companies provided vans for employees to commute to and from the workplace. Then, employees started to organize carpools where they took turns to drive

others [28]. *Carpooling* service, which usually requires long-term commitment among employees, is within the trip-based ride-sharing category. In this case, origin and destination are fixed, and for a long time—therefore carpooling is also known as *recurring ride-sharing*. Although carpooling was a popular transportation mode, it experienced a sudden and massive decay in the eighties. A possible cause is purely economic in nature, namely the price of gasoline. The increment in family income and vehicle availability are regarded as parallel causes [26].

With the increment in vehicle ownership rates, people expected more convenience when traveling as passengers. The gradual incorporation of more flexible transportation modes was observed. First, *flexible carpooling*, a semi-organized ride-sharing alternative, was common since no prearrangements nor fixed schedules are needed. Rather, ride-sharing trips are originated spontaneously at predetermined locations on a first-come-first-served basis. The driver benefits as she can use high-occupancy-vehicle lanes and toll costs are split. Yet, its main limitation lies in the lack of flexibility of its routes, which does not allow door-to-door transportation. This practice is also known as *slugging* or *casual carpooling*. Secondly, with the help of the Internet, private matching ride-sharing agencies appeared that started to publish the offers and requests made by drivers and passengers. In the best case, algorithms were run to match batches of offers and requests. In other cases, participants had to contact each other to agree on a schedule. This is what we call *static ride-sharing* as all the requests are known before route planning. Even though routes flexibility and communication between participants improved, the ride-sharing decline continued [26]. Currently, most of the efforts are focused on trip-based *dynamic ride-sharing*.

Dynamic ride-sharing enables drivers to serve requests on short notice or even en-route. Therefore, participants do not have to publish their trips well in advance, contrary to what happens in carpooling and the static version of ride-sharing. A matching algorithm assigns riders to a driver automatically by considering time, detour tolerances and personal restrictions even if the driver has already started her route. The earliest attempt to approach dynamic ride-sharing was by Winter and Nittel [76]. This work considered an agent-based system that aims to maximize the number of served riders. Winter and

Nittel [76] showed that negotiation within a mid-range ad hoc wireless network—no central communication and no planning component—can be used to plan ride-sharing trips without affecting quality.

Agatz et al. [1] developed an approach with a different objective: minimize the total system-wide vehicle distance while giving service guarantees to the participants. That is, all participants are guaranteed to depart from their origins and arrive at their destinations within their indicated time windows. They also found that dynamic ride-sharing may even be sustainable with relatively low participation rates within sparse urban areas with many employment centers. This is particularly interesting as one of the reasons for carpooling decline is thought to be this multinucleated urban form, which has become a trend in American cities [26].

Large-scale dynamic taxi-sharing was approached by Tian et al. [65], Ma et al. [46,47] and Huang et al. [37]. All these works presented systems with service guarantees, and for multiple passengers on board. Tian et al. [65] presented Noah, an implementation of Huang et al.'s kinetic tree [37]. A kinetic tree is a structure that maintains all the valid trip schedules with respect to the driver's current location. It enables efficient handling of new requests compared with commonly used approaches such as branch and bound and mixed-integer programming. Ma et al. [46] presented T-Share, aimed at minimizing the total travel distance considering a fixed fleet size. T-Share also includes a pricing scheme designed to charge a passenger properly and provide taxi drivers with more profit.

Pricing is another constraint ride-sharing passengers can include in their requests. SHAREK, a ride-sharing service proposed in [4,14], allows passengers to specify not only time windows but also the maximum price they are willing to pay. SHAREK estimates the cost for each driver based on the distance of the passenger's route plus the detour from the driver's original route to serve this passenger. The drivers who are dominated by others, i.e., drivers who charge high costs and make passengers to wait longer, are pruned from the list of candidate drivers. Then, further three-phase pruning, which considers the temporal and price constraints, is performed without the need to calculate shortest paths. This work as well as the large-scale taxi-sharing systems presented before agreeing on how expensive shortest paths computation is. For example, Noah [65] implements a

caching scheme to avoid the very frequent shortest path computation, characteristic in large-scale routing problems. T-Share, on the other hand, implements a spatio-temporal index where distances (resp. travel times) between entries are pre-computed. SHAREK does not compute routes but acts as a plugin for already existing services.

As mentioned before, ride-sharing widespread is yet to happen and lack of incentives can be considered as one of the reasons. If effective incentives are provided by rewarding participants for being more flexible, critical mass can be attained in ride-sharing. Stiglic et al. [61] studied the extent to which three different types of participant flexibility impact ride-sharing performance. The authors showed that participant flexibility is a rather important factor to increase matching rate, especially in low-participation-rate systems. As in Stiglic et al. [60], a hierarchical optimization approach that first maximizes the number of matched participants and then the system-wide distance savings is used in this work.

Most of the ride-sharing (resp. taxi-sharing) approaches consider passenger vehicles only. Alonso et al. [5] presented an optimal method which also applies to shared vans and minibuses. Their approach decouples the problem by first computing feasible trips and then assigning these trips to drivers. The authors claim that instances such as New York, with thousands of requests, can be solved in real-time. Their simulations show that high-capacity taxi-sharing yields a reduced fleet of vehicles—below 25% of the active taxis in New York City—able to satisfy 99% of the requests, with a mean waiting time and delay of about 2.5 min. Also, the authors concluded that lower fleet size with larger capacity and longer waiting/delay times—this last point confirms the findings by Stiglic et al. [61]—increase the possibilities for ride-sharing. Obviously, the increment in capacity affects travel time, yet the authors found such an effect to be negligible.

Autonomous Vehicles (AVs) are being tested extensively and they are gradually showing their proficiency on the roads, which may catalyze ride-sharing ubiquity. Fagnant and Kockelman [25] performed simulations of shared AVs within an idealized city and across Austin, Texas. The authors concluded that each shared AV could potentially replace about 11 conventional vehicles, yet at the cost of increased traveled distance. However, this is still better compared with the scenario where dynamic ride-sharing is not allowed. This work also offers a benefit-cost analysis for ride-sharing providers that includes optimal

fleet sizing. It shows that shared AVs could be quite profitable: A fleet operator is simulated to achieve a substantial 19% return on his/her investment.

Up to this point, we have discussed work on the classical ride-sharing (resp. taxi-sharing) model where a driver picks up passengers from their source locations. That is, passengers do not head to intermediate locations to meet with the drivers to alleviate detours. Most of the work on ride-sharing with meeting points highlight the extra savings the inclusion of such a strategy yields to the system. It is worth to mention that recently, Uber introduced their new service called UberPool [67], which asks the riders to go to a walking-distance meeting point to be picked up. This is anecdotal evidence about the importance meeting points have in ride-sharing. In the following section, we survey some of the work on trip-based ride-sharing with intermediate meeting points.

Trip-Based Ride-Sharing + Meeting Points

Xing et al. [77] presented SMIZE—a multi-agent ride-sharing system—that, besides driver and passenger agents, relies on routing and access-to-database-like agents to match participants and compute their routes respecting maximum response times and user preferences, e.g., gender, smoking. The routing agents also compute walking routes for passengers whose start or destination locations are in a pedestrian zone. These meeting (resp. dropping off) points—called boundary nodes by the authors—are computed by their approach. In case the passenger's start location is within a pedestrian zone, she walks to the boundary node to take a ride, or if her destination is in the pedestrian zone, she is dropped off in the boundary node from where she walks to her destination. The authors concluded that the number of drivers is a rather important factor concerning matching rate and that with enough drivers, travel times can be reduced significantly compared with public transport.

Olsen [51] considered a version of carpooling where users count on their vehicles which they are willing to drive to a "park-and-ride" lot and ride-share afterward to a common destination. His contribution is demonstrating that this problem is APX-complete[1] when it is restricted to vehicles with capacity 4 and unweighted undirected graphs. The

[1] Problems in the complexity class *APX* allow polynomial-time approximation algorithms. APX is in *NP*.

author discussed the complexity of the problem but proposed no algorithm to solve it efficiently. Takise et al. [64] also consider ride-sharing with a common destination. Olsen as well as Takise et al. and Zhang et al. [78] generalized the Minimum Steiner Tree (ST)[2] problem as our work does. However, Takise et al.'s algorithm resembles the seminal work of Dreyfus and Wagner [21], which computes the exact solution of the ST problem. Thus, their algorithm does not scale since its complexity is exponential on the number of users. On the other hand, Zhang et al.'s problem is different from ours because their problem considers different destinations.

Aissat and Oulamara [2] presented one exact and two heuristic-based approaches to find intermediate points that minimize the total travel cost while keeping detour costs reasonable. A parameter called detour factor determines the extent to which a detour can be still regarded as reasonable. 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. [60] showed 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 possibility of matching some riders with one driver. The authors showed that considering meeting points other than origins and destinations improves the matching rate and thus, the overall functionality of the ride-sharing system.

Goel et al. [32] presented the benefits of intermediate meeting points from the perspective of privacy. Their approach computes a Pareto front of optimal intermediate meeting points by maximizing two objectives, namely, coverage and k-anonymity[3]. Based on estimated population density, coverage circles with different radius are centered at intermediate meeting points which the authors call Pick up Points (PuPs). The authors' approach finds the PuPs that ensure privacy and safety to the users. At the same time, their approach significantly reduces the average vehicle travel distance per traveler.

Aïvodji et al. [3] also developed a privacy-preserving approach that computes meeting points in ride-sharing. The objective of their approach is to help the participants syn-

[2]The ST problem is explained in more detail later in this chapter.

[3]A user is spatially k-anonymous if her exact location is replaced with a spatial region that contains at least $k-1$ other users

chronize their itineraries without revealing their origins and destinations. Participants are put in contact by an external third party and then, to preserve privacy, certain operations, such as the computation of candidate meeting points, are performed locally. The two sets of candidate meeting points, one from the passenger and the other from the driver, are not revealed to each other, yet the common candidates are computed by using private set intersection techniques. Shared paths between candidate pick-up and candidate drop-off points are computed. Then, each shared path is assigned two costs, one from the driver and one from the rider, which are computed locally. Finally, the election of ideal pick-up and drop-off points is made by voting. The authors claim that their distributed approach offers solutions of quality comparable to centralized approaches.

The approaches above are all single-hop, that is, passengers ride-share once per trip. Drews and Luxen [20] presented a *multi-hop* ride-sharing paradigm where trips can include an arbitrary number of transfers while respecting driver and rider preferences. To reduce complexity, transfers can only occur on a predefined set of meeting points, which are called stations. The goal is to find the sequence of rides that allows the passenger to travel from the origin to the destination via multiple hops. The authors' approach adapts time-expanded graph techniques where a station is not one vertex but a set of vertices, each representing either departure or arrival time. Then, a best multi-hop fit can be computed by an earliest-arrival query in the graph. By tuning the parameters that control the extent of waiting times and detours, the authors observed reasonable matching rates. However, they also found that increasing the number of transfers to more that two does not yield a better matching rate.

Apart from [20], none of these studies regards their intermediate locations the way we conceive hot-spots. That is, the intermediate locations chosen by their algorithms are not constrained to a subset of suitable points. Eventually, their intermediate points could be located at any point as long as they satisfy the specific constraints imposed by their objectives. In the case of [20], although they do constrain the meeting points, their problem is different from ours as we do not ask riders to make transfers.

2.1.2 Activity-Based Ride-Sharing

Important part of this thesis is based on our previous work about *activity-based ride-sharing* [44]. Our previous work and Wang et al. [73] were the first to propose activity-based ride-sharing. In the activity-based ride-sharing model, a user submits a transportation request which contains the place where to be picked up and the activity to be performed. This differs from the classical trip-based ride-sharing model, where the user informs the origin and destination instead. Activity-based ride-sharing expands the set of possible destinations, which translates into increased matching rate between drivers and passengers [44, 73]. This is at the expense of giving up the preferred destination under the assumption that the user can perform the informed activity at any destination chosen by the algorithm.

In Figure 2.2 (left), a driver and a passenger are using trip-based ride-sharing services. Both have their preferred shopping stores. The driver starts at location D_1 and her preferred shopping store is at location D_2. The passenger starts at location P_1 and her preferred store is at P_2. They agree to meet at an intermediate point x and ride-share until y. From there, both travel independently to their preferred stores. In this example, the passenger must walk considerably from y to P_2. This risks the ride-sharing transaction and the passenger may opt for a different transportation mode.

In Figure 2.2 (right), the same driver and passenger are using an activity-based ride-sharing service. They still have their preferred shopping stores; however, they are willing to give them up and instead they inform their activity, which is `shopping`. Now, the algorithm searches the best destination where both can go shopping. In this case, the algorithm finds a neutral store, which may even be owned by another retailer. This does not prevent both participants from ride-sharing as they can still perform their informed activity. Moreover, the passenger does not have to walk, which encourages her to use the service.

The carpooling problem appears to be similar to activity-based ride-sharing in the sense that users agree to go to the same destination. Yet, there is an important difference. In carpooling, a single fixed destination is agreed on ahead of time, e.g., workplace, whereas in activity-based ride-sharing there are multiple destinations and they are

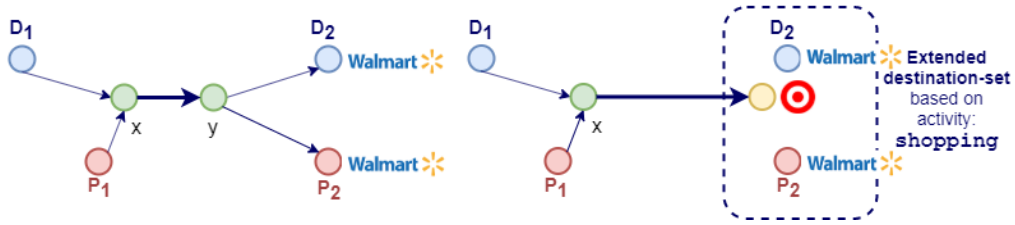

Figure 2.2: **Left.** *Trip-based ride-sharing:* A driver who starts from $D1$ has her preferred shopping destination D_2. A passenger located at P_1 prefers another store at P_2. Both use a trip-based ride-sharing service, thus they inform their origins and destinations. Their travel plan comprises to meet at x and ride-share until y. From there on, both travel independently to their preferred stores. **Right.** *Activity-based ride-sharing:* Both participants use an activity-based ride-sharing service and thus, they inform only their origins and preferred activity. The service computes the optimal destination to where they ride-share. The set of possible destinations for each user is extended.

known ad-hoc.

Wang et al. [73] designed ABRA, an activity-based ride-sharing algorithm that avoids combinatorial explosion due to the expanded the set of destinations, by including a space-time filter. The authors assume that the users have a complete list of full-day activities with predefined running order. To perform these activities, there is a chain of trips with time and detour budgets. Thus, the space-time filter compute alternative destinations based on such trip's space-time budgets. The second assumption is that AVs operate as taxis with maximum capacity of 2 passengers. The algorithm then optimizes the sequence of pick-ups and drop-offs. The matching process attempts to match a pair of trips from different trip chains and different users if they satisfy time budgets. 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 their goal is to show the increment in terms of matching rate with respect to trip-based algorithms, the authors do not aim to offer a readily deployable solution.

Activity-based ride-sharing was integrated with *social network-based ride-sharing* [74] into a holistic new model called *collaborative activity-based ride-sharing* in the work of Wang et al. [75]. This new model inherits the advantages of both models: the trust-based strategy of social network-based ride-sharing and the extended set of target destinations due

to participants' flexibility, from activity-based ride-sharing. The authors acknowledge the benefits social ties have as they may positively influence ride-sharing participants' decisions, yet the authors also warn that these ties might threaten matches with strangers located nearby. By considering the level of friendship—including strangers—different detour tolerances are assigned to potential partners. The authors showed that this new model increased the matching rate compared to both models if analyzed individually, i.e., social network-based and activity-based. However, their approach is not efficient as it occurred in their prior work where they tested the activity-based model [73].

Mahin and Hashem [48] proposed a variation in the activity-based model. The driver does not change her original route and it is assumed she visits at least one Point of Interest (POI) on her way. On the other hand, riders are flexible about their POI—it is not necessarily the same as the one the driver passes by—however, they do specify maximum detour and preferred chain within the POI type, e.g., if the activity is having fast food, the rider can specify which fast-food chain. Riders are picked up and dropped off at the driver's start and end locations—same as in slugging. A trip would be: First, riders go to the driver's start location, then all travel together to the driver's chosen POI from where riders head to their POIs, then when all join again at the driver's POI, they head to the driver's destination from where riders travel independently to their destinations. This variation guarantees riders will have a complete ride-sharing arrangement when they must visit one or more POIs as part of their trip. Yet, the authors' approach is for a single driver only.

2.1.3 Summary of Related Work in Ride-Sharing

Table 2.1 summarizes the related work in ride-sharing. Column **"Obj."** refers to the objective function the approach is optimizing. When the value in this column is `[D, T]`, the approach is optimizing a cost function that can be assumed to be either distance or time. This is possible when driving speeds are available because time and distance measurements can be converted from one to the other. The column **"True Ride-Sharing"** refers to approaches where the drivers have specified their origin and destination. That is, the drivers are not serving passengers for profit. Also, in this column, the approaches with

Table 2.1: Ride-sharing approaches. D = minimizing system travel distance, P = maximizing the number of participants, T = minimizing system travel time, O = minimizing other costs.

	Reference	Obj.	Constraints	Riders	Meeting Points	True Ride-Sharing
Trip-Based	Winter and Nittel [76]	P, T	time	one	-	hybrid
	Xing et al. [77]	T	time, personal	one	✓	✓
	Agatz et al. [1]	D	time	one	-	✓
	Ma et al. [46,47]	D	time	multiple	-	-
	Olsen [51]	O	-	multiple	✓	✓
	Tian et al. [65]	D, T	time	multiple	-	-
	Drews and Luxen [20]	D, T	time, detour	one	✓	✓
	Aissat and Oulamara [2]	D, T	time	one	✓	✓
	Huang et al. [37]	D, T	time	multiple	-	-
	Cao et al. [14], Alarabi et al. [4]	n/a	time, price	one	-	✓
	Stiglic et al. [60]	P, D	time	one	✓	✓
	Aïvodji et al. [3]	T	time	one	✓	✓
	Goel et al. [32]	P, D	time	one	✓	hybrid
	Stiglic et al. [61]	P, D	time	one	-	✓
	Takise et al. [64]	D	-	multiple	✓	✓
	Zhang et al. [78]	D	-	multiple	✓	✓
	Alonso et al. [5]	T	time	multiple	-	-
	Wang et al. [74]	P	time, detour	multiple	-	✓
	Fagnant and Kockelman [25]	P	time	multiple	-	-
Activity-Based	Wang et al. [73]	P	time	multiple	-	-
	Wang et al. [75]	P	time, detour	multiple	-	-
	Mahin and Hashem [48]	D	time, detour, personal	multiple	-	✓

value `[hybrid]` are those which, besides considering not-for-profit drivers, consider taxi-sharing or another public transportation mode.

Both trip-based and activity-based ride-sharing can be categorized as either static or dynamic. This characterization is given by how the approach deals with transportation requests. If the approach computes travel plans for the requests that are available at planning time only, it is considered as static. If the approach can schedule the requests that come up after the planning stage and still satisfies the constraints of the plans that are already in execution, the approach is within the dynamic ride-sharing category.

It is noticeable the emphasis given to dynamic ride-sharing in both trip- and activity-based ride-sharing. Our proposed approaches are static, and the reasons are two. First, by approaching the problems as static, we were able to reduce them to well-known graph-

theoretic problems and thus, our solutions to them are more generic. Second, we can process the available transportation requests in batch and the ones that become available later are stored to be processed in the next batch. This last workaround enables our solutions to be ready for deployment.

In the next section, we present the relationship of our activity-based ride-sharing models with graph theory. We also explain the related graph-theoretic problems as they are building blocks of the contributions presented in this thesis.

2.1.4 Activity-Based Ride-Sharing and Graph Theory

All our proposed models have been represented as new graph-theoretic problems which are NP-hard. That is, we cannot aim at having exact solutions to them in real-time nor even in a reasonable time. We, however, designed approximate algorithms that are readily deployable.

In our previous work [44], we generalized the destinations and the set of intermediate meeting points. Generalizing the destinations was achieved by including the concept of activities. Destinations in transportation requests are not fixed anymore. Thus, instead of having one destination, there is a set of possible destinations where the preferred activity, informed through the request, can be performed. This enables grouping people to get to common POIs. Also, we did not limit the intermediate meeting points to be located within walking distance. That is, users can drive their vehicles towards the agreed intermediate meeting points and then they get on others' vehicles to continue their trips. We showed that further savings in terms of traveled distance are achieved. Computing a travel plan in this model is equivalent to solving a new version of the ST problem. We call this new ST problem version, Variable Steiner Tree (VST) problem. Following, we explain these two graph-theoretic problems.

Minimum Steiner Tree (ST) Problem

An ST is a problem that was initiated in the Euclidean plane. Given a set of points in the Euclidean plane, the ST is the minimum-cost tree that spans the points in the set. The

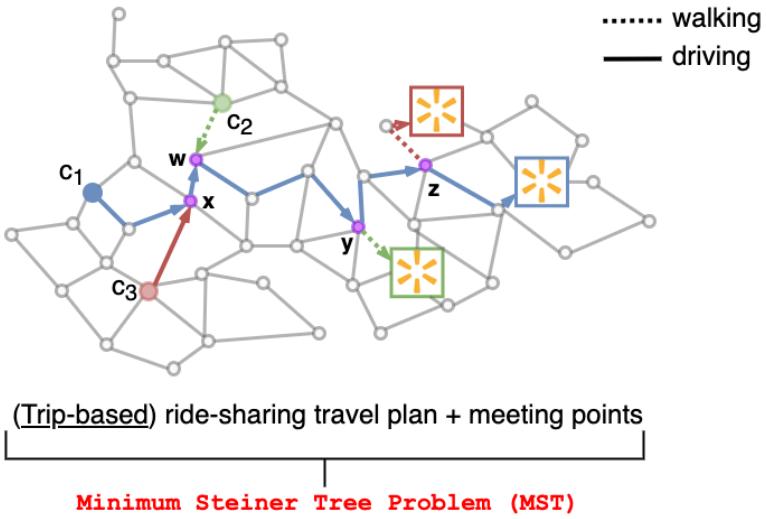

Figure 2.3: Users c_1, c_2 and c_3 want to ride-share. They have informed their preferred destinations—a Walmart store with the same border color as the user. They have agreed to be picked up and dropped off at intermediate points. Users c_1, c_2, c_3 and the three stores are terminals in this instance of the ST problem. The intermediate points w, x, y and z correspond to the Steiner vertices. The hardness of the ST problem lies in finding the Steiner vertices that minimize the cost of the tree.

tree may span additional points in order to minimise the cost. The points that must be visited are called terminals, whereas the additional points are called Steiner points. Later, the ST problem was extended to various metric spaces. Among them, the rectilinear ST problem, i.e., the Steiner tree in the rectilinear plane, and the ST problem on graphs are considered the most important. In 1972, Karp [42] showed that the ST problem on graphs is NP-hard. Later, Garey and Johnson [31] showed that the rectilinear ST is also NP-hard while Garey et al. [30] showed that the Euclidean ST problem is NP-hard. In this thesis, we propose generalisations of the ST problem on graphs.

An instance of a trip-based ride-sharing problem with intermediate meeting points is an instance of the ST problem on graphs—see Figure 2.3. Formally, given an undirected weighted graph $G \coloneqq \langle V, E, c \rangle$, where V is the set of vertices, E is the set of edges, and $c : E \to \mathbb{R}_+$ is a cost function; and the set of terminals $D \subset V$, finding the ST T that spans D is defined as:

$$T \coloneqq \arg\min_{T' \in \mathbb{T}} c(T') \tag{2.1}$$

where $\mathbb{T}$ is the set of all possible Steiner Trees that span D. The size of $\mathbb{T}$ is exponential to the sizes of the sets V and D.

There are multiple exact [21, 27, 50] and approximate algorithms [36, 43, 54] for this problem. Yet, the seminal work of Dreyfus and Wagner [21] is the basis and baseline of our VST-based Ride-Sharing (VST-RS) algorithm[4].

Dreyfus and Wagner developed a fast dynamic programming solution which has been the best for the last four decades. At the beginning, their algorithm builds Steiner subtrees for every combination of two terminals and one Steiner vertex, and stores the best Steiner vertex with its corresponding combination. Then, the same process is performed for every combination of three terminals based on the results for two terminals and so on, until the number of terminals for each combination is $|D| - 1$.

The complexity of Dreyfus and Wagner's algorithm is $\mathcal{O}(3^{|D|}|V| + 2^{|D|}|V|^2)$. 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 vertices in the network [27, 50]. In our work, the number of terminals (users) is a small proportion of the number of vertices in the network. Hence, Dreyfus and Wagner's algorithm is still the best building block on which base our work.

Variable Steiner Tree (VST) Problem

A VST is the minimum-cost tree that must span a subset of vertices of a graph, called terminals, may span additional vertices, called Steiner vertices, and must span one extra vertex from a subset of vertices. Its definition is similar to the ST problem, yet there is an additional terminal, which "varies", i.e., it is not fixed, and it belongs to a subset of vertices. This additional terminal corresponds to a POI in our activity-based ride-sharing model. Since passengers are open to ride-share to any destination if they can perform their preferred activity at the computed destination, destinations are now a set to choose from. Formally, we are still given a subset of terminals $D \subset V$, however, another subset $P \subset V \setminus D$ is given. Finding a VST T is defined as in (2.1), but each Steiner Tree now spans $L \coloneqq D \cup \{p\}$ where $p \in P$—see Figure 2.4.

[4] Our VST-RS algorithm is explained in more detail later in this chapter.

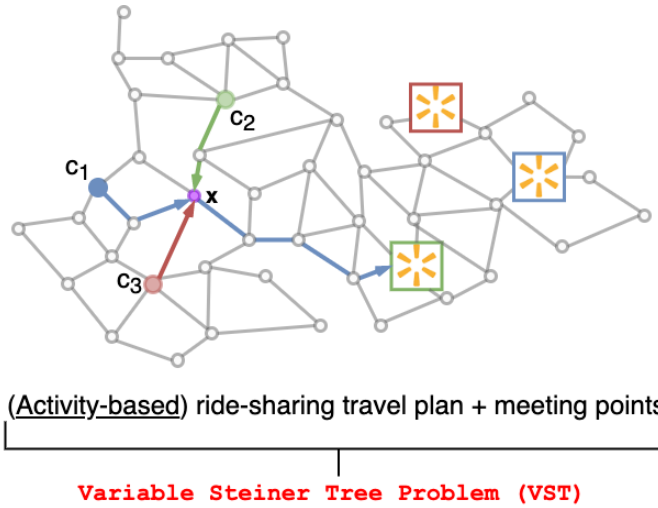

Figure 2.4: Users c_1, c_2, c_3 and the three Walmart stores are the terminals. Again, the users have agreed to meet at intermediate points. They also gave up their preferred destination and now all go to the same destination. There is only one Steiner vertex: x. The VST problem is not only about finding the Steiner vertices but also the store, from the subset of Walmart terminals, that minimizes the cost of the tree.

Similar to ST, VST is NP-hard as the latter is at least as hard as the former. We present our approximate approach VST-RS that computes optimal travel plans for activity-based ride-sharing requests. Computing a single travel plan in our activity-based ride-sharing model with intermediate meeting points corresponds to solving an instance of the VST problem. In VST-RS, an initial partition of the set of users is performed based on heuristics. Then, VST-RS computes the optimal travel plan for each subset, that is, we solve each instance of the VST problem exactly.

Figure 2.5 shows a comparison between ST and VST. Set $\{a, b, c, d, e, p_1, p_2\}$ is given as input to both problems. However, this input is broken down into $\{a, b, c, d, e\}$ and $P = \{p_1, p_2\}$ for VST. Bold edges are part of the ST (left) and VST (right). Both trees span Steiner vertices as these vertices enable further cost minimization. The VST spans p_2 as this POI yields the minimum cost. Computing the VST is at least as hard as computing the ST as it involves not only trying every combination of terminals and Steiner vertices, as it occurred with the ST, but also, for each of those combinations, trying with each $p \in P$ (or at least some of them if pruning techniques are applied over P).

In Chapter 3, we present a constrained version of the VST problem. We call this new

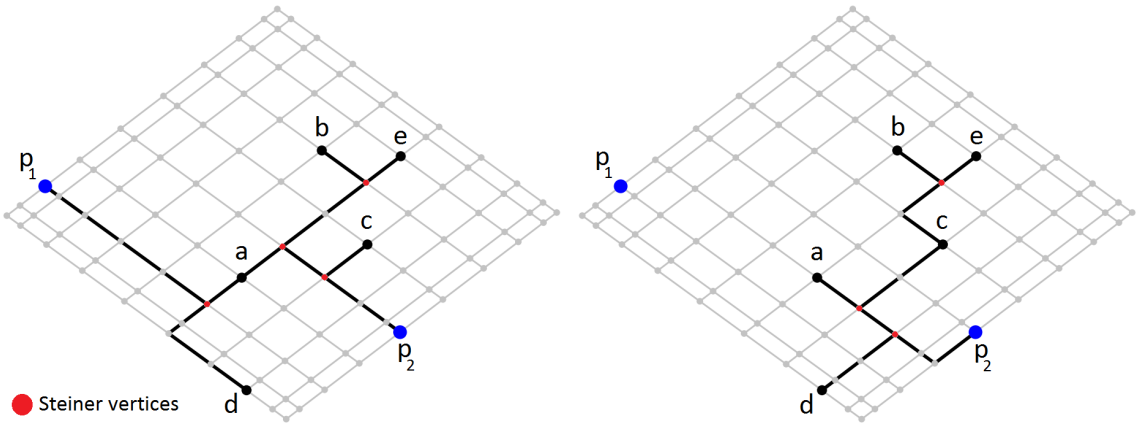

Figure 2.5: **Left.** ST spans terminals $\{a, b, c, d, e, p_1, p_2\}$. **Right.** VST spans terminals $\{a, b, c, d, e\}$ and the POI p_2 which, among $\{p_1, p_2\}$, is the one that minimizes the cost.

graph-theoretic problem, Constrained Variable Steiner Tree (C-VST). Indeed, we constrain the set of candidate Steiner vertices to be within the set of hot-spots $H \subset V$. This is because we realized the impracticality of most intermediate meeting points for parking. This led us to constrain the meeting points to suitable parking places only—*hot-spots*. Having hot-spots as intermediate meeting points is not only convenient in terms of parking but also helps alleviate other related problems ride-sharing presents. In particular, we show how they would solve the round-trip problem passengers face, which has to do with the difficulty of finding a round-trip ride-sharing arrangement.

In Chapter 4, we present our final activity-based ride-sharing algorithm that we call VST-based Congestion-Aware (VST-CA). This is a congestion-aware algorithm as we projected ourselves into the not-far-future when ride-sharing is the main cause of congestion. VST-CA is an approximate solution to a more general version of the Multicast Congestion (MC) problem. It is a meta-algorithm that iteratively call VST-RS. We describe the MC problem below.

Multicast Congestion (MC) Problem

Our meta-algorithm VST-CA presented in Chapter 4 is designed to alleviate congestion caused mainly by ride-sharing. VST-CA is a meta-algorithm that invokes iteratively a ride-sharing algorithm that compute travel plans. We invoke VST-RS as we opted for the

activity-based ride-sharing model. Computing a single travel plan in our activity-based model with intermediate meeting points is equivalent to solving the VST problem. There may be more than one group of users for each of whom a travel plan is needed. Thus, VST-RS finds the forest of VSTs of minimum cost. VST-CA also finds the forest of VSTs, yet the minimum cost is attained by minimizing the maximum number of VSTs an edge is part of.

There is a similar problem in telecommunication networks called the MC problem. In this problem, source nodes serve multicast requests on the same network. Every user within a multicast request must be connected to the source node corresponding to that request. There may be edges in the network that are used by more than one request. The goal is to reduce the level of congestion in the network, which means minimizing the maximum number of times edges are used.

Let us formally define the MC problem. Given an unweighted network $G \coloneqq \langle V, E \rangle$, and a set of multicast requests $S_1, \ldots, S_k \subseteq V$, the problem is to find a set of k STs $T_1, \ldots, T_k$ where each T_i connects the terminals—users and source—within request S_i. The goal is to minimize the maximum number of times an edge is used by the STs. The MC problem is a generalization of two well-known problems: the *routing problem*, which finds integral paths with minimum congestion between two terminals, and the ST problem. The main difference with our congestion problem lies in the type of trees we compute, which are VSTs instead of STs.

Vempala and Vöcking [69], Jansen and Zhang [39, 40], and Baltz and Srivastav [10] construct their approximate solutions to the MC problem based on linear program relaxations of their integer linear formulations. Jansen and Zhang, and Baltz and Srivastav work on the dual formulations of their primal relaxations, where the interpretation of the dual variables corresponds to the costs of the edges. That is, by solving the dual problem, the optimal costs of the edges guide the selection of the edges—primal variables are indicator vectors over the set of edges—that are part of the solution. VST-CA is inspired by this approach of optimizing the costs of the edges to divert vehicles from the busiest edges. In VST-CA, the cost of the solution is minimized as a consequence of minimizing the overlap. The cost is modeled as a positive, nonlinear, and strictly increasing function

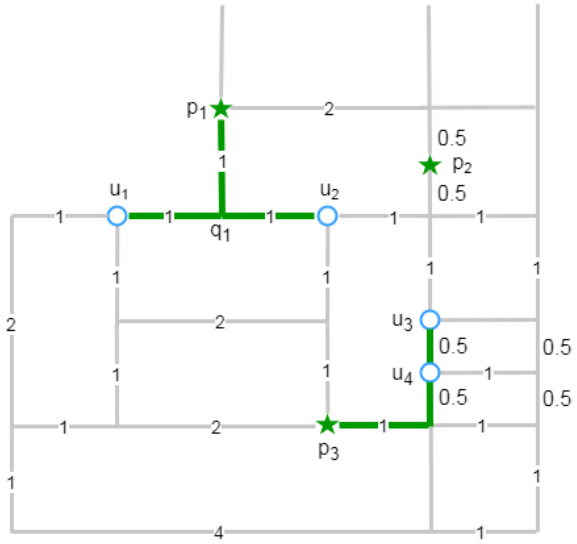

Figure 2.6: A simple road network. Users u_1, u_2, u_3 and u_4 can perform their common activity at any candidate POIs p_1, p_2 and p_3. Bold edges correspond to the optimal activity-based travel plans.

of traffic load.

The contributions presented in this thesis have our previous work [44] as a starting point. We present the most important insights of this preceding work in the next section.

2.1.5 Our Previous Work: Activity-Based Ride-Sharing + Meeting Points

In our previous work [44], we presented the VST-based Ride-Sharing (VST-RS) approach to compute multiple travel plans in our proposed activity-based ride-sharing model with intermediate meeting points. An example of this model is shown in Figure 2.6. Users u_1, u_2, u_3 and u_4 are interested in the same activity, which can be performed in any of the three POIs p_1, p_2 and p_3. In the optimal solution, u_1 and u_2 would meet at q_1, leave one of their vehicles there and share the other to go to p_1; u_3 would pick up u_4 and go to p_3. Bold edges represent the optimal travel plans. The total travel cost for all vehicles is $1 + 1 + 1 + 0.5 + 1.5 = 5$. There are many other suboptimal plans. If the users employ a trip-based ride-sharing service, they specify their destinations. If we assume that users specify their nearest POIs, then the total travel cost would be $2 + 1.5 + 1.5 + 1.5 = 6.5$.

A road network can be represented by the undirected graph $G \coloneqq \langle V, E, c \rangle$, where ver-

tices V are the road intersections, edges E are road segments between two intersections, and $c : E \to \mathbb{R}_+$ is a cost function. Let z be the same number of seats in each vehicle. Let $U \subset V$ and $P \subset V$ be the sets of users and POIs, respectively. We aim to find interconnecting routes—travel plans—from all members of U to one or more elements of P that minimizes the total cost of the edges on the routes.

Within a time interval, users U issue their requests where an individual request comprises pick-up location and activity. We assume users ride-share provided they are to perform the same activity only. We acknowledge that there exist POIs that enable to carry out different activities, e.g., malls that feature opportunities for shopping as well as eating fast food, or that are in close proximity to several destinations. Hence ride-sharing of people with different activities in mind could pay off after all. However, we decided to group individual requests per activity since, otherwise, we would have required to solve a more complex approach.

Finding the optimal travel plans involves dividing all users, who are interested in the same activity, into groups so that each group can fit in a vehicle—vehicles have limited capacity z. Then, the total travel cost is the sum of the cost of each group who meet at intermediate points and then go to the most convenient POI. This total cost should be minimum. Computing a single travel plan in this model is equivalent to solving an instance of our VST problem. Thus, VST-RS computes the near-optimal forest of VSTs.

For each activity i we get POIs $P_i \subseteq P$ where users $U_i \subseteq U$ can carry out activity i. Travel plans that corresponds to activity i will be computed for the subset of users and POIs $\langle U_i, P_i \rangle$. Therefore, we need to find out the optimal division of U_i into subsets $\{U_i^j\}$, and for each subset U_i^j find out a POI $p_i^j \in P_i$ such that the tree T_i^j that spans $U_i^j \cup \{p_i^j\}$ is a VST. The total cost of the road segments of T_i^j is given by $c(T_i^j) \coloneqq \sum_{e \in T_i^j} w_e$, where w_e corresponds to the cost of the road segment e that is part of the VST. Hence, a forest f_i for activity i can be defined as $f_i \coloneqq \bigcup_{j=1}^{t} T_i^j$, where t is the number of VSTs for this activity. The cost of the travel plans for activity i is thus given by $c(f_i) \coloneqq \sum_{j=1}^{t} c(T_i^j)$.

Formally, given an undirected weighted graph $G \coloneqq \langle V, E, c \rangle$, set of k activities requested within a time interval, set of users U and POIs P grouped into $\{\langle U_i, P_i \rangle\}_{i=1}^{k}$ where U_i and P_i correspond to activity i. The problem consists in finding a set of forests F de-

fined as:

$$F \coloneqq \arg\min_{F' \in \mathbb{F}} \sum_{i:1<i\leq k \;\&\; f_i \in F'} c(f_i) \tag{2.2}$$

where $\mathbb{F}$ is the set of all possible sets of forests. Here the size of $\mathbb{F}$ may be large. A brute force approach to explore $\mathbb{F}$ would be prohibitive. Our approximate approach VST-RS computes each f_i independently.

VST-RS 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[5] 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—POI.

Secondly, 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 closest POI. Let $dist : V \times V \to \mathbb{R}_+$ be the shortest distance function between two vertices, and $M : V \to V$ be an auxiliary function that returns the medoid of the Voronoi cell where a vertex is in. Thus, these constraints are reflected in the following results:

Theorem 2.1. *In an optimal solution, a terminal u_j meet another terminal at a Steiner vertex m_{ij} only if $dist(u_j, m_{ij}) < dist(u_j, M(u_j))$ [44]—see Figure 2.7 (left, center).*

Corollary 2.1. *In an optimal solution, a set of terminals U', that have already met at Steiner vertex m_1 meet other users at Steiner vertex m only if $c(T') < c(T'')$, where T' spans $U' \cup \{m_1, m\}$ and T'' is the VST that spans $U' \cup \{m_1, M(m_1)\}$ [44]—see Figure 2.7 (right).*

[5] In a Voronoi cell, the distance of a vertex, which belongs to this cell, to the medoid of this cell is shorter than its distance to any of the medoids of the other Voronoi cells.

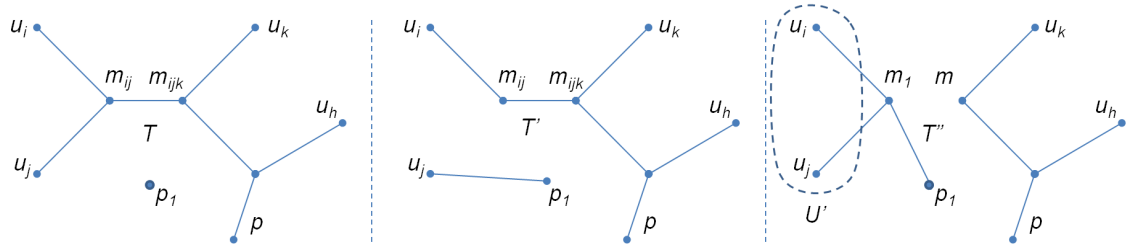

Figure 2.7: **Left.** An optimal travel plan. **Center.** A suboptimal solution when u_j travels directly to her closest POI instead of meeting u_i at m_{ij}. **Right.** Suboptimal solution when set of users U' do not meet other users at m after having met at m_1. They again go directly to their closest POI p_1.

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 ($z < S$). The division of users into groups of at most S users is the first stage of VST-RS—line 3, Algorithm 1.

In its second stage—lines 4-14, Algorithm 1—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 and Wagner's algorithm [21] which computes the exact solution of the ST problem. In VST-RS, a terminal (resp. set of terminals) chooses to connect to a Steiner vertex as long as Theorem 2.1 (resp. Corollary 2.1) holds, otherwise it connects to its closest POI, which may result in choosing more than one POI per subgroup of z users.

Let the powerset $\wp$ of users and POIs be the collection of subsets of an instance of the Set Cover problem. Let the cost of the VST that spans each subset be the cost of such subset in $\wp$. 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 VST problem.

The worst-case time complexities of the first and second stages are $\mathcal{O}(|U_i||V|^2)$ and $\mathcal{O}\left(\frac{|U_i|}{S}(3^S N + S^z|V|^2)\right)$, respectively, with the second one being dominant. The time complexity of Dreyfus and Wagner's algorithm [21] is $\mathcal{O}(3^{|U_i|}|V| + 2^{|U_i|}|V|^2)$. In our algorithm, if $S = |U_i|$, i.e., there is just one group which contains all users within activity i, and z is big enough, its time complexity is comparable to Dreyfus and Wagner's, i.e., it leads

Algorithm 1 Compute the overall optimal travel plan for one activity [44].

```
1: function VST-RS(U, P, G)                          ▷ U : users, P : POIs, G : graph.
2:     plan ← ∅
3:     groups ← COMPUTEGROUPS(U)                                  ▷ First Stage.
4:     for each group ∈ groups do                                 ▷ Second Stage.
5:         OptCost, OptPOI, J, bestDiv, bestSubgrouping ←
   COMPUTECOSTEACHCOMBINATION(group, P, G)
6:         subgroups ← ∅
7:         if number of users in group > z then
8:             OptCost, bestSubgrouping ← COMPUTEOPTIMALDIVISION(group, OptCost)
9:             subgroups ← get subgroups from group and bestSubgrouping recursively
10:        else
11:            subgroups ← subgroups ∪ group
12:        for each subgroup ∈ subgroups do
13:            component ← Recover plan for subgroup using
   OptCost, OptPOI, J, bestDiv, bestSubgrouping
14:            plan ← plan ∪ {component}
15:    return plan
16: end function
```

to combinatorial explosion. However, we show experimentally that the cost of the travel plan is stable when the group size S is larger or equal to 8 and the capacity of a vehicle z is 4. Thus, $S = |U_i|$ does not hold when $|U_i|$ is big enough. Also, the inclusion of the constraints in our algorithm, embodied by Theorem 2.1 and Corollary 2.1, yields to the term $S^z|V|^2$ instead of $2^{|U_i|}|V|^2$ as in Dreyfus and Wagner's. Consequently, in practice, our algorithm's performance is much better than the solutions based on Dreyfus and Wagner.

The space complexity of our approach is dominated by the second stage as well, and it is $\mathcal{O}(|U_i|^z|V|)$.

2.2 Crowdsourced Delivery

Crowdsourced delivery is the goods-based version of ride-sharing. Instead of transporting people, private drivers pick up products from retailers' stores and deliver them to retailers' online customers. Both retailers and drivers benefit because retailers separate themselves from the complexities of delivery logistics and drivers earn extra money. Walmart tested a similar model in 2013 by asking its in-store customers to deliver parcels to

its online customers. In the same year in Stockholm, DHL launched a platform called MyWays which allowed registered drivers to deliver parcels along their daily routes in exchange for a small fee [19]. Nowadays, Walmart offers a service called Spark Delivery in some US cities [71], which is similar to MyWays, except that drivers do not provide their routes.

In current crowdsourced delivery models, retailers' platforms choose the pick-up store for each order at a time when drivers' routes are unknown or not provided [72]. Then, when orders come, a dispatcher automatically assigns them to private drivers based on custom business rules, e.g., an order is assigned to a driver who is within a certain distance from the already chosen pick-up store [11]. The pick-up store for each order is fixed at the time when delivery routes are computed. This leads to suboptimal delivery routes since it dismisses other pick-up stores that may be closer to the original driver's route. In Chapter 5, we propose *a new crowdsourced delivery model that generalizes the current crowdsourced delivery model as activity-based ride-sharing does with trip-based ride-sharing.* In our proposed model, we leave the pick-up location (resp. the destination in activity-based ride-sharing) open provided that the online customer gets the same products. That is, ad hoc drivers have a set of possible pick-up locations for each order instead of a fixed one.

In our proposed model, drivers register their original routes in the retailer's platform. Then, when orders come, the pick-up store for each order is the store that optimizes the delivery route. Both pick-up store and delivery route are computed at the same time based on drivers' original routes. The solution space when computing the optimal delivery route is expanded because the pick-up store can be any store of the retailer. An optimal solution to our new model is guaranteed to be cheaper than an optimal solution to the current model. This is by definition since the current model's solution is part of our larger space.

Although many of the existent studies in crowdsourced delivery [8,16,17,41] do consider the routes of the private drivers, *none of them computes the delivery routes and chooses the pick-up stores as part of the same optimization process*. From this standpoint, our new model is at an advantaged position as it offers guaranteed cheaper delivery routes. These

Table 2.2: Crowdsourced delivery approaches

	Reference	**Description**
Concurrent optimization of routes and stores	*Our model in Chapter 5*	Alternative pick-up stores
Routes and stores are independently optimized	Arslan et al. [8]	Hybrid PDP-like: Private drivers + dedicated drivers
	Chen and Chankov [16]	PDP-like
	Chen et al. [17]	Transfer parcels between drivers
	Kafle et al. [41]	Partial outsourcing: Trucks still do last mile and crowds do last leg.

studies are summarized in Table 2.2.

Current crowdsourced delivery models are similar to the Pickup and Delivery Problem (PDP). In PDP, a fleet of vehicles also serve transport requests, and each request consists of one pick-up and one delivery location. However, Arslan et al. [8] highlight that, in crowdsourced delivery, the fleet of vehicles is not dedicated but optional to independent private drivers. In crowdsourced delivery, private drivers' journeys already take place, and their start and end locations are different from a depot. Computing an optimal delivery route for each driver in current models is equivalent to solving the Hamiltonian Path with Precedence Constraints Problem (HPPCP).

In our model, computing an optimal delivery route for each driver is equivalent to solving a new Hamiltonian path problem. We call it *Group Hamiltonian Path with Precedence Constraints Problem (GHPPCP)*. This new graph problem is different from the HPPCP. In HPPCP, the delivery route for a driver goes through all her customers and the stores already chosen by the retailer, whereas in GHPPCP, a delivery route goes through all customers, yet there is a "group" of stores to choose from.

The PDP is generalized by the General Pickup and Delivery Problem (GPDP) introduced by Savelsbergh and Sol [57]. In GPDP, each transport request can be picked up and delivered from and to multiple locations. GPDP generalizes not only PDP but also other

Table 2.3: GPDP specializations and our model. Note that the characterizations are at *customer* level.

Problem	Pickup	Delivery	Qty.	Description
GPDP	$\|S\| \geq 1, \|V_k \cap S\| \geq 0$ A driver k can pick up from different stores.	$\|C\| \geq 1, \|V_k \cap C\| \geq 0$ A driver k can deliver to different locations.	≥ 1	Multi-origin, multi-destination multi-products.
PDP	$\|S\| = 1, \|V_k \cap S\| \leq 1$ A driver k picks up from at most one store.	$\|C\| = 1, \|V_k \cap C\| \leq 1$ A driver k delivers to at most one location.	≥ 1	Door-to-door multi-product requests.
DARP	$\|S\| = 1, \|V_k \cap S\| \leq 1$ A driver k picks up a person.	$\|C\| = 1, \|V_k \cap C\| \leq 1$ A driver k drops off a person.	$= 1$	Door-to-door one-person requests.
VRP	$k^+ = k^- = k$ AND $\big(\|S\| = 1, k = s \in S$ OR $\|C\| = 1, k = q \in C\big)$		≥ 1	Either pickup or drop-off locations are at the depot.
MPDP	$\|S\| \geq 1, \|V_k \cap S\| \leq \|S\|$ A driver k can pick up from different stores.	$\|C\| = 1, \|V_k \cap C\| \leq 1$ A driver k delivers to at most one location.	≥ 1	Multi-origin, one destination multi-products.
Our model (Chapter 5)	$\|S\| \geq 1, \|V_k \cap S\| \leq 1$ **Although there are more than one store, a driver k picks up from at most one.**	$\|C\| = 1, \|V_k \cap C\| \leq 1$ A driver k delivers to at most one location.	≥ 1	Alternative pick-up locations.

transportation problems such as Dial-A-Ride Problem (DARP), Vehicle Routing Problem (VRP) and Multi-Pickup and Delivery Problem (MPDP). Even though our new model has some resemblance to PDP, it is *not* under the umbrella of GPDP. *In our model, each transportation request has more than one possible pick-up location and only one of them must be visited*. Table 2.3 show characterizations of GPDP and its specializations. Note how our model's `Pickup` definition is not captured by the GPDP definition.

Our model also puts a clear differentiation between crowdsourced delivery and ride-sharing. Before our model, it was possible to utilize ride-sharing approaches to solve the crowdsourced delivery problem. Indeed, Tong et al. [66] proposed a unified approach to routing in sharing-economy applications in general, i.e., either ride-sharing, food delivery or crowdsourced delivery. However, we cannot use unified approaches such as this or any other ride-sharing algorithm in our model since people's pick-up locations are fixed and not computed on routing time.

2.2.1 Crowdsourced Delivery and Graph Theory

In this section, we describe two problems related to our model. The first is the PDP, which is an extensively studied problem in transportation science. Secondly, the route that a single driver follows to serve the customers within an assignment, in the crowdsourced version of PDP, can be reduced to the HPPCP.

Pickup and Delivery Problem (PDP)

In the PDP, a fleet of drivers start at and return to a central depot. They serve transportation requests and each request specifies a single pick-up location and a single delivery location. The goal is to find a set of routes for the drivers such that all customers are served with the minimum cost for the delivery company.

Formally, in PDP, we are given the depot location n_0 and sets C and S of online customers' and stores' locations, respectively. Online customers submit their requests to retailers. Then, we can build a directed weighted graph $G \coloneqq \langle V, A \rangle$, where $V = \{n_0\} \cup C \cup S$; and A is the set of arcs that satisfy a priori constraints of the problem, e.g., drivers cannot go to any online customer when they start their journey as they must visit a store first. For each arc $(i, j) \in A$, we are given the cost $c_{i,j}$ and time $t_{i,j}$ to travel from i to j. Therefore, the problem consists in finding the set of delivery routes P defined as:

$$P \coloneqq \arg\min_{P' \in \mathbb{P}} \sum_{P_k \in P'} \sum_{(i,j) \in A_k} c_{i,j} \tag{2.3}$$

where $\mathbb{P}$ is the set of all possible sets of routes. Each P_k is associated with a driver k and is defined as $P_k \coloneqq \langle V_k, A_k \rangle$, where $V_k \subseteq V$ and $A_k \subseteq A$.

The cardinality of $\mathbb{P}$ may be large, and a brute force approach would be prohibitive. The PDP is NP-hard. Moreover, for each P_k, the following must be satisfied:

1. P_k starts at n_0.

2. $|V_k \cap S| \leq 1$, for each $q \in C$. That is, products for a customer are picked up from only one store, yet a driver can visit more than one store since she may serve more customers. There may be drivers who are assigned no customers whatsoever.

3. Let s_q be the pick-up store from for the online customer q. Let B_i^k be the time at which driver k starts service at location i. Thus, $B_q^k \geq B_{s_q}^k + t_{s_q,q}$, that is, the store must be visited before the customer—*precedence constraint*.

4. P_k ends at n_0.

Computing the route for a single driver in the crowdsourced version of PDP—where the concept of depot is void—is equivalent to solving an instance of the minimum-cost HPPCP. Following, we describe HPPCP in detail.

Hamiltonian Path with Precedence Constraints Problem (HPPCP)

The HPPCP is a variant of the well-known Hamiltonian path problem. In this classical graph problem, a source and destination vertices are given, and the goal is to find a path between both vertices that visits all the other vertices in the graph only once. The Hamiltonian cycle is a similar problem where the source and destination vertices are the same. There may be more than one Hamiltonian path/cycle in a graph. Finding whether a graph contains at least one is an NP-complete problem [42]. Finding the minimum-cost Hamiltonian cycle is also known as the Traveling Salesman Problem (TSP), which is NP-hard.

In the HPPCP, besides the source and destination vertices, sets of direct and immediate precedence constraints are given. These constraints require that certain vertices must precede certain other vertices in any feasible Hamiltonian path—see Figure 2.8. The HPPCP is also known as Sequential Ordering Problem (SOP). It was introduced by Escudero [23] and further studied in [9, 24, 29, 59].

Formally, let $G \coloneqq \langle V, A \rangle$ be a directed weighted graph with vertex set V and arc set A. For each arc $(i, j) \in A$, $c_{i,j}$ corresponds to its cost. Let $P \coloneqq \langle V, R \rangle$ be a directed acyclic graph that encodes the *direct* precedence relationships built upon the same set of vertices V. An arc $(i, j) \in R$ means that vertex i should precede j, represented as $i \prec j$, in a feasible path. Therefore P must be acyclic. Let s and t be the source and destination vertices, respectively. Thus, $\{(s, j) \,|\, j \in V \setminus \{s\}\} \subset R$ and $\{(i, t) \,|\, i \in V \setminus \{t\}\} \subset R$. Therefore, the problem consists in finding a feasible Hamiltonian path P_k defined as:

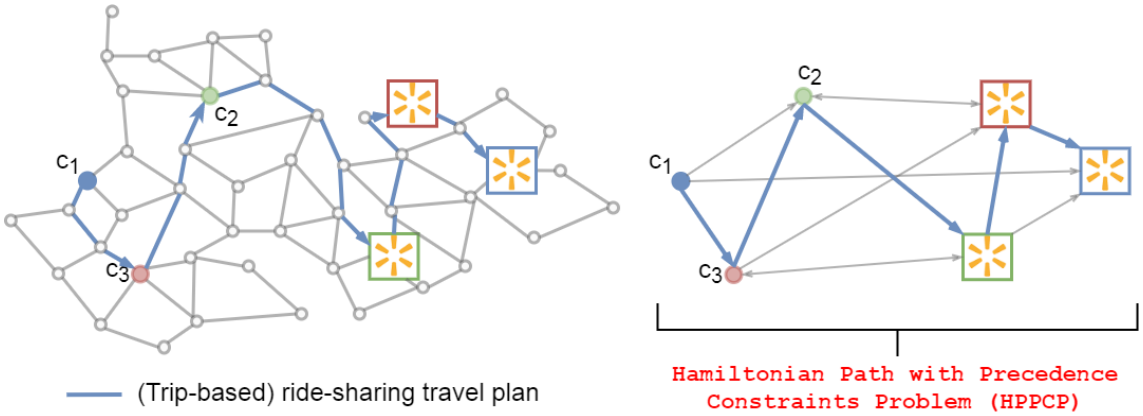

Figure 2.8: Three participants c_1, c_2 and c_3 want to ride-share. User c_1 is the driver who picks up and drops off the other two users from their source and to their destination locations—trip-based ride-sharing. The *direct* precedence constraints are implicitly given as the passengers must be visited before their destinations. **Left.** The whole road network and the optimal travel plan in bold edges are depicted. **Right.** A directed subgraph is constructed. Its vertex set is $\{c_1, c_2, c_3, s_1, s_2, s_3\}$, where s_i is the preferred Walmart store for participant c_i. Its arc set satisfies a priori constraints of the problem, e.g., a driver cannot go to a passenger's destination before visiting the passenger. This subgraph is part of the input for the HPPCP. The driver's source and preferred store are the source and destination vertices given as input, as well. The optimal travel plan in bold edges is also depicted in this subgraph.

$$P_k \coloneqq \underset{P'_k \in \mathbb{H}}{\arg\min} \sum_{(i,j) \in A'_k} c_{i,j} \tag{2.4}$$

where $\mathbb{H}$ is the set of all feasible Hamiltonian paths on G, and A'_k is the set of arcs that conform a certain $P'_k \in \mathbb{H}$. Note how this definition corresponds to a single-vehicle PDP version of (2.3), yet each path now must satisfy precedence constraints as well. Of course, $\mathbb{H}$'s cardinality is huge.

If transitivity is allowed, i.e., if $i \prec j \wedge j \prec k \implies i \prec k$, there exists more than one level of precedence. Furthermore, $i \prec j \wedge k \prec j$ is also possible, i.e., two or more vertices precede one. This definition models several real-life problems, especially within the manufacturing realm, e.g., job scheduling with one machine. Yet, SOP cannot enforce $i \prec j \vee k \prec j$ as our model does, e.g., a Walmart store must be visited before a customer but visiting any Walmart store is enough.

In SOP, *immediate* precedence relationships may also be defined. For this effect, besides having P, a vector I is needed. The n-th element I_n indicates the vertex that must precede vertex n immediately. That is, a path other than $(\dots, I_n, n, \dots)$ is not feasible. We do not include immediate precedence constraints in our problem definition as SOP does. We are keen to explore it as real delivery requirements must be modeled in these terms.

SOP can also be reduced to *single-vehicle* GPDP. Note that many-to-many constraints in SOP would represent multiple pickup to multiple delivery in GPDP. Yet, immediate constraints and transitivity are still missing in GPDP.

Chapter 3

Activity-Based Ride-Sharing with Hot-Spots

In Chapter 2, we presented our previous work on activity-based ride-sharing with intermediate meeting points. In this preceding work we assumed that users can meet at any intersection in the road network. In this chapter, we drop such an assumption. We propose an enhanced ride-sharing model where meeting points are suitable points only. We call them hot-spots. *Hot-spots are shown to increase 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 (C-VST), 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.*

3.1 Overview

THIS chapter proposes to enhance ride-sharing and make it more convenient, attractive and safer by the inclusion of special meeting points that we call *hot-spots*. We envision hot-spots as places where users find it easier to leave their vehicles and take someone else's vehicle 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 [20].

We propose that users drive to hot-spots and leave their vehicles there and then continue their trips sharing others' vehicles. Some people may argue downsides in this proposed model. For example, users who own a vehicle and have it ready to use would not be willing to park somewhere in order to get in the vehicle of someone else, especially

for rather local bound activities as shopping. Also, parking costs may have an impact.

Although owning a vehicle may be perceived as advantageous, there are situations where own vehicles would rather be used exceptionally. For instance, attending to public events or even going shopping to crowded malls. It seems to be less stressful and cheaper to park at a hot-spot (considering parking cost, if any as it may be subsidised) and ride-share from it. Also, our hot-spots model is a ride-sharing version of the *park-and-ride* initiatives that run in many cities around the world, with the difference that our model includes private drivers instead of public transport.

We compare our hot-spots model with two models. *Model A*, where users meet at other places instead of hot-spots. In this model, finding a parking spot is perhaps difficult. *Model B*, where users are picked up from their initial locations, e.g., their home address, and thus, they reveal their locations to strangers and cause larger detours to drivers.

We argue that *hot-spots are also important because round-trip ride-sharing arrangements are significantly more likely than in the aforementioned ride-sharing models, i.e., models A and B, where hot-spots are not present.* 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 makes ride-sharing convenient for passengers since round-trips are more likely, and they can leave their vehicles in suitable places for parking. It is convenient for drivers too, since they do not need to make large detours. Hot-spots make ride-sharing attractive because they gather more people and thus, drivers will find more people who share costs with. Ride-sharing becomes trustworthy because privacy is achieved as users do not need to provide precise home/work locations. Moreover, safety is increased through video surveillance. Finally, our scheme with hot-spots keeps vehicle occupancy rate comparable to model A, which in turn has been proved to increase the occupancy rate compared with model B [2, 32, 60].

Our proposed model can be built on top of either trip-based or activity-based ride-sharing. Since activity-based ride-sharing increases the matching rate [44,73], we decided

to build our model based on this paradigm. 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.

VST-based Ride-Sharing (VST-RS), the state-of-the-art algorithm in activity-based ride-sharing, was proposed in our previous work [44] to solve the Variable Steiner Tree (VST) problem. We observed that if the road network is modeled as a graph and the users, Points of Interest (POIs) and meeting points are a subset of the vertices in the graph, the shared route for each subset of users would be a VST. In each VST, the leaves are the users—terminals—the intermediate vertices are the meeting points—Steiner vertices—and the root is the POI that minimizes the cost of the tree. In this chapter, we include a new constraint on the feasible set of Steiner vertices. Meeting points must be hot-spots only. We redefine the VST problem as the Constrained Variable Steiner Tree (C-VST) problem and we show that even with such a constraint this new problem is NP-hard.

We introduce an approximate solution to the C-VST problem that maximizes 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 domain of this function is the powerset of the vertices that may connect to a Steiner vertex, combinatorial explosion occurs when it is maximized naively. 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 near-optimal solution to the C-VST problem that can be deployed as a real-time service.

We show 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, while at the same time, our proposed ride-sharing model makes round-trips significantly more likely and keeps vehicle occupancy rates comparable to model A.

Our contributions in this chapter 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 (C-VST) and prove it is NP-hard.
- We present a fast approximate algorithm for the C-VST problem that allows the development of a readily deployable solution for activity-based ride-sharing with hot-spots. A monotonic nondecreasing function that constructs near-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 in our previous work [44].

3.2 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 and they do not ride-share. 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 $G \coloneqq \langle V, A, c \rangle$ 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 $c : A \to \mathbb{R}_+$ is defined as $c(A) \coloneqq \sum_{a \in A} d_a$, where d_a is the distance 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 $G \coloneqq \langle V, A, c \rangle$, the set of user locations U, the set of POI locations P and the set of hot-spots H.

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

The C-VST problem is a generalization of the classical MDSF problem. We prove that C-VST is NP-hard even with the additional constraint on the feasible set of Steiner vertices. 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 3.1. *The kC-VST problem is NP-complete.*

Proof. To prove that the kC-VST problem is NP-complete we must prove that it is in NP and there exists a polynomial time reduction from an NP-complete problem. Since the decision version of the MDSF problem, that is, the Directed Steiner Forest (DSF) problem 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 $G \coloneqq \langle V, A \rangle$. In this reduction, we construct a directed graph $G' \coloneqq \langle V', A' \rangle$, 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. All pairs of hot-spots are connected with arcs in both directions. User vertices only get directed arcs towards hot-spots and POIs. POIs only get incoming arcs from hot-spots and users. 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 that altogether form a forest F in G. F will have Steiner vertices that belong to H only because F' has Steiner vertices that belong to

H only. This is enforced by the direction of the arcs in A'. Any path from a user to a POI can only be direct or via a hot-spot vertex as other users are excluded. $c(F) = c(F') \leq k$ because the cost of each arc of F' is the shortest distance 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'. Again, this is due to the way A' was built. These arcs will form a forest F' in G' whose cost $c(F') = c(F) <= k$. Hence, F' is a DSF in G'.

Therefore, the kC-VST problem is NP-complete. □

Corollary 3.1. *The C-VST problem 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 the C-VST problem is not in NP. Hence, the C-VST problem is NP-hard. □

3.3 Our Solution

This section presents our solution to the C-VST problem. The baseline is a modified version of the state-of-the-art algorithm VST-RS presented in our previous work [44]. We modified this state-of-the-art algorithm to consider the constraint on the feasible set of Steiner vertices and make it comparable to our solution.

Our approximate solution maximizes 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 domain of this function is the powerset of the vertices that may connect to a Steiner vertex, combinatorial explosion occurs when it is maximized naively. 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 near-optimal solution to the C-VST problem that is readily deployable.

3.3.1 Algorithm

Our main algorithm is Algorithm 2. It computes and builds the approximate solution for the C-VST problem. The solution is built iteratively in a bottom-up manner. Layers of depth-1 trees, that we call Subtree-SVs—see Definition 3.2—are piled to create VSTs. The first-level Subtree-SVs, 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 will have Steiner vertices as leaves. We call these leaves pseudo-terminals—see Definition 3.1.

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 3.5—can be seen as the proportion of cost savings that a set of terminals (resp. 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 (resp. pseudo-terminals)—of a particular candidate. This is presented in Algorithm 3 and specifically in Algorithm 4.

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 3 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.

Before describing our algorithms in detail, we define new functions, concepts and present more results. Recall that a road network is modeled with a graph $G \coloneqq \langle V, A, c\rangle$ where $U \subset V$, $P \subset V$ and $H \subset V$ correspond to the set of user, POI and hot-spot locations, respectively. If we use MDSF's terminology, then the users would be terminals, the POIs would be roots and the hot-spots would be Steiner vertices. Thus, we use these terms interchangeably. Moreover, the shortest distance between two vertices in G is the function $dist : V \times V \to \mathbb{R}_+$, and we define an auxiliary function $M(v)$ that returns the medoid of the Voronoi cell where v is in.

Definition 3.1. Pseudo-terminal *is a Steiner vertex that has been greedily chosen as part of the*

solution. After a Steiner vertex is chosen, it behaves like a terminal.

Definition 3.2. Subtree-SV *is a tree of depth 1 where the root*[1] *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 3.3. Independent cost *is a function $Ic : (U \cup H) \to \mathbb{R}_+$ defined for a terminal and a pseudo-terminal as following:*

$$Ic(t) \coloneqq \begin{cases} dist(t, M(t)) & t \text{ is a terminal} \\ \sum_{w \in L} Ic(w) & t \text{ is a pseudo-terminal} \end{cases} \tag{3.1}$$

where $L \coloneqq l(Sv(t))$.

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

Definition 3.4. Cumulative loss *is a function $Cl : (U \cup H) \to \mathbb{R}_+$ defined for a terminal and a pseudo-terminal as following:*

$$Cl(t) \coloneqq \begin{cases} 0 & t \text{ is a terminal} \\ \sum_{w \in L} Cl(w) + dist(w, t) & t \text{ is a pseudo-terminal} \end{cases} \tag{3.2}$$

where $L \coloneqq l(Sv(t))$.

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, i.e., no vertex connect to it.

[1] Readers should not confuse the root of a Subtree-SV with the root of a Steiner tree in the solution.

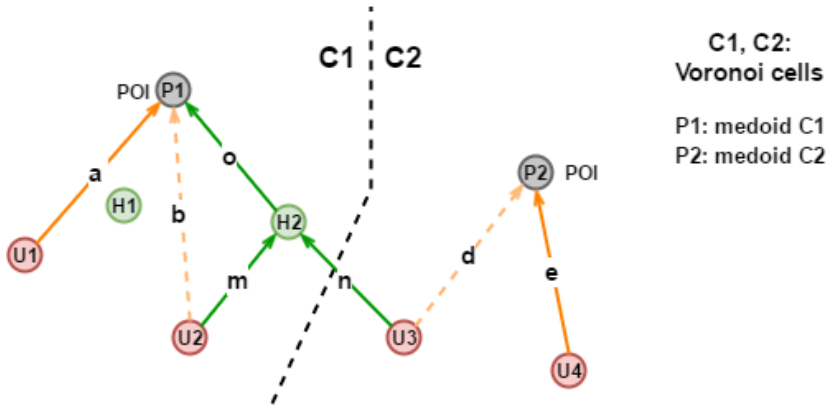

Figure 3.1: Let $U2$ and $U3$ be two terminals and $H2$ a hot-spot. $H2$ is the root of a Subtree-SV whose leaves are $U2$ and $U3$. $P1$ is the closest POI to $H2$ and $U2$, whereas $P2$ is so to $U3$. Note the Voronoi cells $C1$ and $C2$ where the medoids are $P1$ and $P2$, respectively. Thus, $Gr(\{U2, U3\}, H2) = \frac{b+d}{o+m+n}$, $Lr(U2, H2) = \frac{m}{b}$ and $Lr(U3, H2) = \frac{n}{d}$.

Definition 3.5. Gain ratio *is a set function* $Gr : 2^L \times H \to \mathbb{R}_+$ *defined for the set of leaves* L *of a Subtree-SV, whose root is a candidate Steiner vertex, as following:*

$$Gr(L,h) \coloneqq \frac{\sum_{w \in L} Ic(w)}{dist(h, M(h)) + \sum_{w \in L} Cl(w) + dist(w,h)} \tag{3.3}$$

where h *is a candidate Steiner vertex.*

Gain ratio of the set of leaves L *of a candidate Steiner vertex* h *can be seen as the proportion of cost savings that a set of terminals (resp. 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 3.1.*

Definition 3.6. Loss ratio *is a function* $Lr : (U \cup H) \times H \to \mathbb{R}_+$ *defined for a leaf of a Subtree-SV, whose root is a candidate Steiner vertex, as following:*

$$Lr(t,h) \coloneqq \begin{cases} 0 & t = h \\ \frac{Cl(t)+dist(t,h)}{Ic(t)} & t \neq h \end{cases} \tag{3.4}$$

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 (3.4) discerns whether* t *and* h *are the same or not.*

Loss ratio $Lr(t,h)$ measures the proportion of cost increment of a terminal (resp. 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 3.1.*

Definition 3.7. Well-formed Subtree-SV *of a candidate Steiner vertex h is a Subtree-SV where its leaves*

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

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

Definition 3.8. Committed terminal (resp. pseudo-terminal)
Let t be a committed terminal or pseudo-terminal then, $\forall h \in H : dist(t, M(t)) < dist(t,h)$.

Since our goal is to maximize Gr, we present an update rule, in Theorem 3.2, to attain a stationary point.

Theorem 3.2. *The set function $Gr(L,h)$ for a particular candidate Steiner vertex h is nondecreasing under the update rule*

$$L_{k+1} \leftarrow L_k \setminus \{t\} \text{ only if } \frac{1}{Gr(L_k,h)} < Lr(t,h) \tag{3.5}$$

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

Proof. At iteration k:

$$Gr(L_k,h) = \frac{Ic(t) + \sum_{w \in W} Ic(w)}{dist(h, M(h)) + Cl(t) + dist(t,h) + \sum_{w \in W} Cl(w) + dist(w,h)} \tag{3.6}$$

where $W \coloneqq L_k \setminus \{t\}$.

And at iteration $k+1$:

$$Gr(L_{k+1}, h) = \frac{\sum_{w \in W} Ic(w)}{dist(h, M(h)) + \sum_{w \in W} Cl(w) + dist(w, h)} \tag{3.7}$$

where $W \coloneqq L_k \setminus \{t\}$.

Let $A \coloneqq \sum_{w \in W} Ic(w)$, $B \coloneqq \sum_{w \in W} Cl(w) + dist(w, h)$, $a \coloneqq Ic(t)$, $b \coloneqq Cl(t) + dist(t, h)$, $s \coloneqq dist(h, M(h))$. Thus, equations 3.6 and 3.7 become:

$$Gr(L_k, h) = \frac{a + A}{s + b + B} \qquad Gr(L_{k+1}, h) = \frac{A}{s + B} \tag{3.8}$$

Then, from equations in 3.8:

$$Gr(L_{k+1}, h) = Gr(L_k, h) + Gr(L_k, h)\frac{b}{s + B} - \frac{a}{s + B} \tag{3.9}$$

From equation 3.9 we observe that $Gr(L_{k+1}, h) > Gr(L_k, h)$—nondecreasing—only if:

$$Gr(L_k, h)\frac{b}{s + B} > \frac{a}{s + B} \tag{3.10}$$

$$\frac{1}{Gr(L_k, h)} < \frac{b}{a} \tag{3.11}$$

$$\frac{1}{Gr(L_k, h)} < Lr(t, h) \tag{3.12}$$

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

In the following, we describe each of the algorithms.

Computing and Building an Approximate Solution to the C-VST Problem (Main Algorithm)

Committed terminals—see Definition 3.8—may exist within the set of terminals T given as parameter. Committed terminals have no option but to *connect* directly with their closest POIs as these POIs are closer to these terminals than any candidate Steiner vertex.

Algorithm 2 Compute and build approximate C-VST.

1: **procedure** C-VST(T, H) ▷ T : set of terminals, H : set of hot-spots
2: $N \leftarrow \{n \mid n \in T \wedge n$ is a non-connected terminal$\}$
3: $S \leftarrow \{s \mid s \coloneqq Sv(h) \wedge s$: well-formed Subtree-SV $\wedge\ h \in H\}$
4: **while** $N \neq \varnothing$ **and** $S \neq \varnothing$ **do**
5: $R \leftarrow$ CHOOSESUBTREESVS(S)
6: $T \leftarrow T \setminus \{l(s') \mid s' \in R\}$
7: $T \leftarrow T \cup \{r(s') \mid s' \in R\}$
8: $N \leftarrow \{n \mid n \in T \wedge n$ is a non-connected terminal$\}$
9: $S \leftarrow \{s \mid s \coloneqq Sv(h) \wedge s$: well-formed Subtree-SV $\wedge\ h \in H\}$
10: **end procedure**

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 2.1. Therefore, such committed terminals are not included in the set N, in line 2.

Line 3 initializes the set of well-formed Subtree-SVs—see Definition 3.7. 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 be within the set of hot-spots. Subtrees in S will likely share their leaves. Later, in Algorithm 3, 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 that will be part of the solution—see Algorithm 3. 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 the set N is updated accordingly in line 8. Committed pseudo-terminals will connect directly with their 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 2.1.

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.

Choosing *Subtree-SVs* Greedily

Algorithm 3 Greedily choose Subtree-SVs.

```
1:  function 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 : well-formed}
11:     return R
12: end function
```

A set of well-formed Subtree-SVs is computed by Algorithm 2 and then it is passed as parameter S to Algorithm 3. In turn, Algorithm 3 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. It 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 4. Theorem 3.2 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 that 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 to become 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.

Maximizing Gr by Pruning a *Subtree-SV*

Algorithm 4 Maximize Gain Ratio.

```
1:  function 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 function
```

Algorithm 4 prunes a Subtree-SV s' given as a parameter through the application of the update rule of Theorem 3.2. 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.

Because of Theorem 3.2, 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 of Gr—whose cardinality is 2^L–when searching for an optimal value. Indeed, the worst-case time complexity when maximizing Gr for the leaves of a particular Subtree-SV is $|U|$.

3.3.2 Overall Time Complexity Analysis

The worst-case 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 4—its complexity is $\mathcal{O}(|U|)$ since it traverses all leaves of s' at most, which correspond to all users, according to the worst-case scenario.

In Algorithm 3: Algorithm 4 is executed for each Subtree-SV and there are $|H|$ of them in the worst-case scenario. That is, line 6 is executed $\mathcal{O}(|H||U|)$ times. Line 9 is executed for each Subtree-SV except s_{max}, and for each shared leaf, that is, $\mathcal{O}(|H||U|)$ times. Line 10 checks out, for each terminal (resp. 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 $\mathcal{O}(|H||U|)$ times. Finally, the outer loop is executed $\mathcal{O}(|H|)$ times. Therefore, Algorithm 3's complexity is $\mathcal{O}(|H|^2|U|)$.

In Algorithm 2: lines 3 and 9 have the same complexity as line 10 in Algorithm 3, that is $\mathcal{O}(|H||U|)$. Lines 2 and 8 have the same complexity and each is executed $\mathcal{O}(|U|)$ 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

$$\mathcal{O}\Big(\min(|H|, |U|)\ |H|^2|U|\Big)$$

3.3.3 Extension

Limited Number of Terminals per Steiner Tree (Vehicles with Capacity z)

In our previous work, we further divided groups of S users into subgroups of at most z ($z < S$) users as our goal was to consider the capacity of a vehicle [44]. 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 vehicle can accommodate.

In Algorithm 3, a subset of Subtree-SVs is 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. Similar to Theorem 3.2, Theorem 3.3 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 3.3 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 3.3. *The set function $Gr(L,h)$ for a particular candidate Steiner vertex h is nondecreasing under the update rule*

$$L_{k+1} \leftarrow L_k \cup \{t\} \text{ only if } \frac{1}{Gr(L_k,h)} > Lr(t,h) \text{ or } L_k = \varnothing \tag{3.13}$$

where $t \coloneqq \arg\min_{w \in Y_k} Lr(w,h)$, $Y_k \coloneqq l(Sv(h)) \setminus L_k$.

Proof. If $L_0 = \varnothing$ then, $0 = Gr(L_0,h) < Gr(L_1,h)$.

At iteration $k > 0$:

$$Gr(L_k,h) = \frac{\sum_{w \in L_k} Ic(w)}{dist(h,M(h)) + \sum_{w \in L_k} Cl(w) + dist(w,h)} \tag{3.14}$$

And at iteration $k+1$:

$$Gr(L_{k+1},h) = \frac{Ic(t) + \sum_{w \in L_k} Ic(w)}{dist(h,M(h)) + Cl(t) + dist(t,h) + \sum_{w \in L_k} Cl(w) + dist(w,h)} \tag{3.15}$$

Let $A \coloneqq \sum_{w \in L_k} Ic(w)$, $B \coloneqq \sum_{w \in L_k} Cl(w) + dist(w,h)$, $a \coloneqq Ic(t)$, $b \coloneqq Cl(t) + dist(t,h)$ and $s \coloneqq dist(h,M(h))$. Thus, equations 3.14 and 3.15 become:

$$Gr(L_k,h) = \frac{A}{s+B} \qquad Gr(L_{k+1},h) = \frac{a+A}{s+b+B} \tag{3.16}$$

Then, from equations in 3.16:

$$Gr(L_{k+1},h) = \left(\frac{a}{s+B} + Gr(L_k,h)\right)\frac{s+b}{s+b+B} \tag{3.17}$$

From equation 3.17 we observe that $Gr(L_{k+1}, h) > Gr(L_k, h)$—nondecreasing—only if:

$$\left(\frac{a}{s+B} + Gr(L_k, h)\right)\frac{s+b}{s+b+B} > Gr(L_k, h) \tag{3.18}$$

$$\frac{1}{Gr(L_k, h)} > \frac{b}{a} \tag{3.19}$$

$$\frac{1}{Gr(L_k, h)} > Lr(t, h) \tag{3.20}$$

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

$$span(t) \coloneqq \begin{cases} 1 & t \text{ is a terminal} \\ \sum_{w \in L} span(w) & t \text{ is a pseudo-terminal} \end{cases} \tag{3.21}$$

where $L \coloneqq l(Sv(t))$.

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

Algorithm 5 builds a new Subtree-SV, based on a given one (s'), whose corresponding tree spans up to z terminals—see Definition 3.9. 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 have been analyzed, the upper bound z has been reached or the Theorem 3.3's condition does not hold—lines 7 and 8.

Line 11 checks out whether adding the terminal (resp. pseudo-terminal) with minimum loss ratio from Y_k does not surpass the upper bound. If this is the case, such terminal (resp. pseudo-terminal) is added to the set of leaves L_k—line 12—and also connected to the root h of the new Subtree-SV—line 14. Moreover, Y_k is updated accordingly—line 13.

In the scenario of vehicles with limited capacity, the call to Algorithm 4 from Algorithm 3

Algorithm 5 Accommodate z-leaves.

1: **function** ACCOMMODATE Z-LEAVES(s', z) ▷ s' : Subtree-SV
2: $h' \leftarrow r(s')$
3: $t_{min} \leftarrow \arg\min_{t \in l(s')} Lr(t, h')$
4: $L_k \leftarrow \{t_{min}\}$
5: $Y_k \leftarrow l(s') \setminus L_k$
6: Connect t_{min} with root h
7: **while** $Y_k \neq \varnothing$ **do**
8: **if** $\frac{1}{Gr(L_k, h')} \leq Lr(t_{min}, h')$ **or** $span(h) = z$ **then**
9: **break**
10: $t_{min} \leftarrow \arg\min_{t \in Y_k} Lr(t, h')$
11: **if** $span(h) + span(t_{min}) \leq z$ **then**
12: $L_k \leftarrow L_k \cup \{t_{min}\}$
13: $Y_k \leftarrow l(s') \setminus L_k$
14: Connect t_{min} with root h
15: **return** $Sv(h)$
16: **end function**

in line 6 will be replaced by a call to Algorithm 5. Since time complexity of Algorithm 5 is $\mathcal{O}(|U|)$—it traverses all leaves in Y_k at most—the overall complexity of our solution stays the same.

3.4 Experimental Evaluation

The goal of the experiments is two-fold. Firstly, we present empirical evidence of the superiority of our solution in terms of efficiency and effectiveness with respect to the modified version of our algorithm introduced previously [44], i.e., a version that considers the constraint on the feasible set of Steiner vertices. Secondly, we compare our proposed model and model A—users meet at intermediate points that are not hot-spots necessarily—and show that (a) significant difference in occurrence of *popular* meeting points—those where more than z users coincide in a time window of an hour—exists, which means that 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 vehicle occupancy rate gap significantly between both models.

We refer to our solution as *Gr-based* as a short form of "Gain-ratio-based". Baseline

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

Table 3.1: Parameter set-up in synthetic networks.

solution VST-RS is referred as *mVST-RS* as a short form of "modified VST-RS."

3.4.1 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. POI, hot-spot and user vertices are chosen randomly. Our parameters are *number of users, number of POIs, percentage of graph vertices that are hot-spots, vehicle capacity and number of vertices*. The number of users represents 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 3.1. 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* [52]. 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`. POI 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 (VISTA) 2009-2010 [70]. 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 that is more likely to be specific to users tastes.

3.4.2 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 vehicles. 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 $c(F) \coloneqq \sum_{T \in F} c(T)$ where $c(T) \coloneqq \sum_{a \in T} d_a$ and d_a is the distance of arc a.

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

3.4.3 Comparison with Baseline in Terms of Efficiency and Effectiveness

Processing Time

Theoretically, our solution is less time complex than the baseline—see Sections 3.3.2 and 2.1.5. We show that it is also significantly faster in terms of *processing time*.

With respect to synthetic graphs, Figure 3.2 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 $|U|$.

Figure 3.3 shows that our solution's efficiency is not dependent on the number of POIs ($|P|$) nor the vehicle capacity. Figure 3.3 (left) shows that the baseline's efficiency is

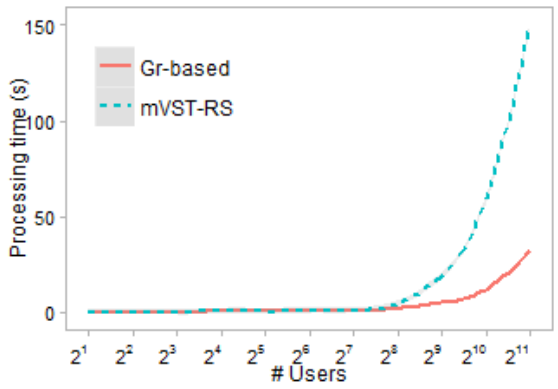

Figure 3.2: Processing time of both approaches when the number of users varies. In the case of *mVST-RS*, it grows exponentially. For *Gr-based*, it grows polynomially. Both cases are predicted on their asymptotic complexity analyses.

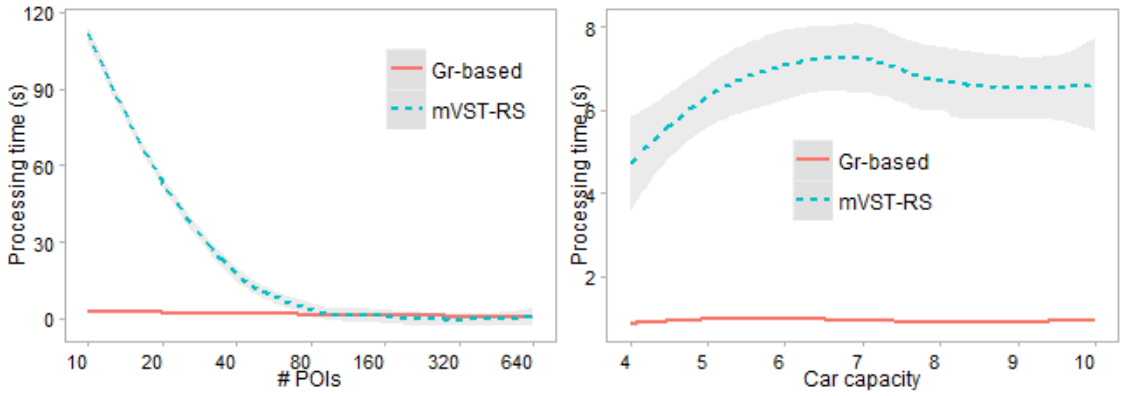

Figure 3.3: Processing time of both approaches when **(left)** the number of POIs and **(right)** vehicle capacity are varied. *Gr-based*'s processing time is invariant under both cases.

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 3.4 (left) 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 3.4 (right) shows how the baseline grows exponentially when $|V|$ increases because it must search much larger areas to find the possible meeting points, whereas our solution is scalable since it searches in $H \subset V$.

With respect to Melbourne-based graphs, Figure 3.5 shows that our solution is one to two orders of magnitude faster than the baseline for any activity throughout the day.

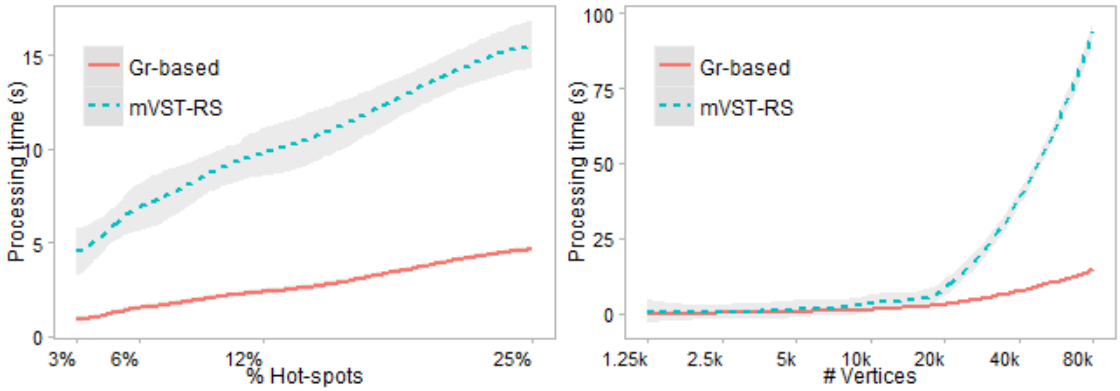

Figure 3.4: **Left.** Processing time of *Gr-based* grows linearly when the proportion of hot-spots increases. However, large proportions are unreal. **Right.** Processing time of *mVST-RS* grows exponentially when the network grows as there are much larger areas in which to search for meeting points.

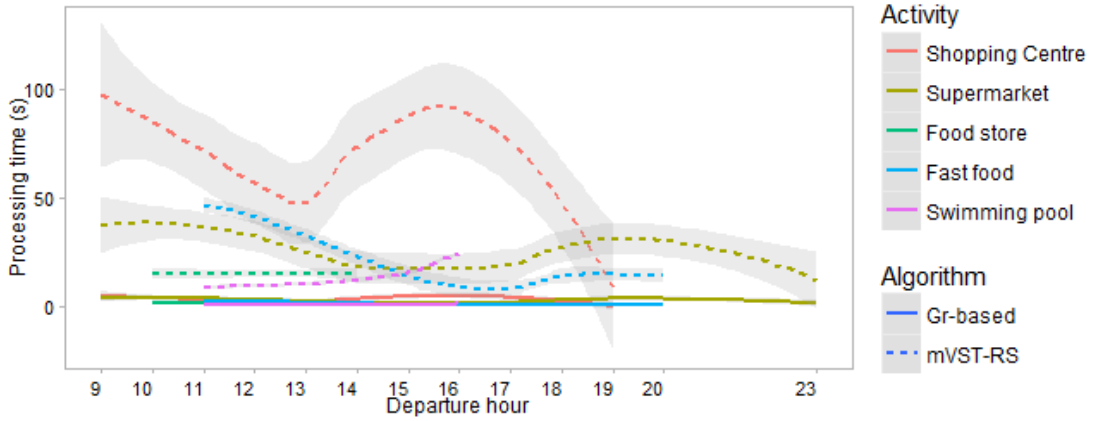

Figure 3.5: *Gr-based* is one to two orders of magnitude faster than *mVST-RS* for different activities throughout the day on Melbourne-based graphs.

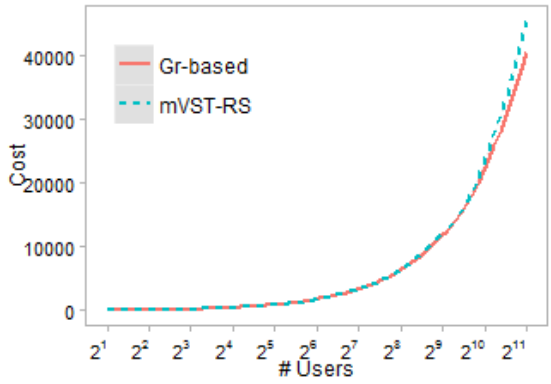

Figure 3.6: Cost of both approaches on synthetic graphs.

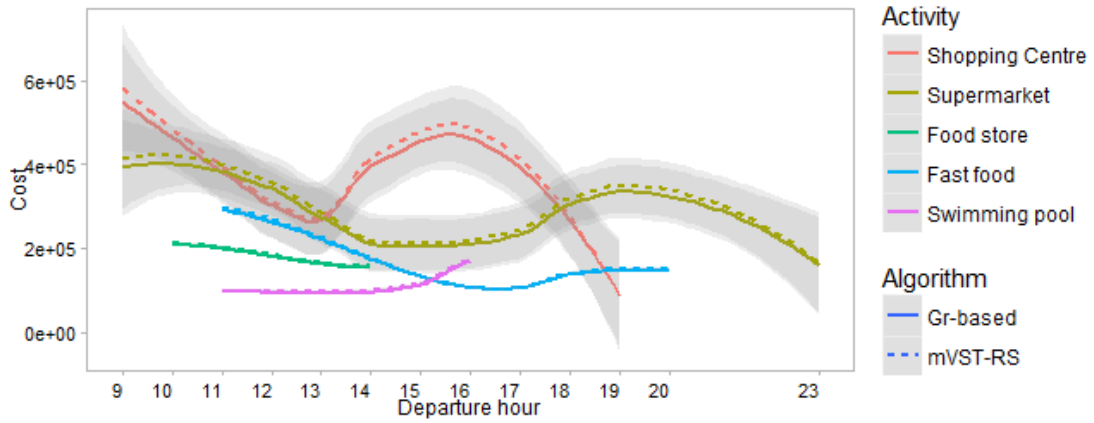

Figure 3.7: Cost of both approaches for different activities throughout the day on Melbourne-based graphs.

Cost

Although our solution is not significantly less *costly* than VST-RS, it is on average in every scenario on both kinds of graphs. Figures 3.6 and 3.7 show cost comparison on synthetic and on Melbourne-based graphs, respectively. Our solution outperforms VST-RS due to the initial division of users carried out in the first stage in 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.

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

Higher Likelihood of a Round-Trip

Figure 3.8 (left) shows the frequency of meeting points where X number of people start their travel from. Let $z = 5$ be the vehicle capacity in these experiments. Let $d \coloneqq \frac{X}{z}$ 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—see Figure 3.8 (left). 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.

Comparable Occupancy Ratio

Figure 3.8 (right) shows how a gradual uptake of our model affects the vehicle 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. If the current infrastructure were used, occupancy ratio would be affected the most. In this case, our algorithm would suggest going directly to the closest POI rather than meeting at a hot-spot that is not located within a convenient dis-

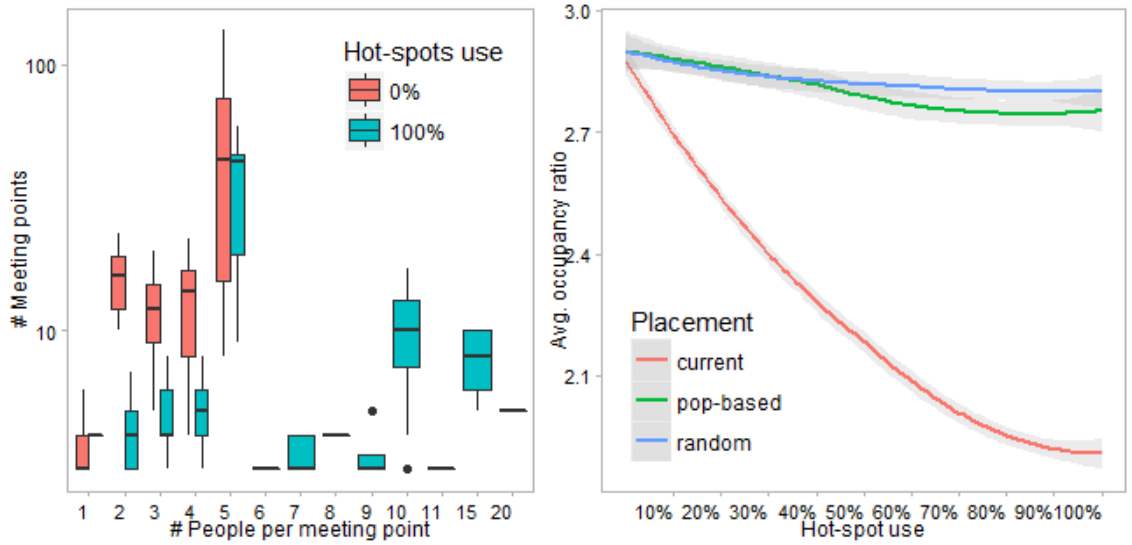

Figure 3.8: **Left.** Frequency of meeting points where a number of people—*X* axis—started their travel from. Our approach uses only hot-spots as meeting points, i.e., 100% hot-spots use, whereas the baseline uses other spots, i.e., 0% hot-spots use. **Right.** Vehicle occupancy ratio under three scenarios of placement of hot-spots at different levels of adoption of our model—gradual hot-spots use.

tance. Placement of hot-spots according to the population distribution, or even randomly, would keep vehicle occupancy ratio at similar levels than model A's. *Model A's vehicle occupancy ratio is an unattainable goal in all three scenarios as it allows meeting at non-suitable points, which are everywhere*. Of course, this is so at the cost of risking finding a suitable place to park the vehicle.

3.5 Summary

This chapter proposes the inclusion of hot-spots into the 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 vehicle occupancy rate to model A, which in turn has been proved to increase the occupancy rate compared with model B [2, 32, 60].

We define a new problem on graphs that we call Constrained Variable Steiner Tree (C-VST). Although the parametric complexity of the VST problem is reduced by con-

straining the set of feasible meeting points—Steiner vertices—to be within the set of hot-spots, we prove that the C-VST problem 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 presented in our previous work [44] 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.

Chapter 4

Congestion-Aware Activity-Based Ride-Sharing

In Chapters 2 and 3, the computed travel plans were considered independent of each other as if there is no effect on travel times when they heavily use particular segments of the road network. This is a plausible assumption since, in its current form, ride-sharing is responsible for a small fraction of traffic load compared to other transportation modes, especially private vehicles. As its benefits become more evident, and obstacles, e.g., lack of liability legislation, that may hinder its larger-scale adoption are being overcome, ride-sharing will be a more common mode of transportation. In particular, Autonomous Vehicles (AVs) are showing their proficiency on the roads, which may also catalyze ride-sharing ubiquity. For example, while an AV owner is at work, he may find appealing to offer his AV as a service or rent it to Uber so that the vehicle serves others' transportation requests. Furthermore, this disruptive technology is backed up by companies like Google—Waymo—Tesla and Uber. Therefore, ride-sharing will soon become a source of traffic congestion itself. In this chapter, we present an efficient congestion-aware ride-sharing algorithm which, instead of finding optimal travel plans based on traffic load generated by other means of transportation, it computes optimal travel plans for thousands of ride-sharing requests within a time interval. Note that in this problem, an optimal travel plan for a group of requests may affect an already computed travel plan for another concurrent group of requests, therefore plans cannot be isolated from each other.

4.1 Overview

SITUATED between the use of private vehicles and public transport, ride-sharing combines their benefits, i.e., the flexibility of a private vehicle with the low cost of public transport. Yet, there are still some obstacles, e.g., lack of legislation, security, privacy, that are slowing down ride-sharing larger scale adoption. These will eventually be

overcome as many researchers are working in this area to solve these problems. Moreover, the adoption of Autonomous Vehicles (AVs) and the predicted perspective about them as an on-demand service will catalyze ride-sharing further. For instance, imagine your AV going to pick you up and after dropping you off it may serve others' transportation requests. This scenario aligns with the vision technology companies such as Google—Waymo—Tesla and Uber have about AVs, in which AVs become an on-demand service. Indeed, this new business model has been dubbed "transport-as-a-service" [7].

With all this improved convenience, ride-sharing demand will grow as well as the traffic load generated by it. Thousands of requests will be simultaneously issued and ride-sharing will be a traffic congestion source itself. In this scenario, it is not about finding optimal ride-sharing travel plans based on the traffic load generated by other transportation modes. It is about computing optimal travel plans for multiple ride-sharing requests within a time interval that overlap at the least extent. Note that in this problem, an optimal travel plan for a group of requests may affect an already computed travel plan for another concurrent group of requests, therefore plans cannot be isolated of each other. In this chapter, we present an efficient congestion-aware ride-sharing algorithm that suggests the shortest-travel-time plans for concurrent groups of requests. This is an extension of our previous work [44] where the computation of optimal travel plans does not consider other concurrent plans.

Our congestion-aware algorithm is a meta-algorithm that iteratively calls any ride-sharing algorithm. Any model of ride-sharing can also be used, e.g., trip-based or activity-based ride-sharing—see Section 2.1. As in Chapter 3, we chose activity-based ride-sharing because it was shown in [44, 73] that matching rate is improved and travel costs are further decreased.

The travel plan depicted in Figure 4.1 (left) is the optimal plan for users u_1, u_2, u_3 and u_4 who are interested in the same activity. Let us assume that requests for two additional activities are issued within the same time interval. Users u_5 and u_6 want to perform an activity that can be carried out in Points of Interest (POIs) p_4 or p_5, whereas users u_7, u_8 and u_9 can perform their activity in p_6 only. The three corresponding travel plans—one per activity—shown in Figure 4.1 (right) are computed without taking into account the

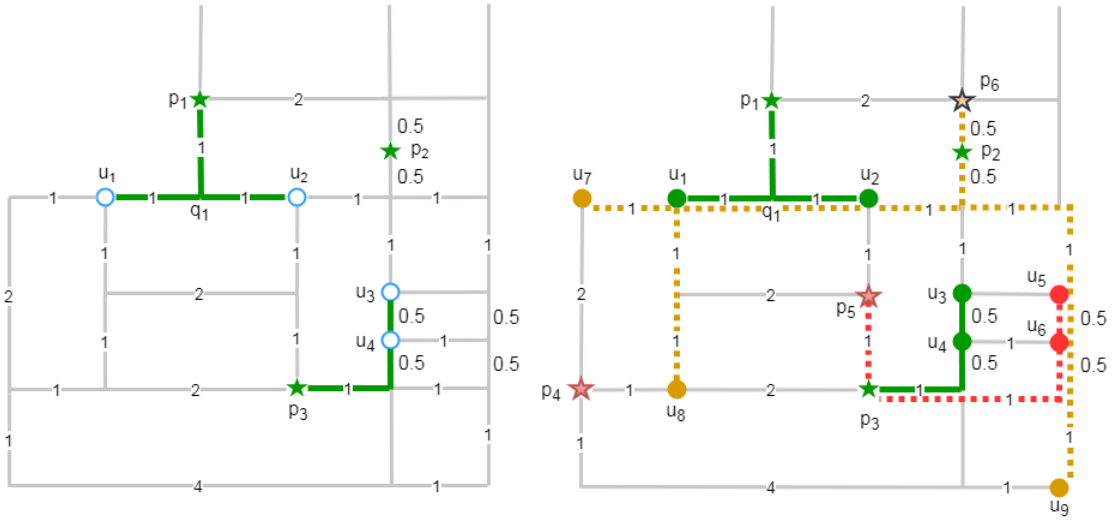

Figure 4.1: **Left.** A simple road network. Users u_1, u_2, u_3 and u_4 can perform their common activity at any of the candidate POIs p_1, p_2 and p_3. Bold edges correspond to the optimal activity-based travel plan which is the answer to requests where users indicated their origins and a common activity. **Right.** Two additional activities: $\langle\{u_5, u_6\}, \{p_4, p_5\}\rangle$ and $\langle\{u_7, u_8, u_9\}, \{p_6\}\rangle$, are processed. None of the three travel plans take into account the effect one has over the other, thus common road segments may end up congested and the plans would not be optimal anymore.

effect one has over the other. As a consequence, the common road segments may be congested, which in turn causes travel times through such segments to increase. For the sake of simplicity, we assume in this example a small number of requests and that roads can be congested with them easily. Our goal is to reduce the overall travel time for all users by re-routing the plans.

4.2 Problem Definition

A road network can be represented by the undirected graph $G \coloneqq \langle V, E\rangle$, where vertices V are the road intersections and edges E are road segments between two intersections. Let z be the same number of seats in each vehicle. Let $U \subseteq V$ and $P \subseteq V$ be the sets of users and POIs, respectively. We aim to find interconnecting routes—travel plans—from all members of U to one or more elements of P that minimizes the total cost of the edges on the routes.

Within a time interval, users U issue their requests where an individual request com-

prises pick-up location and activity. In this chapter we assume users ride-share provided they are to perform the same activity only. Thus, individual requests are grouped by activity. For each activity i we get POIs $P_i \subseteq P$ where users $U_i \subseteq U$ can carry out activity i. A travel plan that corresponds to activity i will be computed for the subset of users and POIs $\langle U_i, P_i \rangle$.

Finding the optimal travel plan for *one activity* involves dividing all users within that activity into groups so that each group can fit in a vehicle—vehicles have limited capacity z. Then, the total travel time of the users within each group to the meeting points and then to the most convenient POI should be minimum. Solving this problem is equivalent to solving the Variable Steiner Tree (VST) problem introduced in [44]. In other words, we need to find out the optimal division of U_i into subsets $\{U_i^j\}$, and for each subset U_i^j find out a POI $p_i^j \in P_i$ such that the tree T_i^j that spans $U_i^j \cup \{p_i^j\}$ is a VST—see Section 2.1.4. The total cost of the road segments of T_i^j is given by $Cost(T_i^j) \coloneqq \sum_{e \in T_i^j} c_e$, where c_e corresponds to the cost of the road segment e that is part of the VST. Hence, a forest f_i—travel plan for activity i—can be defined as $f_i \coloneqq \bigcup_{j=1}^{t} T_i^j$, where t is the number of VSTs for this activity. The cost of the travel plan for activity i is then given by $Cost(f_i) \coloneqq \sum_{j=1}^{t} Cost(T_i^j)$.

In this chapter, we consider congestion. Within a travel plan f_i, VSTs T_i^j may overlap causing congestion. Travel time—cost—on congested road segments tends to increase as a result of growing traffic load, first slowly but after more rapidly if conditions do not change. The costs are usually modeled as positive, nonlinear, and strictly increasing functions of traffic load. Thus, the cost of T_i^j shifts to $Cost(T_i^j) \coloneqq \sum_{e \in T_i^j} \psi(l_e, c_e)$, where $\psi(l_e, c_e)$ corresponds to travel time as a function of traffic load l_e and free flow travel time c_e of the road segment e.

The problem we aim to solve in this chapter can be defined as:

Given: An undirected road network $G \coloneqq \langle V, E \rangle$, set of k activities requested within a time interval, set of users U and POIs P grouped into $\{\langle U_i, P_i \rangle\}_{i=1}^{k}$ where U_i and P_i correspond to activity i.

Find: A set of forests F such that $F \coloneqq \arg\min_{F' \in \mathbb{F}} \sum_{i:1<i\leq k \;\&\; f_i \in F'} Cost(f_i)$, where $\mathbb{F}$ is the set of all possible sets of forests.

Here the cardinality of $\mathbb{F}$ may be large. A brute force approach to explore $\mathbb{F}$ would be prohibitive. In our approximate approach, for the sake of simplicity, we assume that one activity alone does not lead to any significant congestion and thus, we optimize between activities and not within one activity.

4.3 Our Solution

We chose VST-based Ride-Sharing (VST-RS) [44] as our non-congestion aware approach and as baseline. It computes an optimal travel plan for a set of users and POIs which we assume correspond to one activity. Such travel plan is suitable for this set of users since they can perform their common activity in any POI included in this set. At the same time, another set of users may want to travel and carry out a different activity which can be performed in another set of POIs. Both travel plans can be computed independently without interfering to each other. All this assumes that ride-sharing does not cause congestion by itself. If it does, one must consider each ride-sharing arrangement in light of others. This is a prohibitively expensive task. In our congestion-aware solution, for the sake of simplicity, we assume that arrangements within an activity do not affect each other, but instead, ride-sharing-based traffic between activities should be considered.

Figure 4.2 (left) shows three activities which are processed without taking into account the effect one travel plan has over the other. There are some overlaps between the travel paths for different activities. As a consequence, the common road segments may be congested if we assume their capacity is considerably small[1]. Let $OptCost_{NCA}$ and

[1] In this hypothetical example, the capacities of the road segments are small and two vehicles in the same road segment will cause congestion for simplicity of the presentation.

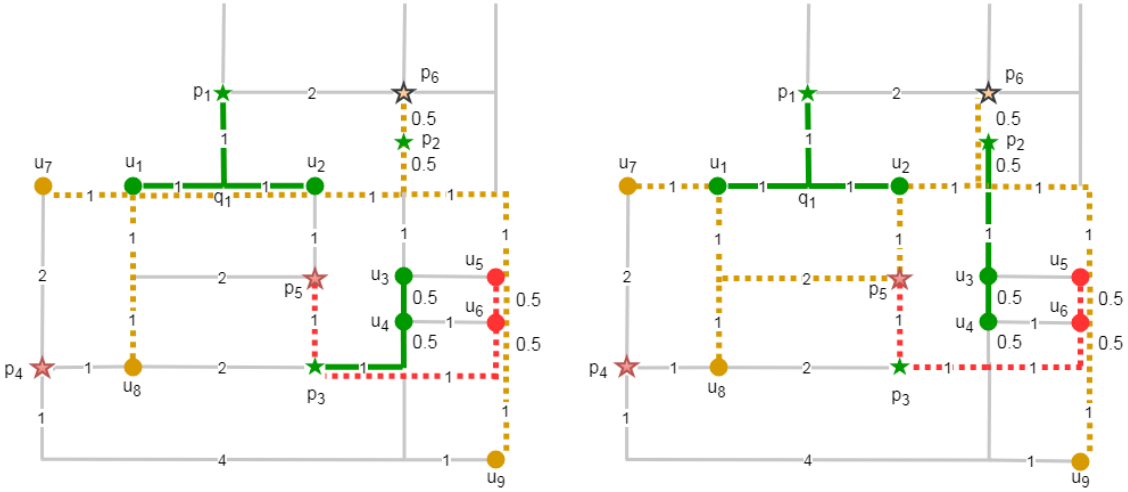

Figure 4.2: **Left.** A non-congestion-aware solution. Three activities $\langle\{u_1, u_2, u_3, u_4\}, \{p_1, p_2, p_3\}\rangle$, $\langle\{u_5, u_6\}, \{p_4, p_5\}\rangle$ and $\langle\{u_7, u_8, u_9\}, \{p_6\}\rangle$, are processed without taking into account the effect one travel plan has over the other. The common road segments may be congested, and the plans are not optimal anymore. **Right.** Congestion-aware solution. Travel plans are re-routed, consequently congestion is minimized, and the total travel time is improved.

$OptCost_{CA}$ be the optimal total travel times of the non-congestion-aware and congestion-aware solutions in Figures 4.2 (left) and 4.2 (right), respectively. We measure the travel time based on a function $\psi(l_e, c_e)$, where l_e is the load—number of concurrent vehicles—and c_e is the current travel time of road segment e. Thus, $OptCost_{NCA} = 12\psi(1,1) + 8\psi(2,0.5)$ and $OptCost_{CA} = 18\psi(1,1) + 3\psi(2,0.5)$. In other words, there are 12 one-unit segments with only 1 vehicle and 8 half-unit segments with 2 vehicles in $OptCost_{NCA}$; whereas there are 18 one-unit segments with only 1 vehicle and 3 half-unit segments with 2 vehicles in $OptCost_{CA}$. The selection of $\psi(\cdot)$ will determine whether the congestion-aware solution is less expensive than its counterpart. For example, assuming that $\psi(1,1) = 1$ and $\psi(2,0.5) = 2$ would result in a less expensive congestion-aware solution as $OptCost_{NCA}$ is 28 and $OptCost_{CA}$ is 24.

In this chapter, we adopt the widely used function within transportation science called Bureau of Public Roads (BPR) function [49]. This function is defined as:

$$bpr(l_e, c_e) \coloneqq c_e\Big(1 + \alpha\Big(\frac{l_e}{y_e}\Big)^{\beta}\Big) \tag{4.1}$$

where l_e is the traffic load, c_e is the free flow travel time and y_e is the capacity of the road segment e; and parameters α and β vary according to the characteristics of the network. With higher values of β, the effects of the congestion appear more rapidly. Parameter α reflects the weight the term $(\frac{l_e}{y_e})^{\beta}$ has in the formula and consequently, it determines the increase over the original travel time. Our choice of values for these parameters is specified in Section 4.4.

4.3.1 Overview of Our Solution

A congestion-aware brute-force approach would first choose an activity and compute its travel plan. As it is the first activity, i.e., there is no travel plan already loaded in the network, this travel plan stays put. A second activity is chosen and after its travel plan is computed, re-routing of this plan over congested segments is performed. Then, a third activity is chosen, and re-routing of this plan is also performed over congested segments. This process continues until the last activity. Then, a new permutation—ordering—of activities is chosen and the whole process is repeated. The final solution includes the set of travel plans with minimum total cost. For a moderate number of activities, e.g., 10, this brute force approach would compute 10! travel plans.

Our approximate approach VST-based Congestion-Aware (VST-CA) avoids trying every possible ordering of activities. Instead, it computes the travel plans for a specific ordering, but it makes more than one pass for this ordering. Between each pass—each iteration—travel times of congested segments are updated by using a function $\psi(\cdot)$ of the traffic load, i.e., congested segments become more expensive. Thus, in following iterations, travel plans computed for activities located at the beginning of such ordering would implicitly consider the travel plans of activities located at the end of the ordering computed in previous iterations. Within each iteration, no re-routing is made. Re-routing is implicitly performed when a travel plan is computed in the following iterations since expensive routes are less appealing to be included in them. Then, for a moderate number of activities, e.g., 10, $10x$ travel plans would be computed, where x is the maximum number of iterations. This number x is based on the convergence of traffic loads. That is, the algorithm stops when the traffic loads of the road segments between iterations do not

change. Convergence of traffic loads in VST-CA is analyzed in Section 4.3.3.

This iterative cost-updating approach is inspired in both Baltz and Srivastav's approach [10] for the Multicast Congestion (MC) problem—see Section 2.1.4—and especially in a popular approach within traffic science called Static Traffic Assignment (STA). In [10], their practical congestion algorithm has two phases. It does a first pass over the set of multicast requests and for each request computes an ST, updates the loads and the costs of the segments so that the computation of the ST for the next request will take this into account. In the second phase, the algorithm iterates over the set of multicast requests up to 100 times. In each iteration and in each request, costs of the segments of the whole network and of the tree for that request are updated. Then, a new ST is computed, and it replaces the ST for the same request if this new tree is less expensive. Then, loads of the segments are updated for the construction of the next ST. Different cost-updating functions are used in each phase and the set of multicast requests is arranged in no specific way.

In STA, traffic planners are interested in knowing what would happen to the network, particularly in terms of delay, if a considerable number of people travel through specific road segments. To figure it out, traffic is loaded into the network according to an Origin-Destination (O-D) matrix, which indicates how many people travel from zone i to zone j in cell (i, j). This loading process involves an iterative process of shortest-path computations between zones until it reaches user-equilibrium[2]. Many authors and institutions have proposed their STA approaches and Patriksson [53] summarizes them into a four-step process. First, an all-or-nothing solution is computed based on free-flow travel times for all the travel requests from the O-D matrix. All-or-nothing assignment means that all the travel requests within a pair of zones are assigned to the shortest path between such zones, and none to other paths. However, in this initialization step, only a fraction of the total demand is loaded into the network. Then, the following steps are in a loop until user-equilibrium is reached. The second step computes shortest paths based on travel times from previous iterations. In the third step, a fraction of the shortest paths computed in the previous step are loaded. Finally, the convergence criterion is checked.

[2] Also known as Wardrop's equilibrium. Under equilibrium conditions, no traveler can find a lower-cost route if unilateral decisions are made.

It is worth to notice how similar these two approaches address congestion while at the same time minimize costs. In both cases, the order of the requests is secondary, and they are iterated many times and in each iteration, the costs of the segments are modified. Our travel plans are just a generalization of Steiner Trees and Shortest Paths[3]. Thus, the spirit of these two approaches can be adopted to deal with VSTs—our travel plans.

4.3.2 Algorithm

VST-CA—Algorithm 6—is our iterative congestion-aware algorithm. VST-CA receives a set of tuples A as a parameter, where each tuple is of the form $\langle U_i, P_i \rangle$. U_i is the set of users who are interested in performing activity i. P_i is the set of POIs where such activity can be performed in. In each iteration of VST-CA, travel plans for all activities are processed. As indicated before, the ordering in which travel plans are processed is not relevant in our approach.

At the beginning of each iteration—line 4—traffic loads of all road segments are reset. Then, VST-CA processes each activity in the "for loop" from line 5 to line 15. Note that a new travel plan p_i for activity i is computed by VST-RS in line 8, only if a random number falls within a changing probability λ. Otherwise, the current iteration's plan $plan_i^k$ is updated with the previous iteration's $plan_i^{k-1}$ in line 14. In the first iteration, travel plans for all activities are computed because $\lambda = 1$. The reason of λ is two-fold: (1) it leads to convergence of VST-CA, and (2) it mimics the human behavior that not everyone chooses to switch to a new plan. In Section 4.3.3, we analyze the convergence of VST-CA.

In the case a new travel plan p_i is computed by VST-RS, the current iteration's plan $plan_i^k$ is updated with p_i in line 10, only if $Cost(p_i)$ is less than $Cost(plan_i^{k-1})$. Otherwise, the current iteration's plan is updated with the previous iteration's in line 12. After the current iteration's plan $plan_i^k$ has been set, the traffic loads of the road segments that belong to such plan are updated in line 15. Note that travel times are not updated. This task is deferred until the travel plans for all activities, in this iteration, are set. Travel times are re-computed in line 16 according to function $\psi(\cdot)$. This is a function of the traffic

[3]In shortest paths, two vertices are connected. In Steiner Trees, more than two vertices are connected. In VSTs—our travel plans—more than two vertices are connected, and there exist variable roots.

loads and current travel times which makes congested road segments expensive. This iterative process stops when either convergence of traffic loads or a maximum number of iterations is reached. This stop condition is verified in line 3.

Since travel times are not updated within the "for loop" from line 5 to line 15, computation of travel plans could be parallelized. Moreover, the independence VST-RS has with respect to shortest-paths pre-processing is rather important for VST-CA in terms of efficiency. VST-CA updates travel times in each iteration, which would have involved to re-compute shortest paths for all pairs otherwise. The asymptotic time and space complexities of VST-CA are the same as VST-RS because this iterative process is performed *maxiter* times at most.

Algorithm 6 Compute congestion-aware optimal travel plans for simultaneous activities.

1: **procedure** VST-CA($A, G, maxiter$) ▷ A : set of tuples of users and POIs per activity, G : graph, *maxiter* : max. number of iterations.
2: $k \leftarrow 1$
3: **while** traffic loads keep changing **and** $k \leq maxiter$ **do**
4: $l_e \leftarrow 0, \quad \forall e \in E(G)$ ▷ reset traffic loads
5: **for each** tuple $\langle U_i, P_i \rangle \in A$ **do**
6: $\lambda \leftarrow 1/k$ ▷ probability of plans to be reassigned
7: **if** $rnd() < \lambda$ **then** ▷ $rnd() \in [0.0, 1.0)$: pseudo-random number
8: $p_i \leftarrow$ VST-RS(U_i, P_i, G) ▷ compute new travel plan
9: **if** $Cost(p_i) < Cost(plan_i^{k-1})$ **then**
10: $plan_i^k \leftarrow p_i$ ▷ switch to new plan only if its travel time is shorter
11: **else**
12: $plan_i^k \leftarrow plan_i^{k-1}$ ▷ otherwise, stick to the previous one
13: **else**
14: $plan_i^k \leftarrow plan_i^{k-1}$ ▷ plan for activity i from previous iteration
15: $l_e \leftarrow l_e + 1, \quad \forall e \in E(plan_i^k)$ ▷ update traffic loads
16: $c_e^k \leftarrow \psi(l_e, c_e^{k-1}), \quad \forall e \in E(G)$ ▷ update travel times. c_e^0 : free-flow travel times
17: $k \leftarrow k + 1$
18: **end procedure**

4.3.3 Convergence of Our Algorithm

In VST-CA—Algorithm 6—we compute a new travel plan only if a random number falls within a changing probability λ. This strategy is adopted for two reasons: (1) it leads to

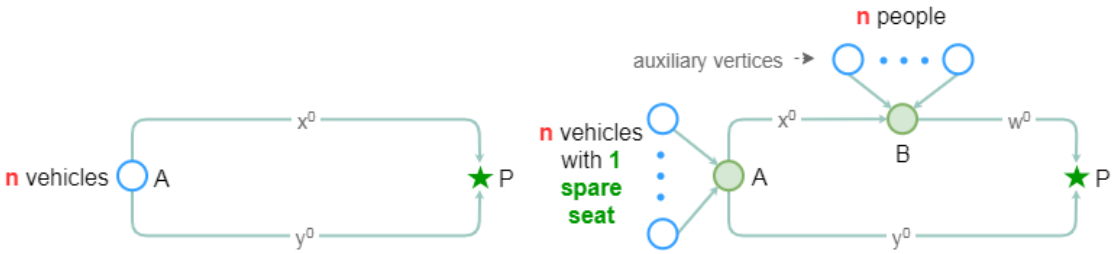

Figure 4.3: Oscillation may happen in an *all-or-nothing* assignment of **(left)** *shortest paths*, and **(right)** *VSTs*.

convergence of VST-CA, and (2) it mimics the human behavior that not everyone chooses to switch to a new plan and those who chose to do so are less as of time passes. In this section we focus on the first.

Oscillation is a behavior that has been observed in STA when "all-or-nothing" assignment of shortest paths is used [68]. All-or-nothing assignment means that all trips within a pair of zones are assigned to the shortest path between such zones, and none to other paths. For example, in Figure 4.3 (left), n vehicles want to go from zone A to POI P and $x^0 < y^0$, where x^0 and y^0 are the free-flow travel times of the routes between these two zones. Based on these times, this procedure assigns n vehicles to the top route and 0 to the bottom one. Then, new travel times x^1 and y^1 are computed based on that load. Thus, $x^1 \leftarrow \psi(n, x^0), y^1 \leftarrow y^0$. In the new iteration, if $x^1 > y^1$ then, the all-or-nothing assignment is: 0 vehicles to the top route, n vehicles to the bottom route. New travel times are computed: $x^2 \leftarrow x^1 = \psi(n, x^0), y^2 \leftarrow \psi(n, y^1) = \psi(n, y^0)$. Since $x^0 < y^0$ then, $x^2 < y^2$ and we have reached oscillation. Thus, the necessary and sufficient conditions for oscillation in this scenario are: (1) $x^0 < y^0$, and (2) $\psi(n, x^0) > y^0$.

Note that oscillation prevents all-or-nothing assignment from converging because traffic loads keep changing between iterations. In VST-CA, all-or-nothing assignment would have been used if the travel plan $plan_i^k$ for activity i were *always* updated with a new travel plan p_i computed by VST-RS. We are interested in showing whether oscillation can occur with travel plans—VSTs—as well as with shortest paths.

The easiest scenario, where oscillation may happen in our case, is to assume that there are n vehicles of ride-sharers, all going to the same POI P and there are only two routes, such as the ones presented for shortest paths in Figure 4.3 (left). We dismiss this scenario,

although it may happen, it is trivial. A more interesting scenario is shown in Figure 4.3 (right). Here, we assume that the n optimal travel plans for n activities bring n vehicles, with 1 spare seat each, to vertex A. The last leg of each plan involves picking up a user from vertex B. Since in our problem, more than one user does not share the starting point, we create auxiliary vertices, all connected to B with no-travel-time edges, i.e., travel time is 0. To finish setting up the scenario, the edges connecting the first part of the travel plans to A have the same travel time.

As we are assuming that the optimal travel plans involve picking up the users from B, we can conclude that $x^0 + w^0 \leq y^0$. In an all-or-nothing assignment of these VSTs, n vehicles will be assigned to the route segments $\overline{AB}$ and $\overline{BP}$, and 0 to the segment $\overline{AP}$. It is worth to notice that the relation $x^0 < y^0$ suffices to still choose the route segments $\overline{AB}$ and $\overline{BP}$ over $\overline{AP}$. The selection of the optimal travel plan is invariant to w^k, i.e., $x^k + w^k$—cost of a plan where users in B are picked up—*versus* $y^k + w^k$—cost of a plan where n vehicles take the bottom route and the users in B go independently to P. Therefore, this scenario reduces to the shortest-path scenario and the necessary and sufficient conditions for oscillation are the same, i.e., (1) $x^0 < y^0$, and (2) $\psi(n, x^0) > y^0$.

We observe that oscillation is almost unavoidable in all-or-nothing assignments. Therefore, to force convergence in VST-CA, we control the portion of plans that need to be re-computed through a random parameter λ. The parameter decreases as the algorithm proceeds to new iterations. It will end up with a value close to 0, which in turn will lead to keeping the travel plans unchanged between the last two iterations. This results into traffic loads convergence. This idea is similar to how STA proceeds to avoid oscillation since it considers a fraction of the whole demand of all-or-nothing shortest paths when loading them into the network.

4.4 Experimental Evaluation

In this section, we make a comparison between our congestion-aware approach VST-CA and VST-RS. We used city road networks, namely Manhattan, Melbourne and Quito, to create congestion at city-level. We generated a grid-like road network where the length of

each edge was set randomly 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.

We are interested in testing whether VST-CA is able to reduce the total travel time resulting from congestion generated by ride-sharing. Such congestion is generated by the ride-sharing travel plans computed for all activities requested within a time interval by our non-congestion-aware approach VST-RS. Then, VST-CA is also run for the same sample and the total travel costs resulting from both approaches are compared.

Travel cost is the value of the BPR function which has parameters α and β. Parameter α determines the increase over the original travel time, therefore we choose to keep β constant and show the effect of different values of α. Parameter $\beta = 4.0$ is chosen as this is the value widely used in traffic science.

Travel cost was measured for VST-RS and VST-CA with different parameters for real and synthetic networks. The values of these parameters and the number of samples that have the parameter configured with that value are shown in Tables 4.1 and 4.2. *User density* and *POI density* are parameters that correspond to the ratios $\frac{\#\ Users}{Network\ size}$ and $\frac{\#\ POIs}{Network\ size}$, respectively. In the case of a real network, a sample corresponds to a random square within a city of roughly 9 km^2. User density is the only parameter that is varied for a city and POI density is computed based on the existing POIs within the sample. In the case of a synthetic network, a sample is a square grid-like network whose edge costs are drawn from a uniform distribution between 0 and 1. In these networks, users are uniformly distributed, and POI locations follow either a uniform or Zipfian distribution. Larger values of user and POI densities were ruled out because the networks were almost fully-congested or there was not ride-sharing at all, respectively. The total number of samples for real and synthetic networks are 1210 and 1395, respectively.

In each sample, the *proportion* of travel cost of VST-CA with respect to the travel cost of VST-RS is computed. If the value of the proportion is less than 1, it means that VST-CA was able to reduce the travel cost. If the value of the proportion is 1, it means that VST-CA had no effect and there was no reduction. This proportion cannot be greater than 1 be-

Table 4.1: Parameter set-up in real networks.

Parameter		# Samples
City (Borough)	Manhattan	260
	Melbourne	487
	Quito	463
User density	(0.0, 0.2]	884
	(0.2, 0.4]	230
	(0.4, 0.6]	64
	(0.6, 0.8]	26
	(0.8, 1.0)	6
POI density	(0.0, 0.1]	929
	(0.1, 0.2]	178
	(0.2, 0.3]	59
	(0.3, 0.5)	44

Table 4.2: Parameter set-up in synthetic networks.

Parameter		# Samples
POIs dist.	Uniform	697
	Zipfian	698
α	0.15	475
	0.50	440
	1.00	480
User density	(0.00, 0.02]	60
	(0.02, 0.04]	180
	(0.04, 0.08]	319
	(0.08, 0.16]	338
	(0.16, 0.32]	318
	(0.32, 0.64]	180
POI density	(0.00, 0.01]	640
	(0.01, 0.02]	377
	(0.02, 0.03]	20
	(0.03, 0.04]	358

cause, in VST-CA, a plan is updated with a new plan only if the latter is cheaper than the former—see Algorithm 6, line 9.

Among many factors, congestion in road networks is a capacity-dependent matter. Capacity of road segments varies greatly, and such information is not always possible to obtain. We run the experiments with uniform capacities assuming different levels of traffic pre-loading.

We introduce a metric we call Weighted Average of Relative Loads (WARL). This metric is intended to indicate the *level of congestion resulting from the traffic assignment using VST-RS—our non-congestion-aware approach*. This is because, *besides confirming whether VST-CA can reduce travel cost, we are also interested in seeing at what level of congestion produced by VST-RS, VST-CA is indeed able to reduce it*. Let l_e, c_e^0 and y_e be the current load, free-flow travel time and capacity of the road segment e, respectively. First, let us define relative load as the ratio $r_e \coloneqq \frac{l_e}{y_e}$ and $w_e \coloneqq \frac{c_e^0}{\sum_{e \in E} c_e^0}$ as the weighted free-flow travel time of road segment e. The product $r_e w_e$ is the weighted relative load. Then, WARL is the aggregation of the weighted relative loads of the road segments in the solution: $WARL \coloneqq \sum_{e \in F} r_e w_e$, where F is the solution, i.e., set of travel plans—VSTs—for all activities.

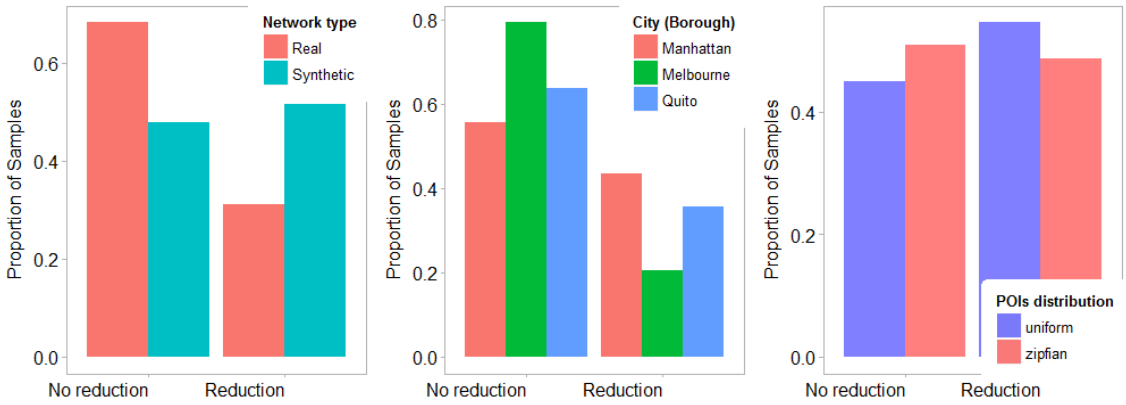

Figure 4.4: **Left.** Proportion of samples for each type of network grouped by whether travel cost is reduced or not by VST-CA. **Center.** In real networks. **Right.** In synthetic networks.

4.4.1 Effect of VST-CA on Travel Cost

Figure 4.4 (left) shows the proportion of samples where VST-CA reduced and did not reduce travel cost, in *real* and *synthetic* networks. Figure 4.4 (center) is for real networks only and proportions are within each city—borough. Figure 4.4 (right) is for synthetic networks only and proportions are within each POI distribution, i.e., *uniform* and *Zipfian*.

It can be observed in Figure 4.4 (left) that VST-CA did reduce the travel cost in both types of networks, but the proportion of observations with no improvement exceeded the travel-cost-reduced ones in real networks. In Figure 4.4 (center), Melbourne and Quito had many samples where VST-CA was not able to reduce travel cost. There is no clear superiority on performance when VST-CA is run on uniform- or Zipfian-POI-distributed networks as it is shown in Figure 4.4 (right).

In the case of the behavior observed in real networks, particularly in Melbourne and Quito, we mainly attribute it to:

1. High number of samples with *low user density*, as it can be observed in Figure 4.5 (left) for Quito and Melbourne.

2. High number of samples with *low POI density*, as it noticeably happened with Melbourne—see Figure 4.5 (center).

3. Existence of several samples for Quito and Melbourne where *the number of POIs*

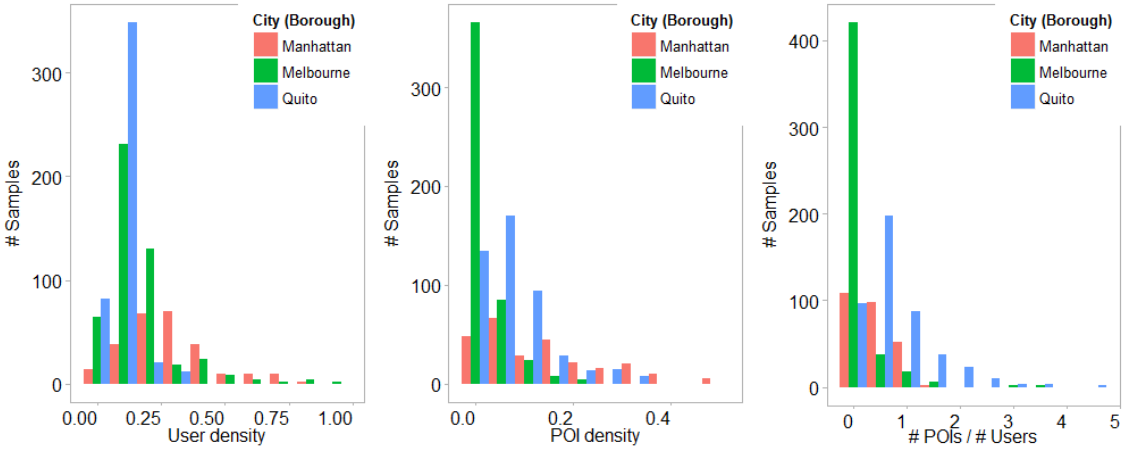

Figure 4.5: **Left.** High number of samples from Melbourne and Quito have low user density. On the other hand, Manhattan has a more uniform distribution. **Center.** High number of samples with low POI density, especially in Melbourne. This causes too much congestion at the last mile. **Right.** Some Melbourne and Quito samples have more POIs than users—right tail of the histogram—hence congestion is hardly reached. This phenomenon does not occur with Manhattan.

surpassed the number of users—up to 5 times. See the right tail of the histogram in Figure 4.5 (right). This phenomenon does not occur with Manhattan.

When the number of users is low—low user density—or the ratio between the number of POIs and the number of users is high, i.e., cases (1) and (3) mentioned above, congestion may not be reached, and travel-cost reduction is not needed. An example of case (3) is shown in Figure 4.6 (left). In this sample, there are more POIs—markers—than users—small points—hence a ride-sharing arrangement does not involve traversing the whole city. Only a subset of road segments is occupied by more than two vehicles[4], so there is not enough load to affect travel time considerably. There are even some users for whom it is better to go directly to their destination, thus they do not coincide on their paths with other drivers, as it is highlighted by the circle in the same figure.

When POI density is low, i.e., case (2) mentioned above, high levels of last-mile congestion occur, hence cost reduction is not possible either. Figure 4.6 (right) shows a sample network from Melbourne where there are only two POIs close to each other, thus driving to them causes last-mile congestion as the number of users is relatively larger and the last

[4]For some road segments, a small number of vehicles is needed to affect their travel time as they were pre-loaded for the experiments.

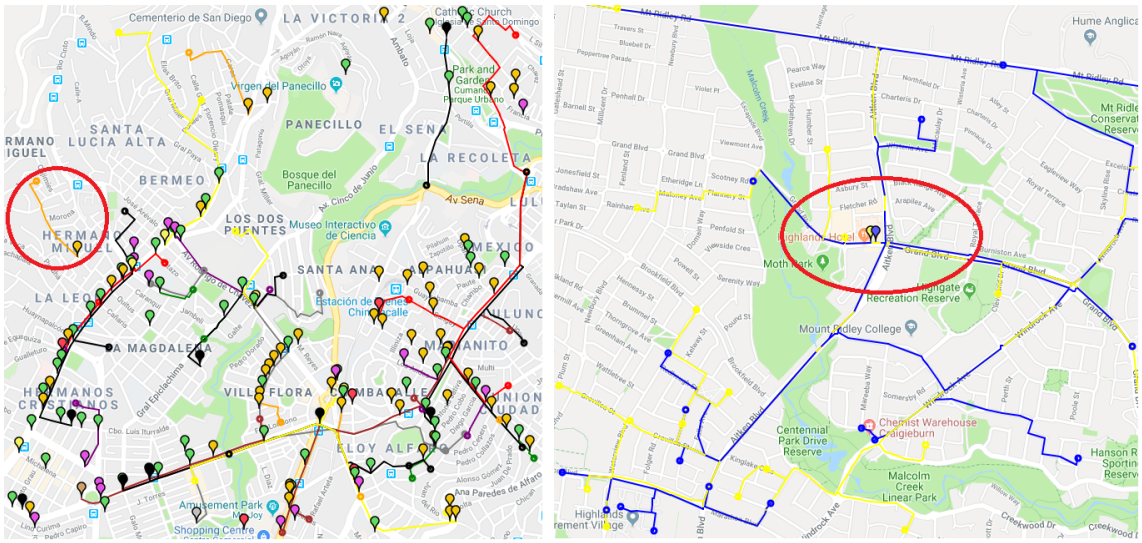

Figure 4.6: **Left.** Sample network from Quito. There are more POIs—markers—than users—small points—so congestion is hardly attained as ride-sharing arrangements do not traverse the whole city. There may even be users for whom it is better to go directly to their POI, thus they do not coincide on their paths with other drivers—encircled. **Right.** Two POIs—markers—located close to each other in a sample network from Melbourne. There is congestion at the last-mile which is difficult to reduce.

legs of their travel plans coincide.

In the case of Manhattan, Figure 4.7 shows the travel plan of VST-RS and VST-CA. Note how VST-RS suggests most users—small points—to go through the shortest route—encircled—i.e., shortest in terms of free-flow travel time. Thus, it gets congested and VST-CA is able to reduce the travel cost by suggesting alternative routes.

We notice that there is a specific WARL interval where VST-CA is able to reduce travel cost by more than 10%. That is, within this interval, VST-CA could improve travel cost significantly, whereas outside this interval, it could not. Figure 4.8 shows travel-cost reduction by WARL for real and synthetic networks. Note that significant reduction, i.e., $> 10\%$, is localized within a WARL interval only. This interval is different for real and synthetic networks. VST-CA could not improve travel time when the network is either almost free or over-congested—outside this interval.

The results for both real and synthetic networks for all parameters are shown in Tables 4.3 and 4.4.

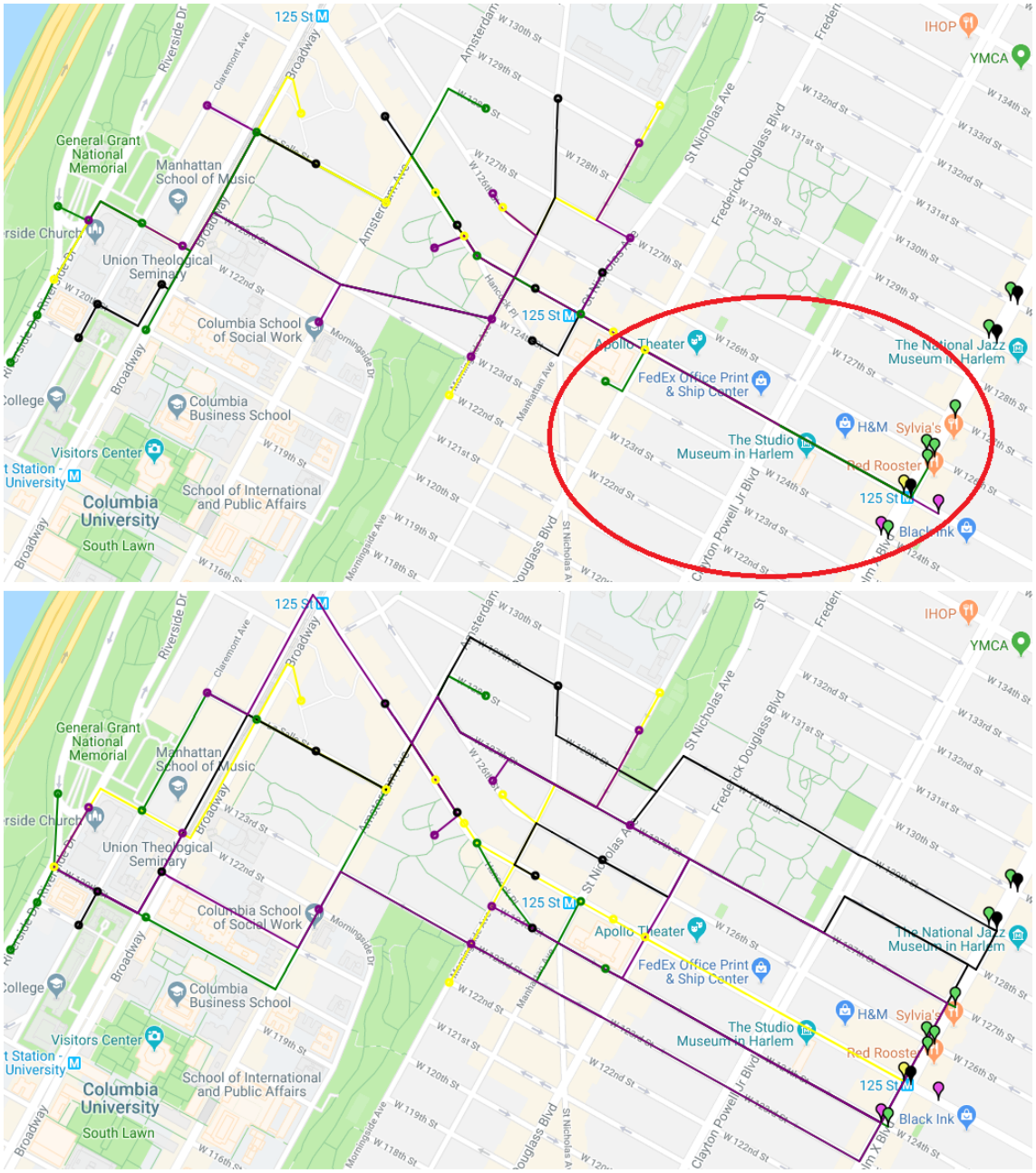

Figure 4.7: VST-RS **(top)** and VST-CA **(bottom)** in action on a Manhattan sample. Note how VST-RS suggests most users—small points—to travel along *Martin Luther King Jr. Blvd.*—encircled—which is the shortest in terms of free-flow travel time. VST-CA was able to fix the travel plan and reduce its cost after 3 iterations by suggesting alternative routes.

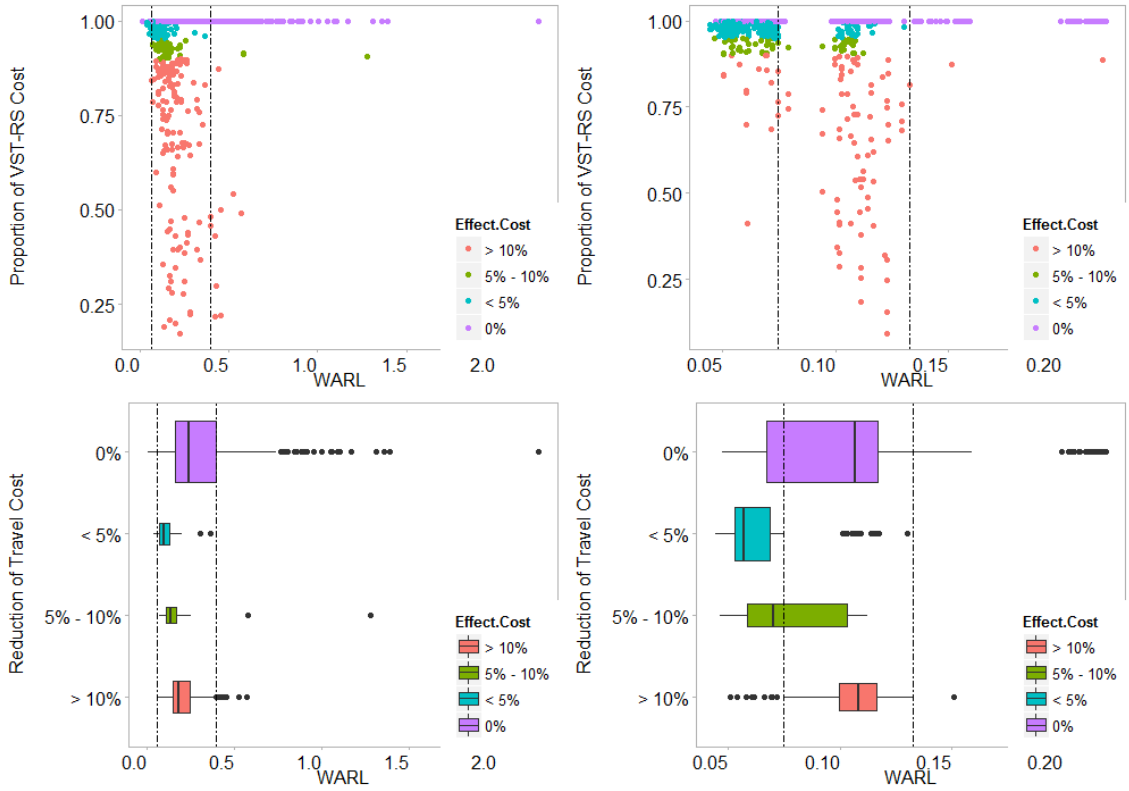

Figure 4.8: Interval of WARL—between vertical lines—within which larger (> 10%) travel-cost reduction is possible in real **(left)** and synthetic **(right)** networks.

Table 4.3: Effect of VST-CA on Travel Cost in Real Networks. Intervals of WARL where the largest cost reduction was reached are bold. Column **"Reduction"** is the sum of "> 10%", "5% – 10%" and "< 5%" columns.

Parameter	Reduction of Travel Cost					# Samples
	> 10%	5% – 10%	< 5%	Reduction	No Reduction	
# Samples	221	58	101	380	830	1210
City (Borough)						
Manhattan	23.66%	10.31%	9.54%	43.51%	56.49%	260
Melbourne	15.57%	2.46%	2.46%	20.49%	79.51%	487
Quito	17.81%	4.08%	13.73%	35.62%	64.38%	463
User density						
(0.0, 0.2]	19.82%	3.15%	8.78%	31.75%	68.25%	884
(0.2, 0.4]	12.07%	10.34%	9.48%	31.89%	68.11%	230
(0.4, 0.6]	25.00%	6.25%	0.00%	31.25%	68.75%	64
(0.6, 0.8]	3.85%	7.69%	3.85%	15.39%	84.61%	26
(0.8, 1.0)	0.00%	0.00%	0.00%	0.00%	100.00%	6
POI density						
(0.0, 0.1]	17.27%	2.36%	4.61%	24.24%	75.76%	929
(0.1, 0.2]	24.44%	10.56%	12.22%	47.22%	52.78%	178
(0.2, 0.3]	11.67%	11.67%	45.00%	68.34%	31.66%	59
(0.3, 0.5)	20.45%	22.73%	20.45%	63.63%	36.37%	44
WARL						
(0.0, 0.2]	**20.82%**	7.70%	15.74%	44.26%	55.74%	604
(0.2, 0.4]	**20.72%**	1.81%	1.30%	23.83%	76.17%	386
(0.4, 0.8]	7.95%	1.14%	0.00%	9.09%	90.91%	176
(0.8, 2.4)	0.00%	4.55%	0.00%	4.55%	95.45%	44

Table 4.4: Effect of VST-CA on Travel Cost in Synthetic Networks. Intervals of WARL where the largest cost reduction was reached are bold. Column **"Reduction"** is the sum of "> 10%", "5% – 10%" and "< 5%" columns.

Parameter	**Reduction of Travel Cost**					**# Samples**
	> 10%	5% – 10%	< 5%	Reduction	No Reduction	
# Samples	132	89	502	723	672	1395
POIs dist.						
Uniform	10.00%	8.43%	36.14%	54.57%	45.43%	697
Zipfian	8.86%	4.29%	35.57%	48.72%	51.28%	698
α						
0.15	6.04%	7.92%	52.92%	66.88%	33.12%	475
0.50	10.68%	5.23%	33.64%	49.55%	50.45%	440
1.00	11.67%	5.83%	20.83%	38.33%	61.67%	480
User density						
(0.00, 0.02]	15.00%	5.00%	1.67%	21.67%	78.33%	60
(0.02, 0.04]	24.44%	5.00%	8.89%	38.33%	61.67%	180
(0.04, 0.08]	15.00%	8.13%	27.50%	50.63%	49.37%	319
(0.08, 0.16]	3.24%	6.18%	47.65%	57.07%	42.93%	338
(0.16, 0.32]	3.75%	3.75%	45.94%	53.44%	46.56%	318
(0.32, 0.64]	4.44%	10.00%	48.89%	63.33%	36.67%	180
POI density						
(0.00, 0.01]	16.72%	7.03%	20.00%	43.75%	56.25%	640
(0.01, 0.02]	4.47%	6.05%	40.26%	50.78%	49.22%	377
(0.02, 0.03]	0.00%	0.00%	100.00%	100.00%	0.00%	20
(0.03, 0.04]	2.22%	5.83%	55.83%	63.88%	36.12%	358
WARL						
[0.00, 0.06]	1.29%	5.58%	71.24%	78.11%	21.89%	464
(0.06, 0.10]	7.82%	8.94%	30.73%	47.49%	52.51%	355
(0.10, 0.14]	**20.09%**	6.62%	12.82%	39.53%	60.47%	468
(0.14, 0.18]	7.14%	0.00%	0.00%	7.14%	92.86%	28
(0.18, 0.22]	2.50%	0.00%	0.00%	2.50%	97.50%	80

4.5 Summary

In this chapter, we show the scenario where activity-based ride-sharing becomes the source of congestion. Our non-congestion-aware solution VST-RS, introduced in our previous work [44], is extended to deal with congestion by an iterative method inspired by work in traffic assignment. Our congestion-aware approach, VST-CA, is able to reduce travel cost resulting from congestion caused by VST-RS, in synthetic and real networks such as Manhattan, Melbourne and Quito. Reduction is significant when the level of congestion is within a specific interval.

The processing time of VST-CA is fast enough to be deployable in a real world scenario. This is because VST-CA is based on VST-RS to which shortest-paths pre-processing is not needed. This is paramount for VST-CA as it is an iterative approach that updates the road segments costs in each iteration.

Chapter 5

Adopting Activity-Based Ride-Sharing in Crowdsourced Delivery

In this chapter, we study crowdsourced delivery, a similar transportation model in sharing economy. Now, instead of transporting people as in ride-sharing, private drivers transport products to online retailers' customers. Crowdsourced delivery is a trend among large retail companies, which are competing to gain more of the online market share. These companies recognize that order delivery plays a paramount role. For example, Walmart has bet on crowdsourced delivery where private drivers earn money by picking up and delivering parcels. For each order, the retailer's platform chooses the pick-up store at a time where no knowledge about drivers' journeys is available. In this chapter, we exploit this assumption and propose CD-CRSS, a new model where the pick-up store for each order is chosen along with the computation of the optimal delivery route. CD-CRSS generalizes the current crowdsourced delivery model as activity-based ride-sharing does with trip-based ride-sharing. That is, ad hoc drivers have a set of possible pick-up locations for each order instead of a fixed one. This seemingly subtle change guarantees solutions that are up to 20% *less costly than solutions to the current crowdsourced delivery model where stores and delivery routes are optimized independently. We also show that adopting a retailer-free strategy enables CD-CRSS to minimize even further, up to* 50%*. Given that an overall solution is a set of delivery routes where stores must be visited before customers, we aim to solve multiple instances—one for each delivery route—of a new graph-theoretic problem we call* Group Hamiltonian Path with Precedence Constraints*. We tackle instances of our novel model with $dist_{ra}$–BnB, our two-stage approach that performs (a) approximate assignment, and (b) exact routing. In the assignment stage, we present the* shortest route-adaptive distance*, our novel proximity measure on graphs. This measure is more effective than the shortest distance between a vertex and a path. In the routing stage, we solve instances of our new graph problem. We present our custom branch-and-bound technique that combines upper and lower bounds for an efficient pruning. As a result, our approach is readily deployable as we solve 2048-request instances of our model in 100 seconds in a large city, such as Melbourne, where merely* $\approx$ 40 *requests are realistically submitted in the same time interval.*

5.1 Overview

CROWDSOURCED delivery relies on private drivers who we call ad hoc drivers since their journeys are already taking place independent of the online shopping requests at a given time. Ad hoc drivers may have spare capacity in their vehicles so they can pick up goods from a specific retailer store and deliver them to customers. Thus, both retailer and ad hoc drivers benefit because retailers separate themselves from the complexities of delivery logistics and ad hoc drivers earn extra money. Walmart tested a similar model in 2013, by asking its in-store customers to deliver parcels to its online customers. In the same year, DHL launched MyWays, a platform that allowed registered drivers to deliver parcels along their daily routes in exchange of a fee [19]. Nowadays, Walmart offers a service called Spark Delivery in the US [71], which is similar to MyWays, except that drivers do not provide their routes upfront.

In current crowdsourced delivery models, retailers' platforms choose the pick-up store for each order at a time when drivers' routes are not provided [72]. Then, when orders come, a dispatcher automatically assigns them to ad hoc drivers based on custom business rules, e.g., an order is assigned to a driver who is within certain distance from the already chosen pick-up store [11]. The pick-up store for each order is fixed at the time when delivery routes are computed. This leads to suboptimal delivery routes since it dismisses other pick-up stores that may be closer to the original driver's journey.

Therefore, to be more effective, we propose our model called Crowdsourced Delivery through Concurrent Routing and pick-up Store Selection (CD-CRSS). *CD-CRSS generalizes the current crowdsourced delivery model as activity-based ride-sharing does with trip-based ride-sharing.* In our model, we leave the pick-up location (resp. the destination in activity-based ride-sharing) open. That is, the solution space when computing the optimal delivery route is expanded because the ad hoc drivers have a set of possible pick-up locations for each order instead of a fixed one. Instances of CD-CRSS are guaranteed to have cheaper solutions than instances of the current model because the current model's solution space is part of the larger CD-CRSS' search space.

In CD-CRSS, ad hoc drivers inform their journeys to the retailer's platform. From hereon, we refer to each driver k's original journey as k^+k^- route since the driver starts at

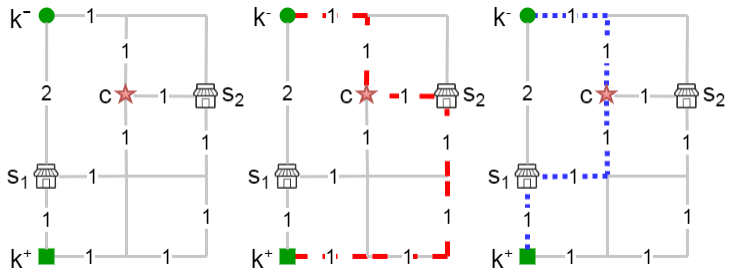

Figure 5.1: **Left.** Customer c placed an order to a retailer whose stores are s_1 and s_2. An ad hoc driver starts from k^+ toward her home k^-. **Center.** Red dashed line would be the driver's delivery route if retailer chooses s_2 as the pick-up store regardless of the driver's k^+k^- route. The delivery route's cost is 7. **Right.** Blue dotted line is the delivery route if the store is the one that minimizes the delivery route's cost by considering driver's k^+k^- route. The delivery route's cost is 5.

k^+ toward k^-. Then, *when orders come, the pick-up store for each order is the store that optimizes the delivery route based on ad hoc drivers' k^+k^- routes*. The retailer might even be able to predict drivers' k^+k^- routes based on their historical behavior.

Figure 5.1 depicts an example where a customer c purchased from a retailer that owns stores s_1 and s_2. An ad hoc driver travels from k^+ to k^-. In the current model, the retailer's platform chooses the pick-up store regardless of driver's k^+k^- route. In this example, the retailer has chosen s_2 for c's order. Thus, the solution space includes s_2 only. When an ad hoc driver is available and within the specified distance from store s_2, the dispatcher automatically assigns the order to this driver. In this case, the driver's *delivery route*—not the k^+k^- route—would be the red dashed line shown in Figure 5.1 (center). On the other hand, our algorithm would have considered driver's k^+k^- route and probed both s_1 and s_2, i.e., larger solution space. Thus, current model's solution is included. In our case, the chosen store would be s_1, and the driver's delivery route would be the cheaper blue dotted line shown in Figure 5.1 (right).

CD-CRSS has the following assumptions: (1) retailers have several stores, (2) products can be prepared at any store and this does not yield extra costs, or the supply chain in the retailer is adjusted to meet these demands, and (3) goods are small enough to fit in a passenger vehicle, which avoids the inclusion of capacity-constraints.

Current crowdsourced delivery is similar to the Pickup and Delivery Problem (PDP) [8]. In PDP, a transport request consists of picking up goods at one location and deliver-

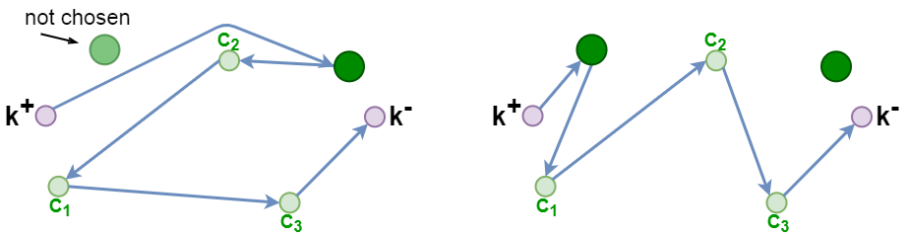

Figure 5.2: A driver is going from k^+ to k^-. **Left.** Hamiltonian path with precedence constraints: driver visits all the customers and the store already chosen. **Right.** Group Hamiltonian path with precedence constraints: driver visits all her assigned customers and the store that optimizes the delivery route chosen from a "group" of stores.

ing them to another. In a feasible route, a pick-up location must precede its corresponding delivery location—store precedes customers. A solution to the PDP is a set of routes to serve all requests with minimum total cost.

In CD-CRSS, instead of having only one fixed pick-up location per request, an ad hoc driver can pick up the products from any store of a retailer—the one that optimizes the delivery route. Ad hoc drivers inform their origins and destinations—or they might be predicted. The aim is again to minimize the total cost.

In current crowdsourced delivery, each driver's delivery route is a solution to the Hamiltonian Path with Precedence Constraints Problem (HPPCP). In CD-CRSS, each driver's delivery route is represented as a new Hamiltonian path problem. We call this problem *Group Hamiltonian Path with Precedence Constraints Problem (GHPPCP)*. In HPPCP, the path goes through all the customers within the assignment and the stores already chosen, whereas in GHPPCP, a path goes through all customers, yet there is a "group" of stores to choose from. See Figure 5.2.

Our min-cost GHPPCP is NP-hard because its special case, when the cardinality of the groups[1] is 1, is the min-cost HPPCP, which is NP-hard [23]. When the cardinality of the groups is > 1, as many instances of the HPPCP as combinations of the elements of the groups must be solved.

Solving an instance of the GHPPCP corresponds to computing the optimal route of *only one* ad hoc driver. In CD-CRSS, there are generally more than one ad hoc driver. Finding the optimal routes for all the ad hoc drivers involves "dividing" the set of cus-

[1]There may be more than one group per GHPPCP instance.

tomers first, and then assigning each subset to the best driver. Assignment and routing are inter-dependent and thus, optimal assignment is NP-hard as well.

An intuitive strategy to solve an instance of CD-CRSS would start by assigning customers to their "closest" ad hoc drivers and then, computing the delivery route for each assignment. In this strategy, the distance between a customer and a driver is the shortest distance between the customer and any vertex in the driver's k^+k^- route. We show experimentally that assignments using this distance yield more expensive routes.

Our strategy to solve an instance of CD-CRSS is $dist_{ra}$–BnB, a two-stage approach that performs (a) approximate assignment, and (b) exact routing. In the assignment stage, we present the *shortest route-adaptive distance*, our novel proximity measure on graphs. It measures the total cost of the driver's delivery route assuming only one customer at a time. We develop a heuristic to update this distance as more information about this delivery route becomes available. The shortest route-adaptive distance can be enhanced through different heuristics. To avoid that some ad hoc drivers end up with many customers, we balance the load of our initial assignments. To solve this load-balancing problem efficiently, we present an approximate approach.

In the routing stage, where we solve multiple instances of our new GHPPCP, our specialized branch-and-bound—BnB—technique combines upper and lower bounds for efficient computation of optimal delivery routes.

As an extension, we show how to deal with large detours. Two spatial partitioning methods are presented. They offer gradually stronger max-detour-cost guarantees to the ad hoc drivers if performed *before assignment*. If spatial partitioning is performed, dedicated drivers may be outsourced to a fleet company to account for possible unattended customers.

CD-CRSS can be adopted by either a retailer or an intermediate company that delivers for more than one retailer. In fact, in this chapter, we present it for the more generic case, that is, for more than one retailer. We also show that when the service makes the retailer selection transparent, i.e., Amazon-like, the cost can be further reduced up to 50% compared with the current model.

In summary, our contributions are:

- A novel crowdsourced delivery model that optimizes both pick-up selection and delivery routes concurrently. Travel plans in this new model are guaranteed to be cheaper than current model's plans. We solve instances of this model with $dist_{ra}$–BnB, a two-stage approach that performs (a) near-optimal assignments, and then (b) optimal routing.
- To compute assignments, we use our novel *Shortest Route-Adaptive Distance*, a proximity measure on graphs that outperforms the shortest distance measure. Load balancing may be needed on initial assignments. We also propose an approximate algorithm for load balancing.
- To compute optimal delivery routes, we solve instances of our *Group Hamiltonian Path with Precedence Constraints Problem*, a NP-hard graph problem. Our exact routing algorithm uses a branch-and-bound technique with combined upper and lower bounds for more effective pruning.

5.2 Problem Definition

In the classical Hamiltonian path problem on graphs, one must find whether there exists at least a path between a source and destination vertices that visits the rest of the vertices only once. Finding a delivery route for an ad hoc driver is equivalent to computing a Hamiltonian path. Yet, this is a special version called Hamiltonian Path with Precedence Constraints Problem as a store must *precede*—be visited before—its corresponding customer. In our model, the store for each order has not been chosen yet, thus the delivery route is a path that visits the stores that minimize the route cost and still visits all the customers once. We call this novel problem Group Hamiltonian Path with Precedence Constraints Problem (GHPPCP) since the optimal store for each order is chosen from a *group* of stores.

In large-scale instances, there are thousands of online customers who submit their requests to different retailers within a time interval. We assume that each request consists of products from one retailer only. These delivery tasks must be assigned among the available ad hoc drivers. After assignment, the delivery route for each driver is in

turn computed. Therefore, the whole problem consists of assignment and then solving as many GHPPCP instances as ad hoc drivers there are.

Let H be the set of ad hoc drivers. Let k^+ and k^- be the start and end locations of the ad hoc driver $k \in H$. Each retailer $r \in R$ owns a set of stores S_r. We can build a directed graph $G \coloneqq \langle V, A \rangle$ whose vertices V correspond to the locations of stores and customers as well as ad hoc drivers' start and end locations. Its set of arcs A satisfy a priori constraints of the problem, e.g., ad hoc drivers cannot go to any online customer when they start their journey as they must visit a store first. For each arc $(i, j) \in A$, $c_{i,j}$ and $t_{i,j}$ are the cost—distance—and time to travel from i to j. Therefore, our whole problem is defined as:

Given: $G \coloneqq \langle V, A \rangle$, $c_{i,j}$ and $t_{i,j}$ for all $(i, j) \in A$.

Find: Set of delivery routes $P \coloneqq \{P_k \mid P_k \coloneqq \langle V_k, A_k \rangle, V_k \subseteq V, A_k \subseteq A\}$, each associated with an ad hoc driver $k \in H$, such that the cost $\sum_{k \in H} \sum_{(i,j) \in A_k} c_{i,j}$ is minimum.

Moreover, for each P_k, the following must be satisfied:

1. P_k starts at k^+.

2. $|V_k \cap S_r| \leq 1$ for all $r \in R$. That is, products for a customer are picked up from only one store of a retailer, yet an ad hoc driver can visit more than one retailer since she may serve more customers. There may be drivers who are assigned no customers whatsoever.

3. Let s_q be the chosen pick-up store for the online customer q. Let B_i^k be the time at which driver k starts service at location i. Thus, $B_q^k \geq B_{s_q}^k + t_{s_q,q}$, that is, the store must be visited before the customer—*precedence constraint*.

4. P_k ends at k^-.

5.3 Our Solution

Multi-vehicle routing problems can be approached as two-stage problems. In the first stage, we assign a subset of customers to each driver and, in the second stage, we compute the driver's delivery route to serve her subset of assigned customers. An optimal assignment can be reduced to the Set Cover problem if we assume that the route cost of each assignment in the powerset of assignments is provided by an oracle. The Set Cover problem is NP-hard and therefore, we can only aim to solve CD-CRSS instances approximately. Besides our approach, three baseline approaches are presented. Moreover, as benchmark, an off-the-shelf solver is used to compute exact solutions to the Mixed-Integer Linear Program (MILP) formulation of CD-CRSS.

5.3.1 Overview of Our Solution

We call our two-stage approach: $dist_{ra}$–BnB.

Assignment. Our intuition is that a good assignment should contain the customers located close enough to the driver's k^+k^- route. We present our novel *Shortest Route-Adaptive distance*, which is a new way of measuring the proximity of a customer to a path. It considers the delivery route the driver would follow to serve a customer. We show experimentally that assignments using our shortest route-adaptive distance yields much cheaper routes than using simple shortest distance from the customers to the drivers' k^+k^- routes.

The initial assignment may ask some drivers to deliver orders to a considerable number of customers. Since, in the second stage, we compute optimal routes by using our specialized branch-and-bound (BnB) technique, the route's computation time may be prohibitive for big-sized assignments. Therefore, we balance the load of our initial assignment. We show experimentally that assignments of at most 8 customers are enough to keep comparable costs as increasing the size of the assignments yields only marginal cost reductions.

Routing. We compute an optimal route for each assignment and select the optimal store for each order within the assignment, by using our BnB technique. In each assign-

ment, the driver can have more than one pick-up store of the same retailer for each order. This means that a single BnB tree is not enough for the routing of each assignment. In fact, for each assignment, there will be as many BnB trees as combinations of stores—one per retailer—exist. To speed up the computation, we combine upper and lower bounds to prune the BnB trees more effectively.

Next, we explain each stage in detail, yet we do so in a bottom-up manner. We start with the routing stage as this corresponds to each ad hoc driver. From there, we continue with the assignment stage as this is focused on solving the higher-level Set Cover problem.

5.3.2 Routing

Routing an assignment is the second stage and is equivalent to computing the group Hamiltonian path with precedence constraints. Since it is enough for a driver to visit one store per retailer as she could pick up all customers' orders of the same retailer, *the Hamiltonian path must visit all the customers but not all the stores. This is what makes GHPPCP different from the classical Hamiltonian path with precedence constraints.* In $dist_{ra}$–BnB, we opted for optimal routing. We also present a *nearest-neighbor* routing scheme as a baseline.

Nearest-Neighbor (NN) Routing

The route for an assignment begins with the ad hoc driver's start location and from there, the next nearest feasible destination is computed and appended to the route. In our problem, the next feasible destination will be a store. Then, the next feasible destination can be either another store from a different retailer or a customer of the already visited store. We keep doing this until the destination is reached. We also use the cost of this route as an upper bound in our BnB scheme detailed in the next section.

Branch-and-Bound (BnB) Routing

We compute the optimal route for each assignment by using branch-and-bound (BnB). To find an optimal routing for an assignment, we must explore the different combinations

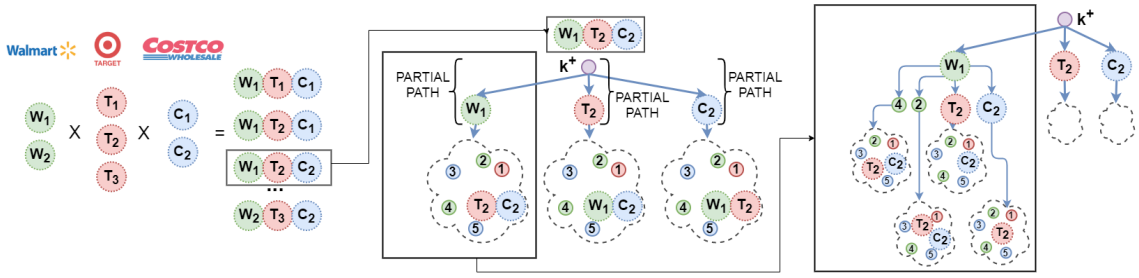

Figure 5.3: **Left.** Combinations of stores of three different retailers. **Center.** Three initial partial paths in a BnB tree for a specific combination of stores. Each path starts at the driver's origin. A BnB tree can be branched using the other stores in the combination and the customers in the assignment. **Right.** The partial path with minimum lower bound over all the partial paths of all the BnB trees is branched. Note that the paths are feasible, i.e., driver cannot visit other retailer's customers without visiting one of their stores first.

of stores, where each combination consists of one store per retailer. This is because, it is enough for a driver to visit one store per retailer to pick up all customers' orders of the same retailer. See Figure 5.3 (left). Then, a BnB tree is computed for each combination of stores. See Figure 5.3 (center).

We can think of a BnB tree as a set of partial paths, all originating from the BnB tree's root—ad hoc driver's start location. For each BnB tree, we first create as many initial partial paths as there are stores in the combination. An initial partial path corresponds to the small path from the origin to the store. See Figure 5.3 (center). Each partial path has associated lower and upper bounds of the cost of it becoming a full path. A partial path is branched if its lower bound is the minimum *over all the partial paths of all the BnB trees*. See Figure 5.3 (right).

To become a full path, a partial path must visit all the customers in the assignment and the end location k^-. Let $Z \coloneqq \overline{S} \cup \overline{C} \cup \{k^-\}$, where $\overline{S}$ is the set of nonvisited stores within the combination and $\overline{C}$ is the set of nonvisited customers in the assignment. Let n be the last already visited vertex in the partial path. In Figure 5.3 (center), for the leftmost partial path $\rho = (k^+, W_1)$, we have $\overline{S} = \{T_2, C_2\}$, $\overline{C} = \{1,2,3,4,5\}$, and $n = W_1$. We can formally define the cost lower bound $\mathcal{L}^{\rho}_{cost}$ associated with partial path ρ as:

$$\mathcal{L}^{\rho}_{cost} \coloneqq c(\rho) + \min_{v_i \in f(n,Z,\rho)} dist(n, v_i) + \mathcal{L}^{Z}_{r} + \mathcal{L}^{Z}_{c} \tag{5.1}$$

where $c(\rho)$ is the distance of ρ, $f(n, Z, \rho)$ is a function that returns the *set of next feasible vertices in Z departing from n and considering the already visited locations in* ρ, $dist(n, v_i)$ is the shortest distance between n and v_i, and $\mathcal{L}_r^Z$ and $\mathcal{L}_c^Z$ are the row- and column-wise lower bounds, respectively. This notion of row and column comes from the matrix of shortest distances $\mathcal{D}^Z$ between all pairs of elements in Z. For the partial path $\rho = (k^+, W_1)$ and $Z = \overline{S} \cup \overline{C} \cup \{k^-\} = \{T_2, C_2, 1, 2, 3, 4, 5, k^-\}$, $f(n, Z, \rho) = \{T_2, C_2, 2, 4\}$. Customers $\{1, 3, 5\}$ cannot be visited as their corresponding stores are not in ρ—not visited. $\mathcal{L}_r^Z$ is computed as follows:

$$\mathcal{L}_r^Z \coloneqq \sum_{v_i \in Z} \min_{v_j \in g(v_i, Z)} \mathcal{D}_{v_i, v_j}^Z \tag{5.2}$$

where $g(v_i, Z)$ is a function that returns the *set of next feasible vertices in Z departing from* v_i. For the same Z and $v_i = T_2$, $g(v_i, Z) = \{C_2, 1, 2, 3, 4, 5\}$. Customers $\{3, 5\}$ are returned as feasible since C_2 may have been visited. The order in which elements in Z are visited is unknown. For $v_i = 1$, $g(v_i, Z) = \{C_2, 2, 3, 4, 5, k^-\}$. k^- is feasible since customer 1 has been visited. The ad hoc driver cannot go to T_2 since this store must have been visited before 1. For $v_i = k^-$, $g(v_i, Z) = \varnothing$.

After this, we update $\mathcal{D}^Z$ as follows:

$$\mathcal{D}_{v_i, v_j}^Z \leftarrow \mathcal{D}_{v_i, v_j}^Z - \min_{v_k \in g(v_i, Z)} \mathcal{D}_{v_i, v_k}^Z \tag{5.3}$$

Finally, we are able to compute $\mathcal{L}_c^Z$ as follows:

$$\mathcal{L}_c^Z \coloneqq \sum_{v_i \in Z} \min_{v_j \in h(v_i, Z)} \mathcal{D}_{v_i, v_j}^Z \tag{5.4}$$

where $h(v_i, Z)$ is a function that returns the *set of previous feasible vertices to* v_i *in Z*. For the same Z and $v_i = T_2$, $h(v_i, Z) = \{C_2, 2, 3, 4, 5\}$. For $v_i = 1$, $h(v_i, Z) = \{T_2, C_2, 2, 3, 4, 5\}$. For $v_i = k^-$, $h(v_i, Z) = \{1, 2, 3, 4, 5\}$.

On the other hand, the cost upper bound $\mathcal{U}_{cost}^{\rho}$ associated with partial path ρ corresponds to the already traveled distance $c(\rho)$ and the distance $c(\hat{\rho})$ of the path $\hat{\rho}$ constructed by appending the next feasible nearest neighbor repeatedly. Therefore, $\mathcal{U}_{cost}^{\rho}$ is

defined as:

$$\mathcal{U}^{\rho}_{cost} \coloneqq c(\rho) + c(\hat{\rho}) \tag{5.5}$$

where path $\hat{\rho}$ can be defined recursively as:

$$\hat{\rho}_{i+1} \coloneqq \begin{cases} (n) & i = 0 \\ \hat{\rho}_i \bullet \left(\underset{v_j \in f(v_i, Z, \hat{\rho}_i)}{\arg\min}\ dist(v_i, v_j) \right) & 0 < i < |Z| - |\rho| \end{cases} \tag{5.6}$$

where $\bullet$ is the sequence concatenation operator, and the base case is $\hat{\rho}_1 \coloneqq (n)$ because $\hat{\rho}$ is constructed starting from the last already visited vertex n.

We explain our BnB routing algorithm in Section 5.3.4 since spatial partitioning must be reviewed first.

5.3.3 Assignment

The assignment of customers to ad hoc drivers is the first stage of our approach. Let us assume that an oracle provides the costs of the delivery routes of every possible assignment. Therefore, the assignment problem reduces to the Set Cover problem. We do not aim to solve instances of CD-CRSS to optimality and thus, we resort to a heuristic-based assignment. Our intuition is that a good assignment should contain the customers located close enough to the driver's k^+k^- route. Here, the concept of proximity between a vertex—customer—and a path arises. On one hand, we can choose the shortest distance between a customer and a driver's k^+k^- route as a plausible proximity measure. On the other hand, we can opt for being delivery-route-aware, that is, we consider as a proximity measure the cost of the route to deliver to this customer. Both assignment criteria are explained in this section.

Path-Generator Voronoi Assignment

Choosing the shortest distance between a customer and a driver's k^+k^- route as proximity measure is equivalent to compute the path-generator Voronoi diagram, where each cell

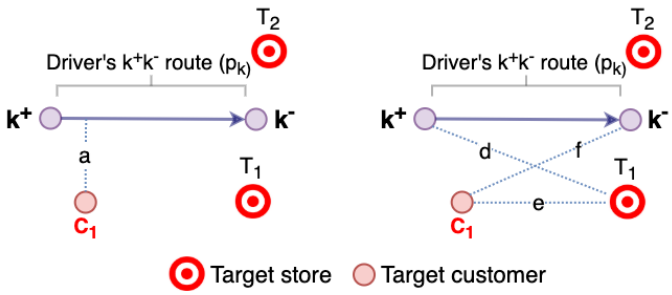

Figure 5.4: Proximity measures between a customer and a driver's path. **Left.** Shortest distance $dist(c_1, p_k) = a$. It is used in path-generator Voronoi-based assignments. **Right.** Shortest route-adaptive distance $dist_{ra}(c_1, p_k) = d + e + f$.

has a path k^+k^- as generator. Path-generator Voronoi diagrams are a generalization of the well-known vertex-generator Voronoi diagrams. In the latter, a subset of vertices are the generators of the cells such that each vertex within a cell has the generator of such cell as its closest one. In the path-generator counterpart, the generators are paths, thus a vertex within a cell has this cell's generator as its closest path. The distance between a vertex and a path is the shortest distance from this vertex to any of the vertices that are part of the path. In Figure 5.4 (left), p_k is the closest path to c_1 according to the shortest distance measure $dist(c_1, p_k)$. The ad hoc driver's k^+k^- route is the generator and the customers within the cell are assigned to her. Also, the stores within the cell would be the ones used in the routing stage.

Shortest Route-Adaptive Distance-Based Assignment

The path-generator Voronoi assignment is intuitive, yet it overlooks store locations. To assign a customer to a driver, it is not enough the customer be close to the ad hoc driver's k^+k^- route since her matching stores may be located in such a way that serving her is expensive for the driver. For example, the stores may be located nearby the driver's destination, whereas the customer may be close to the starting point. See Figure 5.4. Since the driver must visit one of the stores first, the route's cost may be as much as three times the original p_k. A better way to decide whether a customer should be assigned to a driver is by computing the shortest route-adaptive distance. This novel proximity measure between a customer and a driver's k^+k^- route is the min-cost delivery route of this

driver over all matching pick-up stores for this customer. In Figure 5.4 (right), the shortest route-adaptive distance between customer c_1 and path p_k is $dist_{ra}(c_1, p_k) = d + e + f$. This is because, among all matching stores $\{T_1, T_2\}$, store T_1 is the one which minimizes the delivery route distance.

Formally, let k^+ and k^- be the start and end points of the ad hoc driver k. Let p_k be this driver's k^+k^- route. Let S_q be the set of matching stores for customer q. Let $dist(i, j)$ be the shortest distance between vertices i and j. Thus, the shortest route-adaptive distance from q to p_k is defined as:

$$dist_{ra}(q, p_k) \coloneqq \min_{s \in S_q} \{dist(k^+, s) + dist(s, q)\} + dist(q, k^-) \tag{5.7}$$

Initial Assignment. Once the shortest route-adaptive distance is computed between all pairs of customers and drivers, we can build the complete weighted bipartite graph where customers and drivers are the two parts of the graph and each edge cost is the shortest route-adaptive distance. We create artificial non-cost edges between the ad hoc drivers' vertices. Then, *finding a greedy initial assignment is equivalent to compute the Min-cost Spanning Tree (MST) on this graph*. These artificial edges are guaranteed to be in the MST as they have cost zero which, in turn, correctly forces to assign only one customer to a driver since no cycles can be present in a tree.

Load Balancing - Adapting the Distance. This initial assignment may overload some drivers which leads to prohibitive computation time in the routing stage where we compute optimal routes. The fastest computation would be on fully-balanced assignments as every driver would be assigned the minimum number of customers. Yet, the resulting routing from such an assignment is not necessarily the cheapest. Hence, the goal is to find large enough customer assignments that can be routed fast and that keep quality. That is, we want to minimize the maximum assignment size while keeping the total routing cost to the minimum.

Up to this point, we have a MST where some of the inner vertices—drivers—may have high degree—overloaded. Our next task is to minimize the highest degree down to a lower degree m while keeping the cost to the minimum. We show experimentally that $m >$ 8 yield marginal cost reductions at the expense of exponentially growing computation

times.

Before starting the degree minimization on the MST—reassignment—we should *adapt* the shortest route-adaptive distance between some drivers' k^+k^- routes and their assigned customers. To do this, we use the information about the store that minimized the route-adaptive distance for each customer in the previous iteration. If two or more customers assigned to the same driver share the same "best" store—the one which minimized the route-adaptive distance—the distance from the driver's start location to such store can be divided evenly across such customers and thus, the shortest route-adaptive distance of all these customers is reduced. By doing so, we are favoring assignments where there are more than one customer who share the same store, which in turn leads to cheaper routes. We say "favoring" because this update process may be carried out multiple times since the MST is modified as a result of its degree minimization. Therefore, assignments change at each iteration, yet we want to preserve the good ones. See example in Figure 5.5.

Formally, let $C_k^{s^*}$ be the set of customers assigned to driver k who share the best store s^*. Therefore, the shortest route-adaptive distance for $q \in C_k^{s^*}$ is updated as follows:

$$dist_{ra}(q, p_k) \leftarrow \frac{dist(k^+, s^*)}{|C_k^{s^*}|} + dist(s^*, q) + dist(q, k^-) \tag{5.8}$$

To minimize the degree of the MST, we adopt a greedy technique that does local moves from high-degree to low-degree vertices—reassignments between drivers. From the edges in the MST, adjacent to the highest-degree vertex—driver—we remove the most expensive. This leaves a customer orphan, i.e., without assignment. From the edges in the graph, adjacent to this orphan customer and different inner vertex—another driver—we append the least expensive to the MST. This is a "local move". Specifically, in our Algorithm 7, from the MST edges incident to the driver v with the highest-degree Δ, we pick the edge (v, ϕ) which has the largest cost. The other endpoint of this edge is the customer ϕ. If there are drivers $\mathcal{K}$ that share ϕ and whose graph degree is at most $\Delta - 2$, then we "move" ϕ to the driver v' whose shortest route-adaptive distance to ϕ is the smallest. If there are no drivers $\mathcal{K}$, we pick the edge with the next largest cost and repeat the process with the new customer—inner `while` loop, lines 17-32. Then, we

Algorithm 7 Load balancing: minimize degree m of MST.

function MINDEG(G, T, m) ▷ G: graph, T: MST, m: bound
  $\mathcal{S} \leftarrow \varnothing$ ▷ $\mathcal{S}$: drivers on standby
  $mov \leftarrow 0$ ▷ mov: local moves
  while True **do**
    $\mathcal{D} \leftarrow$ DEGREES($T, \mathcal{S}$) ▷ $\mathcal{D}$: degrees non-standby drivers
    $\Delta \leftarrow \max\{\delta \mid \langle k, \delta\rangle \in \mathcal{D}\}$ ▷ Δ: highest degree
    if $\Delta = m$ **then** ▷ requested bound degree reached
      break
    if $\Delta = 0$ **or** ($\Delta = 1$ **and** $mov = 0$) **then**
      break ▷ lowest degree reached + **no** moves
    if $\Delta = 1$ **then** ▷ lowest degree reached + moves
      $\mathcal{S} \leftarrow \varnothing$ ▷ standby drivers are freed to try again
      $mov \leftarrow 0$
      continue
    $v \leftarrow k \in \{k \mid \langle k, \delta\rangle \in \mathcal{D} \wedge \delta = \Delta\}$ ▷ v: driver with Δ
    $\gamma \leftarrow \infty$ ▷ γ: upper bound of edge cost
    while True **do**
      $\mathcal{Q} \leftarrow \{\langle q, c_{v,q}\rangle \mid (v, q) \in E(T) \wedge c_{v,q} < \gamma\}$ ▷ $\mathcal{Q}$: customers assigned to driver v whose edge cost < γ
      if $\mathcal{Q} = \varnothing$ **then** ▷ no edges w. less cost than γ
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{v\}$ ▷ put v on standby
        break
      $c_{v,\phi} \leftarrow \max\{c_{v,q} \mid \langle q, c_{v,q}\rangle \in \mathcal{Q}\}$ ▷ max. cost
      $\mathcal{K} \leftarrow \{k \mid (k, \phi) \in E(G) \wedge (k, \phi) \notin E(T) \wedge k \notin \mathcal{S} \wedge \exists\langle k, \delta\rangle \in \mathcal{D} : \delta \leq \Delta - 2\}$ ▷ non-standby drivers who can serve ϕ and have degree $\leq \Delta - 2$
      if $\mathcal{K} \neq \varnothing$ **then** ▷ choose closest driver to ϕ
        $c_{v',\phi} \leftarrow \min\{c_{k,\phi} \mid k \in \mathcal{K} \wedge (k, \phi) \in E(G)\}$
        $E(T) \leftarrow E(T) \setminus \{(v, \phi)\}$ ▷ v does not serve ϕ
        $E(T) \leftarrow E(T) \cup \{(v', \phi)\}$ ▷ v' now serves ϕ
        UPDATE-DIST-RA(T)
        $mov \leftarrow mov + 1$
        break
      else
        $\gamma \leftarrow c_{v,\phi}$ ▷ new u.b. to try w. next costly cx.
  return T ▷ T: min bounded degree spanning tree
end function

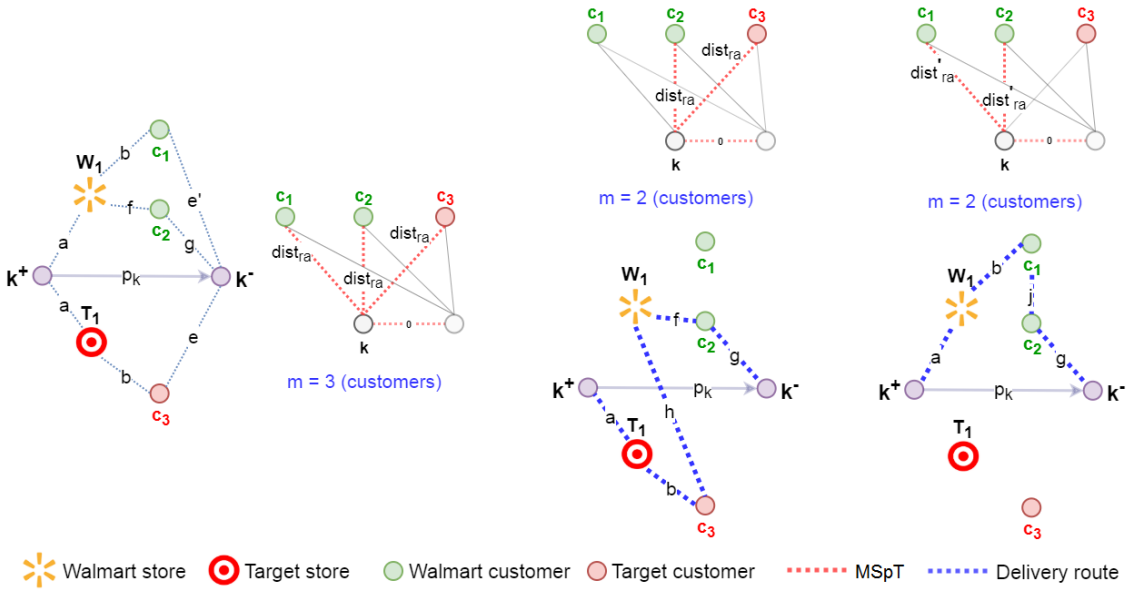

Figure 5.5: **Left.** The shortest route-adaptive distances between the customers and the driver's k^+k^- route are: $dist_{ra}(c_1, p_k) = a + b + e'$, $dist_{ra}(c_2, p_k) = a + f + g$ and $dist_{ra}(c_3, p_k) = a + b + e$. These customers are assigned to this driver based on the MST. The relation between these distances is: $dist_{ra}(c_1, p_k) > dist_{ra}(c_3, p_k) > dist_{ra}(c_2, p_k)$. **Center.** If the max. load is set to $m = 2$ and the shortest route-adaptive distances are NOT updated using the information from the previous assignment, then (c_1, k) would not be part of the new MST and the resulting delivery route's cost would be $a + b + h + f + g$. **Right.** If the shortest route-adaptive distances are updated knowing that W_1 is the best store for c_1 and c_2, then $dist_{ra}(c_1, p_k) \leftarrow \frac{a}{2} + b + e'$ and $dist_{ra}(c_2, p_k) \leftarrow \frac{a}{2} + f + g$. Since $e' - e < \frac{a}{2}$, the relation between distances shifts to: $dist_{ra}(c_3, p_k) > dist_{ra}(c_1, p_k) > dist_{ra}(c_2, p_k)$. When the MST is recomputed, (c_3, k) is not part of this new MST and the resulting cost would be the cheaper: $a + b + j + g$.

repeat the whole process with the new highest-degree driver—outer `while` loop. Every time we update the MST through local moves, the shortest route-adaptive distances for some vertices may be updated—line 28. The algorithm finishes when either $\Delta = m$ or $\Delta = 0$ or $\Delta = 1$ and no local moves were performed. The final assignment is given by the resulting MST.

We experimentally show that bounded assignments ($m = 8$) using our shortest route-adaptive distance result in much cheaper routes than path-generator Voronoi-based ones.

5.3.4 Extension: Space Partitioning - Dealing with Large Detours

Our assignment stage aims to serve all the customers, thus ad hoc drivers are likely to make large detours if the ratio of customers to drivers is large enough and even if the load is balanced. We give gradually stronger max-detour-cost guarantees to the ad hoc drivers *if we opt for partitioning the space before the assignment computation*. Our first partitioning scheme offers an upper bound which depends on the number of customers within the partition. Our second scheme is stronger and guarantees not exceeding a constant factor of the driver's k^+k^- route's distance. Once the partitioning per ad hoc driver is performed, the assignment is computed as before, yet it considers a smaller subgraph.

Path-Expansion Partitioning

The ad hoc drivers' k^+k^- routes are expanded from every vertex along them. In other words, *we are growing shortest-path trees from each vertex along the driver's k^+k^- route*. We show in Figure 5.6 the expansion of a driver's k^+k^- route up to half its distance. In this example, there is only one intermediate vertex i in k^+k^- route. After expanding the k^+k^- route from these three vertices using a Dijkstra-like technique, the resulting partition contains stores from two retailers, Walmart and Target, and their customers identified with similar color. In this figure, the blobs mimic the subsets of vertices whose distance is less or equal to half the k^+k^- route's distance. Let f be the fraction of the distance $dist_k$ of the k^+k^- route. Let $|R_k|$ and $|C_k|$ be the number of retailers and customers within the partition.

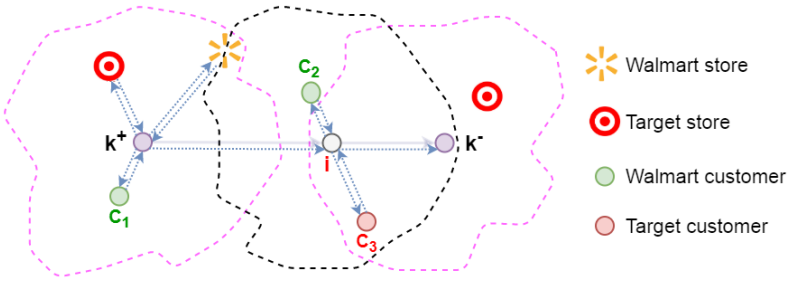

Figure 5.6: Driver's k^+k^- route contains vertex i. This route is expanded up to $f = 0.5$ its distance, from each of its vertices $\{k^+, i, k^-\}$. Walmart and Target stores, and some customers are within the expansion. If we assume that stores and customers are not connected directly, an optimal delivery route would follow this order: 1. k^+, 2-3. either Target or Walmart (RT), 4. C_1 (RT), 5. i, 6-7. either C_2 or C_3 (RT), 8. k^-. RT stands for round-trip.

Proposition 5.1. *The upper bound $\mathcal{U}_k$ of the cost of the driver's delivery route in this partitioning technique is given by:*

$$\mathcal{U}_k = \big(2f(|R_k| + |C_k|) + 1\big)dist_k \tag{5.9}$$

Proof. Let us assume there is no *direct* path between stores nor customers. Yet, we know they are connected *indirectly* through the vertices of the k^+k^- route. This is because the stores and customers were indeed found when the k^+k^- route was expanded from its vertices. If the driver follows her k^+k^- route and from each vertex along it she visits the corresponding store or customer—respecting precedence—we can conclude that the cost of visiting either a store or customer is at most twice—round-trip—f. Also, it is enough to visit one store of each retailer. Finally, we must count the k^+k^- route once as she travels across it. □

Cost-Threshold Partitioning

In our example in Figure 5.6 for an ad hoc driver k, where the expansion fraction $f = 0.5$, number of retailers $|R_k| = 2$, and number of customers $|C_k| = 3$, the guaranteed cost upper bound is $\mathcal{U}_k = 6 \times dist_k$. This is a loose upper bound.

A stronger upper bound is achieved by setting an ellipse-based cost-threshold. Applying ellipse-based techniques, we can find stores and customers within a cost-threshold.

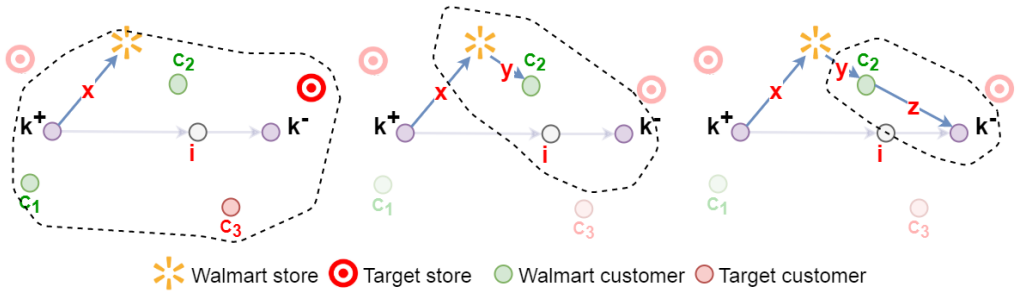

Figure 5.7: **Left.** An ellipse with threshold t is computed based on k^+ and k^-. Customers within it are not guaranteed to be visited after the driver visits Walmart. **Center.** In the next iteration, a new ellipse is computed with foci Walmart and k^-. The new threshold is $t - x$. **Right.** Ellipse has foci c_2 and k^-. Threshold is $t - x - y$. This is the last iteration.

The procedure is shown in Figure 5.7. In the figure, an ad hoc driver has a cost threshold t and the first step is to find, through an ellipse, which stores are reachable. In the interior of an ellipse, every point is located at a combined distance of no more than a constant from two fixed points called foci. In our case, the driver's origin and destination are the initial foci and the constant is the cost-threshold. All the vertices within the resulting ellipse are those within the threshold. This finds the first candidate stores. Customers within this first ellipse are not guaranteed to be reachable after visiting a store. Thus, after choosing a store, a new ellipse must be computed where the first focal point is this chosen store and the second is still the destination. The cost-threshold must be updated by subtracting the already traveled distance. This is done iteratively until the destination is the only vertex within the ellipse.

In the routing stage, we do not only aim to find the cheapest route but mainly to visit the most customers we can within the threshold. Thus, $\mathcal{L}^{\rho}_{cost}$, i.e., lower bound (5.1), is not useful anymore. Instead, we use the partial path distance as tiebreaker as there can be partial paths with the same maximum number of served customers. The partial paths must now be associated with an upper bound $\mathcal{U}^{\rho}_{cx}$ of the number of customers each may serve. Let C_ρ and $\overline{C_\rho}$ be the set of visited and nonvisited customers in the partial path ρ, respectively. Ellipse-based partitioning is performed whenever a new vertex is added to the partial path, i.e., branching stage. Thus, set $\overline{C_\rho}$ is updated with the customers who are within the new ellipse only. Upper bound $\mathcal{U}^{\rho}_{cx}$ is defined as:

$$\mathcal{U}_{cx}^{\rho} \coloneqq |C_{\rho}| + |\overline{C_{\rho}}| \tag{5.10}$$

Algorithm 8 is the BnB algorithm that computes the optimal delivery route. Assigned customers C_k are retrieved from π_k and a min-priority queue Q is used for either minimizing cost or maximizing number of served customers—when a cost threshold is provided. The BnB trees—one for each combination of stores in S_k—and their corresponding initial partial paths are created by calling the function INITIALPARTIALPATHS(). Each entry in Q corresponds to a partial path ρ where its priority is given by either the cost lower bound $\mathcal{L}_{cost}^{\rho}$ or the customers upper bound $\mathcal{U}_{cx}^{\rho}$—depends on whether a threshold is provided. The smallest cost upper bound u is maintained throughout the optimization such that partial paths whose lower bounds are greater than u are not branched anymore—line 37. When the partial path becomes full path, i.e., all the customers are served and the destination is in the path—line 19—it proceeds depending on whether a threshold is provided. In the case of threshold-based partitioning, what we found is the maximum number of customers who are going to be served—line 26—yet we still must search for the cheapest path among the ones which serve the same maximum number of customers—lines 28-29. In the case of path-expansion partitioning or no partitioning at all, this path is the solution—line 21. If the partial path is not full then, it is branched—line 31.

Dedicated Drivers

If space partitioning is performed, there may be customers who were either not matched with a store within their partition or were included in no partition from the beginning or were left outside later due to threshold. We apply a Traveling Salesman Problem (TSP) regime since precedence constraints are already satisfied. Dedicated drivers start from the store where they are needed, and they serve the nearest customers of this retailer only. We opt for an approximation algorithm for the TSP scheme. *Double-tree*[2] is a 2-approximation algorithm that can be used as triangle inequality holds in our graph. It consists on "shortcutting" the Euler tour computed when every edge of the MST that

[2]We are aware of the 1.5-approximation algorithm by Christofides [18], yet we decided to implement the double-tree algorithm for the sake of simplicity.

Algorithm 8 Branch and Bound algorithm for routing.

function BNB(k^+, k^-, π_k, t) ▷ π_k: assignment, t: threshold
  $S_k \leftarrow$ {stores in π_k}
  $C_k \leftarrow$ {customers in π_k}
  Create priority queue Q
  if t = 0 **then** ▷ path-expansion **or** no partitioning
    $I \leftarrow$ INITIALPARTIALPATHS(S_k, C_k, k^+, k^-)
    for each $\rho \in I$ **do**
      $Q.insert(\rho, \mathcal{L}^{\rho}_{cost})$ ▷ cost lower bound
  else ▷ cost-threshold partitioning
    $th \leftarrow dist(k^+, k^-) \times t$
    $I \leftarrow$ INITIALPARTIALPATHS(S_k, C_k, k^+, k^-, th)
    for each $\rho \in I$ **do**
      $Q.insert(\rho, -\mathcal{U}^{\rho}_{cx})$ ▷ # customers upper bound
  $\rho^* \leftarrow ()$ ▷ ρ^*: optimal path
  $\mathcal{U}^{max}_{cx} \leftarrow 0$ ▷ max. # customers
  $u \leftarrow \infty$ ▷ smallest cost upper bound so far
  while Q is not empty **do**
    $\rho, b \leftarrow Q.ExtractMin()$ ▷ $b : \mathcal{L}^{\rho}_{cost}$ or $-\mathcal{U}^{\rho}_{cx}$
    if $\overline{C_\rho} = \varnothing$ **and** $k^- \in \rho$ **then** ▷ all customers visited
      if t = 0 **then** ▷ path-expansion **or** no partitioning
        $\rho^* \leftarrow \rho$ ▷ ρ^*: min-cost path found!
        break
      else ▷ cost-threshold partitioning
        if $\mathcal{U}^{max}_{cx} = 0$ **then**
          $\rho^* \leftarrow \rho$ ▷ ρ^*: candidate path
          $\mathcal{U}^{max}_{cx} \leftarrow b$ ▷ max. # cx. found!
        else
          if $b = \mathcal{U}^{max}_{cx}$ **and** $c(\rho) < c(\rho^*)$ **then**
            $\rho^* \leftarrow \rho$ ▷ upgrade with cheaper
          continue
    $O \leftarrow$ BRANCH(ρ)
    if t = 0 **then** ▷ path-expansion **or** no partitioning
      for each $\rho' \in O$ **do**
        if $\mathcal{U}^{\rho'}_{cost} < u$ **then**
          $u \leftarrow \mathcal{U}^{\rho'}_{cost}$ ▷ update u
      for each $\rho' \in O$ **do**
        if $\mathcal{L}^{\rho'}_{cost} > u$ **then**
          continue ▷ prune branch
        $Q.insert(\rho', \mathcal{L}^{\rho'}_{cost})$
    else ▷ cost-threshold partitioning
      for each $\rho' \in O$ **do**
        if $\mathcal{U}^{\rho'}_{cx} \geq \mathcal{U}^{max}_{cx}$ **then**
          $Q.insert(\rho', -\mathcal{U}^{\rho'}_{cx})$
  return ρ^*
end function

spans the dedicated driver and the unattended customers is duplicated.

5.4 Experimental Evaluation

$dist_{ra}$–BnB and the baselines are evaluated on sample areas of $100km^2$ taken at random from the Melbourne road network. Road intersections and stores of the four most popular retailers are extracted from *OpenStreetMap* [52]. Customer locations are uniformly determined, and their distribution over retailers corresponds to the retailers' current market share. We set the default number of customers to 256, which is a rather pessimistic assumption given that 256 requests are submitted every 12 minutes in Melbourne [55] and yet, $dist_{ra}$–BnB can process 256 requests in $\approx$ 8 seconds. See Figure 5.9 (right). Ad hoc drivers' start locations are chosen to be Zipfian, whereas end locations are uniform. Zipfian distributed locations mimic spatial concentration. Hence, drivers start from crowded areas such as the CBD, where their workplaces are located, toward their homes, which can be located anywhere. The default ratio of customers to ad hoc drivers is 4.0. Larger ratios would miss the goal of employing ad hoc drivers since each driver would serve more than 4 customers on average and thus, they would resemble professional couriers. The parameter values and their defaults are presented in Table 5.1. We run 50 samples for each parameter value. The distribution of the results is shown through box plots.

In this section, we show (a) that aiming for assignments of at most 8 customers is good enough, (b) the extent of savings achieved by our CD-CRSS model compared with the current model, (c) the further savings of CD-CRSS if an Amazon-like model is adopted, (d) that, within CD-CRSS, our shortest route-adaptive distance-based assignments yield much cheaper routes than path-generator Voronoi assignments. We also (e) benchmark $dist_{ra}$–BnB against an off-the-shelf MILP solver. The MILP formulation of our CD-CRSS model is presented in the Appendix A. Finally, (f) the effect of space partitioning is shown. That is, large detours are prevented while serving most of the customers using ad hoc drivers instead of dedicated drivers.

The evaluated approaches are shown in Table 5.2. They are combinations of assignment and routing strategies. $dist_{ra}$–BnB consists of near-optimal assignment by using

Table 5.1: Parameter set-up

Parameter	Values	Default
Number of customers	16 - 2048	256
Ratio customers to drivers	1, 2, 4, 8	4
Max. driver load m	4, 6, 8, 10, 12	8
Fraction f	0.05, 0.1, 0.15, 0.2	
Threshold t	1.05, 1.1, 1.15, 1.2	
Sample area	$100km^2$	
Drivers: $\langle start, end \rangle$	$\langle Zipfian, Uniform \rangle$	
Market share Melbourne (%)	Woolworths: 34, Coles: 27.6, Aldi: 11.4, IGA: 7, Others: 20	

Table 5.2: $dist_{ra}$–BnB and the Baselines

Acronym	Assignment	Routing
V–NN	Path-generator Voronoi	NN
V–BnB	Path-generator Voronoi	BnB
$dist_{ra}$–NN	Route-Adaptive Distance	NN
$dist_{ra}$–BnB	Route-Adaptive Distance	BnB

shortest route-adaptive distance, and optimal routing through our BnB technique. The other three are our baselines. We evaluate them in terms of *service cost, processing time, average detour* and *proportion of customers served*. Service cost for an ad hoc driver is the difference between the delivery route's and the k^+k^- route's distances. The overall service cost is the sum of all ad hoc drivers' service costs plus the distance traveled by dedicated drivers. *Average detour* and *proportion of customers served* are for ad hoc drivers only.

5.4.1 Selecting Adequate Maximum Driver Load

Assignments without load balancing and using our shortest route-adaptive distance measure yield results comparable to optimal solutions computed by a MILP solver. However, processing time at the routing stage grows exponentially as the number of customers per assignment increases. Figure 5.8 (left) shows that the service cost is marginally reduced when the number of customers per assignment is > 8, and yet the processing time differ-

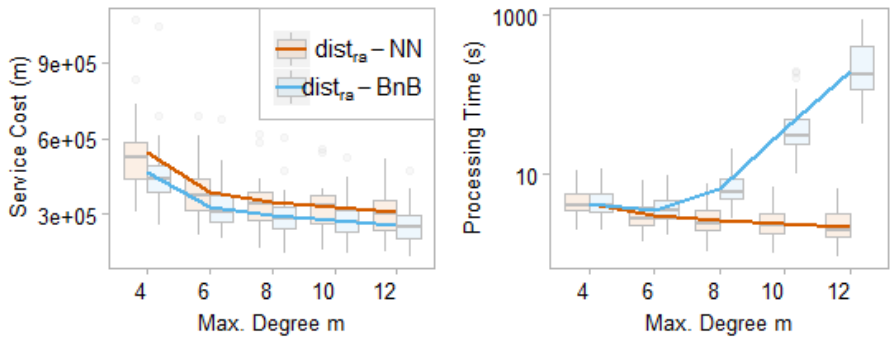

Figure 5.8: **Left.** Assignments of at most 8 customers is enough to keep comparable costs. Increasing the size of the assignments yields only marginal cost reductions. **Right.** Not balancing the assignment load, at least down to 8 customers, increases the processing time exponentially.

ence is rather significant as shown in Figure 5.8 (right). We are after an assignment size that allows efficient computation without affecting the solution quality to a great extent. We chose $m = 8$ to be this default assignment size in the rest of our experiments, except when we benchmark $dist_{ra}$–BnB in Section 5.4.5.

5.4.2 Comparing Our Model with the Current Model

We solve instances of CD-CRSS and the current model using $dist_{ra}$–BnB with maximum driver load $m = 8$. In the current model, the pick-up store is optimized independently from the delivery route. In this set of experiments, we assume the retailer has chosen the nearest store as the pick-up location for each order. In Figure 5.9 (left), it is shown that service cost in CD-CRSS is up to 20% less than the current model's service cost. According to the trend, the reduction may even increase as there are more customer requests. There are a few CD-CRSS instances where the cost is higher. This is expected as we are not computing the optimal solution, i.e., optimal assignment and optimal routing. $dist_{ra}$–BnB is also efficient because it solves instances of both models in comparable processing times. Instances of 256—default—and 2048 customers are solved in ≈ 8 and ≈ 100 seconds, respectively. $dist_{ra}$–BnB is readily deployable since only around 40 requests are submitted in 100 seconds in Melbourne.

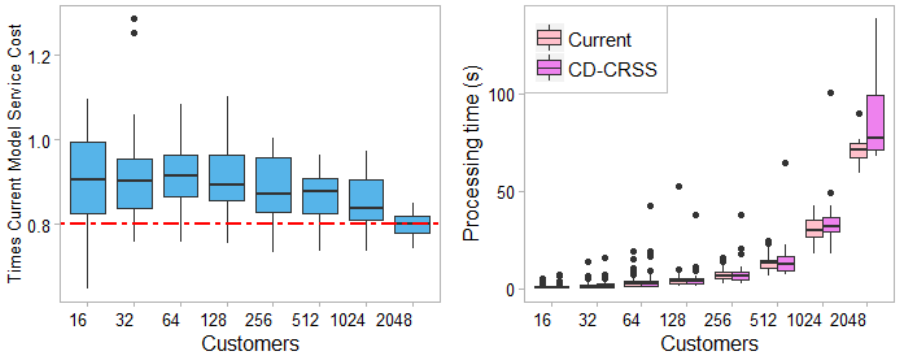

Figure 5.9: **Left.** Service cost in CD-CRSS is up to 20% less than the current model's service cost. **Right.** $dist_{ra}$–BnB is efficient as it solves instances of both models in comparable processing times. 2048-request instances are solved in ≈ 100 seconds.

5.4.3 Adopting an Amazon-like Model

Amazon customers buy based on the product characteristics themselves regardless of where the products come from. It is a retailer-free model that can be adopted by CD-CRSS. This solution would receive the customers' requests and optimize not only the store but also the retailer, along with the delivery route. Having two extra degrees of freedom, i.e., any store and any retailer, enables finding ad hoc drivers who live nearby the customers. It is ideal to get your neighbor to pick up your online shopping. CD-CRSS achieves dramatic further reduction of the cost, up to 50% compared with the current model. Figure 5.10 shows how the service cost in CD-CRSS has been further reduced when an Amazon model is adopted. The retailer-specific scenario in this figure was discussed in Section 5.4.2 and also shown in Figure 5.9 (left).

5.4.4 Effectiveness of Shortest Route-Adaptive Distance

In Figure 5.11, we show the effect of the shortest route-adaptive distance upon routing costs. We compute the ratio of the service cost using our shortest route-adaptive distance to the service cost based on path-generator Voronoi assignments. Our proximity measure significantly outperforms path-generator Voronoi-based assignments in both routing schemes. The difference increases as the number of customers increases. Up to 50% less cost is attained with 2048 customer requests. Baseline V–BnB was not even able to

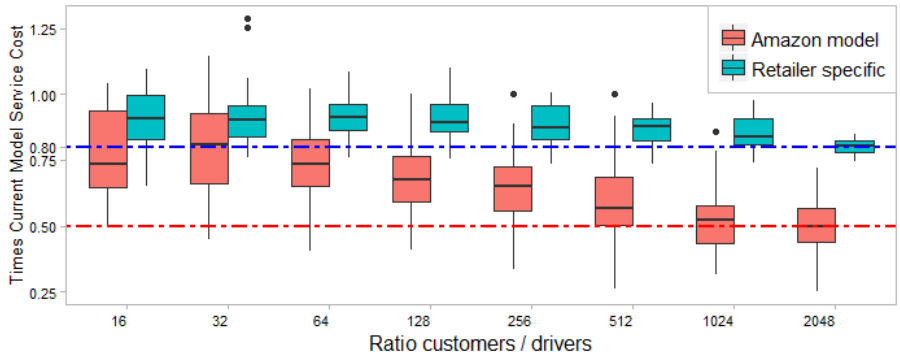

Figure 5.10: Further minimization up to 50% is achieved when an Amazon model is adopted.

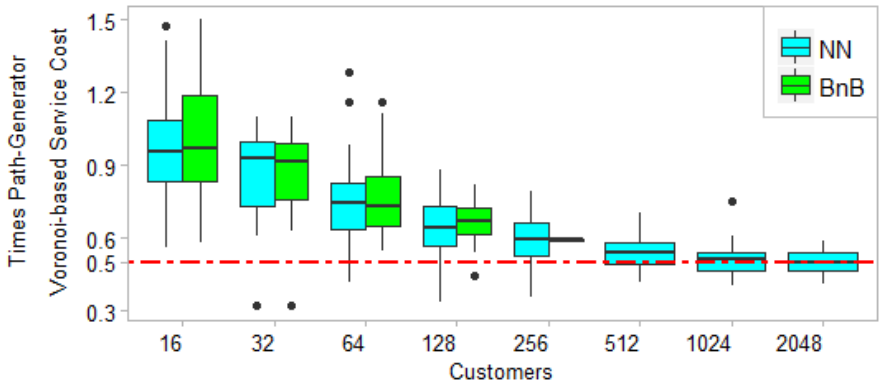

Figure 5.11: *Shortest route-adaptive distance* yields significantly cheaper routes—up to 50% less costly—in both routing schemes than path-generator Voronoi-based assignments. Its effectiveness increases along with the number of customers.

terminate the computation in reasonable time when the number of customers was 512 or more.

5.4.5 Comparing Our Solution with the Optimum

We benchmark $dist_{ra}$–BnB against the MILP formulation, whose solutions are computed by Google's GLOP solver. In this section, the numbers of customers are smaller than those presented in Table 5.1 as the GLOP solver takes prohibitive computation times for instances larger than 16. Moreover, we compare the service cost of the solutions yielded by $dist_{ra}$–BnB without balancing the driver load as by doing so we get the cheapest solutions.

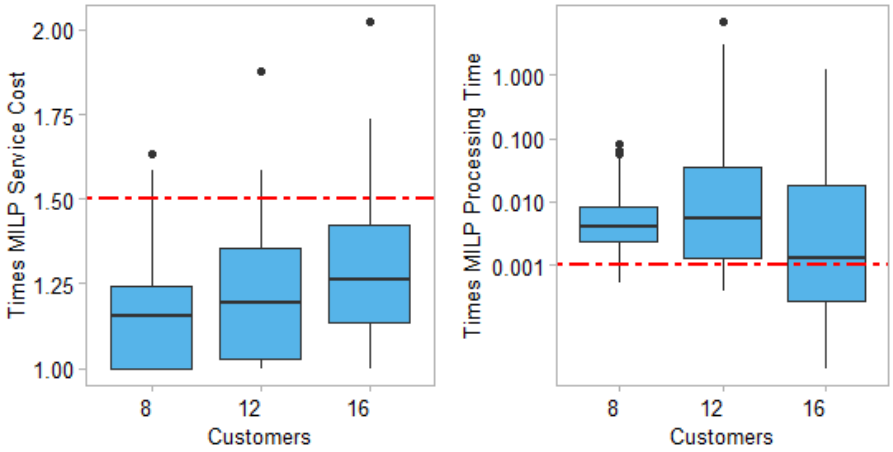

Figure 5.12: **Left.** The service cost of solutions yielded by $dist_{ra}$–BnB is not greater than 1.5 times the optimal service cost computed by a MILP solver. **Right.** Our algorithm finds near-optimal solutions two to three orders of magnitude faster.

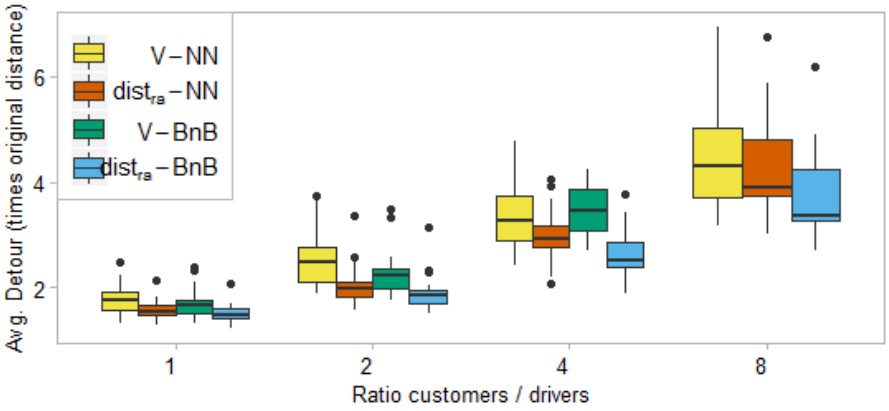

Figure 5.13: Average detour increases as the ratio customers to ad hoc drivers increases.

In Figure 5.12, we show $dist_{ra}$–BnB yields solutions with service cost ≤ 1.5 times the optimum (left) two to three orders of magnitude faster (right).

5.4.6 Effect of Space Partitioning

Space partitioning aims to alleviate the detours ad hoc drivers endure as all customers are served by them. Large detours are evident as the ratio customers to ad hoc drivers increases, as shown in Figure 5.13.

We partition the space by expanding the ad hoc drivers' k^+k^- routes, or by enforcing

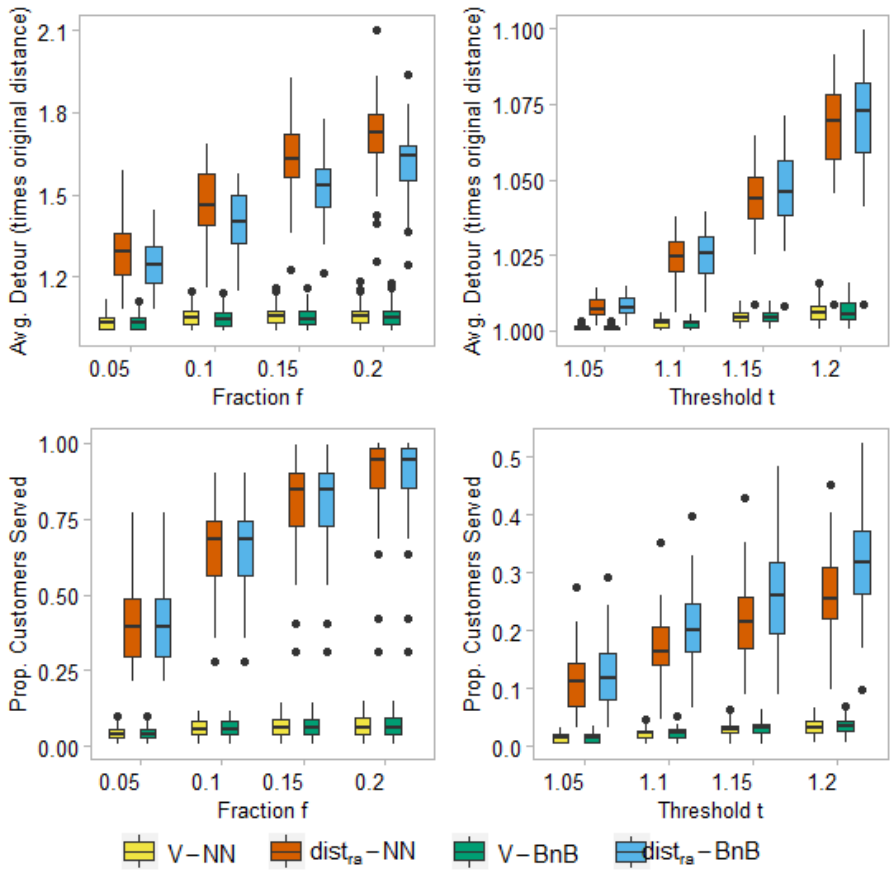

Figure 5.14: Effect of different **(left column)** expansion fractions, and **(right column)** thresholds when space is partitioned before assignment. Note how $dist_{ra}$–BnB can serve a large proportion of customers using ad hoc drivers and keep the detours controlled.

a threshold. In Figure 5.14, average detours behave as expected. In the case of expansion by a fraction, since detours depend on the number of customers within a partition, i.e., upper bound (5.9), they may be still large. When a threshold is set, the average detour is completely controlled.

Besides limiting the detours, the most notable result has to do with the proportion of customers served by ad hoc drivers. Nearly all customers are served by ad hoc drivers when the path-expansion fraction f is $\in [0.15 - 0.2]$. Even though path-generator Voronoi-based routing shows almost no detour, this occurs at the expense of ad hoc drivers serving nearly no customer. Instead, dedicated drivers are heavily used by Voronoi-based routing which may cause significant costs to the retailer compared to ad hoc drivers.

5.5 Summary

Our work proposes CD-CRSS, a novel crowdsourced delivery model where large retailers could achieve up to 20% further savings—compared with current models—because CD-CRSS optimizes pick-up stores and delivery routes concurrently. However, computing a delivery route for each ad hoc driver within this setup is harder as the search space is exponentially expanded. This is equivalent to solve a new problem on graphs that we call Group Hamiltonian Path with Precedence Constraints.

To solve instances of CD-CRSS effective and efficiently, we present $dist_{ra}$–BnB, a two-stage approach that (a) assigns online customers to ad hoc drivers, and then (b) computes optimal routes for the drivers. In the assignment stage, our shortest route-adaptive distance outperforms shortest distances—path-generator Voronoi—by 50% on 2048-request instances. In the routing stage, we customized branch-and-bound techniques to deal with the expanded search space. We speed up the routing by using upper and lower bounds simultaneously. Therefore, $dist_{ra}$–BnB is readily deployable as we were able to route 2048-request instances in $\approx$ 100 seconds for Melbourne, where only around 40 requests are submitted within the same time interval.

We also balance the load of the ad hoc drivers and show that serving at most 8 customers achieves fast routing computation and good quality in realistic settings.

To deal with possible large detours, we present optional spatial partitioning which gives max-detour-cost guarantees to the ad hoc drivers. We show that sufficiently large partitions, e.g., expansions of the original paths of the drivers between 15% to 20% of their distance, prevent large detours while serving most of the customers by using ad hoc drivers instead of dedicated drivers, who may be more expensive.

As a futuristic scenario, we imagined an Amazon-like model where the selection of the retailer is also made transparent to the customer. By doing so, CD-CRSS was able to find ad hoc drivers nearby to the customers more frequently. This reduced the service cost dramatically up to 50% compared with the current model.

Chapter 6

Conclusions and Future Work

IN times when global warming has reached alarming levels mainly due to human contribution, and particularly, because of CO_2 from fossil fuel burning, it is imperative to search for greener alternatives in transportation. Sharing-economy transportation models such as ride-sharing and crowdsourced delivery are believed to be these sought alternatives as they assume the use of vehicle seats that otherwise would have been empty. However, both models have failed to provide the convenience and incentives [15] than competing models such as ride- and delivery-sourcing offer. This has been detrimental for the adoption of ride-sharing and crowdsourced delivery, and it contrasts with what is observed in ride- and delivery-sourcing platforms. Uber and Spark Delivery are notorious examples of such platforms to which more and more users sign up. Yet, ride- and delivery-sourcing are certainly not among environment-friendly alternatives [22, 45, 58, 62]. Our goal in the thesis has been to propose *models* that encourage the massive uptake of ride-sharing and crowdsourced delivery services.

The representation of our models derived into novel *graph-theoretic problems* related to well-known graph problems such as Steiner trees and Hamiltonian paths. These well-known problems were not able to abstract the requirements of our proposed models. In our activity-based ride-sharing model introduced in [44] and presented in Chapter 2, we considered intermediate meeting points and a common destination where a desired activity can be carried out. The classical Minimum Steiner Tree (ST) problem would capture the intermediate meeting points through Steiner vertices, yet the common destination, which is the root of the tree, is computed instead of given. This new degree of freedom is captured by the Variable Steiner Tree (VST) problem, our novel graph-theoretic problem

[44] that is a generalization of the ST problem.

Chapters 3 and 4 further explored the activity-based ride-sharing model presented in Chapter 2. In Chapter 3, we dropped the assumption that an intermediate meeting point can be any intersection in a city. This novel model required the meeting points to be hot-spots only. This requirement constrained the set of Steiner vertices and thus, we defined a constrained version of the VST problem that we called the Constrained Variable Steiner Tree (C-VST) problem. In Chapter 4, we focused on the congestion caused by multiple and simultaneous activity-based ride-sharing travel plans. This congestion-aware model resembles the Multicast Congestion (MC) problem, a well-known graph problem where multiple servers multicast to their customers in the same telecommunication network. Routing from each server to its customers is represented by an ST and thus, the MC problem's goal is to minimize the number of edges used by more than one ST. Our model required VSTs instead of STs. Therefore, we proposed a generalization of the MC problem.

In Chapter 5, our novel Group Hamiltonian Path with Precedence Constraints Problem (GHPPCP) generalizes the classical Hamiltonian path problem in graph theory. In the classical Hamiltonian path problem, one must find whether there exists at least a path between a source and destination vertices that visits the rest of the vertices only once. Finding a delivery route for a driver in our novel crowdsourced delivery model is equivalent to computing a Hamiltonian path. Yet, this is a special version called Hamiltonian Path with Precedence Constraints Problem as a store must *precede*—be visited before—its corresponding customer. In our model, the store for each order has not been chosen yet, thus the delivery route for a driver is a path that visits the stores that minimize the route cost, and still visits all the customers once.

All our novel graph problems are NP-hard. This means that exact solutions to these problems for large—realistic—instances would be prohibitive. Query response times are expected to be quick and the quality of travel plans must be high. Certainly, quality is a variable rather difficult to measure on NP-hard problems and, particularly, with instances of considerable size. Our approximate approaches compute solutions for city-wide instances in reasonable time and thus, they are readily deployable. Our approaches

are also effective since the costs of the travel plans computed by them are comparable to exact solutions on small instances and cheaper when compared to the state-of-the-art approaches.

Note how we aimed at generalization. We observed that *users*, *meeting points* (resp. *pick-up points*), and *destinations* (resp. *online customers*) are the three categories that exist in ride-sharing (resp. crowdsourced delivery). If we generalize the *destinations* by considering any Point of Interest (POI) as candidate destination as long as the users can perform their desired activities in them then, we get *activity-based ride-sharing*—Chapters 2, 3 and 4. What if we generalize the *pick-up points* instead? We cannot pick up a person from two different places at the same time, yet we can do so with goods! The result was a *crowdsourced delivery model with guaranteed cheaper costs*—Chapter 5.

In the next sections, we discuss our contributions and their results. Finally, some research directions are mentioned.

6.1 Activity-Based Ride-Sharing with Hot-Spots

The main question we wanted to answer in Chapter 3 was *whether the constraint put on meeting points to be only suitable places would make ride-sharing more effective*. According to the theory behind an ST, having intermediate points—Steiner vertices—optimally chosen, guarantees a minimum-cost tree. That is, having intermediate meeting points is convenient for both drivers and passengers in terms of cost—distance or travel time—as long as these meeting points are optimal. However, this holds only if the search space for the meeting points considers all the vertices in the road network. When the search space is shrunk by constraining candidate meeting points to be within a subset—hot-spots only—optimality of the system is not guaranteed. In conclusion, meeting only at hot-spots does not guarantee less detour for either drivers or passengers. Is the likely extra detour, due to hot-spots, a good choice?

Chapter 3 shows that hot-spots are indeed a good choice. Ride-sharing becomes trustworthy because safety is increased through video surveillance in the hot-spots. As there exist fewer hot-spots in the road network than normal intersections, there are more users

in each hot-spot. The experimental results showed that hot-spots are rather popular meeting points. In Melbourne samples, from 5 up to 20 people gather in ≈ 33% of the hot-spots in the same hour, to ride-share. Our results show that, as a consequence, the vehicle occupancy ratio reaches ≈ 2.8[1]. This result is a massive improvement when it is compared to 1.08, the ratio shown for Melbourne in Section 1.1. This achieved ratio is comparable to the ratio computed where users are allowed to meet at any intersection, which has been shown to match more people than classical ride-sharing without meeting points [44, 73].

Popular meeting points not only contribute to matching more people but also to maximizing the likelihood of a round-trip arrangement. Since many people start their journey from the same hot-spot and because our model is activity-based—it implies that if they ride-share from the same hot-spot, they are going to the same POI—there are more chances to find a round-trip arrangement for passengers. This is convenient for passengers and also provides incentives for drivers because they can easily find passengers with whom share the travel costs without making detours.

6.2 Congestion-Aware Activity-Based Ride-Sharing

To compute optimal travel plans in our previous models, we needed to solve our tree-based problems: VST and C-VST. Each tree corresponds to the travel plan of a group of people who fit in a vehicle. The assumption was that the interaction between trees can be safely ignored as the proportion of true ride-sharing trips is marginal nowadays. When we considered the forests of VSTs along with their mutual interactions, we observed congested edges in the graph. For that to happen, our assumption must shift to a time when ride-sharing is ubiquitous and there exist large numbers of participants.

Our literature review revealed a similar problem in telecommunication networks called the Multicast Congestion (MC) problem. Congestion in the network is observed when several sources multicast to their corresponding audience in the same network. The difference between the MC problem and ours lies in the type of trees—multicast within each group is represented by an ST. We also found that a transportation science

[1] Section 3.4.2 explains how we computed this statistic.

technique, called Static Traffic Assignment (STA), was used to re-route vehicles in case of congestion between thousands of simultaneous shortest paths. The STA approach is very similar to one of the approaches to the MC problem [10].

We thus designed a meta-algorithm, called VST-based Congestion-Aware (VST-CA), inspired by both STA and this particular approach to the MC problem. VST-CA re-routes vehicles to prevent heavy overlapping between their travel plans. Consequently, reduction in travel time is also achieved compared with a non-congestion-aware approach, e.g., we chose VST-based Ride-Sharing (VST-RS) [44]. Yet, in about half of the samples in both synthetic and real networks, reduction was not possible. We designed a metric, called Weighted Average of Relative Loads (WARL), to detect the causes of this behavior. WARL measures the level of congestion in a solution[2]. We observed that VST-CA was able to reduce travel cost by more than 10% only within an interval of WARL. Otherwise, when the network was either almost free or over-congested—outside this interval—nearly no reduction is observed. This explained our previous results.

VST-CA is also an efficient meta-algorithm. It invokes any non-congestion-aware ride-sharing algorithm until traffic loads reach equilibrium. Since it is a meta-algorithm that avoids *all-or-nothing assignments*, convergence is guaranteed to occur as shown in Section 4.3.3. VST-CA performs a constant number of calls.

6.3 Adopting Activity-Based Ride-Sharing in Crowdsourced Delivery

In Chapters 2, 3 and 4, we generalized the destinations in ride-sharing. The destination for each transportation request was not fixed. Instead, it was the optimal—in terms of cost minimization—among the set of destinations where users can perform their desired activities. In this last contribution, we generalized the pick-up points instead. The pick-up store for each order was not fixed. The chosen pick-up store is the one that minimized the cost of the delivery route of a private driver. This came along with the realization that, even though retailers aim to minimize delivery times, they do not optimize private

[2]Section 4.4 explains how we computed this statistic.

drivers' delivery routes. We proposed a crowdsourced delivery model where drivers can pick up the products for an order from the store of a retailer that minimizes the delivery route of the driver. That is, at planning time, we (a) compute the optimal delivery route, and (b) choose the pick-up store for each order, both tasks as part of the same optimization process. The extra degree of freedom in our model yields a larger search space when computing the optimal delivery route. Therefore, an optimal solution to an instance of our model is guaranteed to be cheaper than an optimal solution within current models. We showed experimentally that our model could achieve up to 20% further savings compared with current models.

A solution is the set of delivery routes, one for each private driver. In a large-scale instance, thousands of customers must be served within short time intervals. We thus solve the problem with a two-stage approach: assignment and routing. To be readily deployable, we compute the assignments through a proposed proximity heuristic, and compute the delivery routes exactly through an efficient branch-and-bound algorithm. The assignment heuristic is a novel proximity measure between a vertex and a path that, within our model, is able to reduce the total service cost up to 50% compared with the baseline heuristic—the shortest-distance measure.

Computing a delivery route in the second stage is equivalent to solving our novel graph-theoretic problem that we call Group Hamiltonian Path with Precedence Constraints Problem (GHPPCP). Our custom branch-and-bound technique solves this novel problem to optimality and efficiently by combining upper and lower bounds. Our two-stage approach computes solutions with service cost of at most 1.5 times the optimum up to three orders of magnitude faster than a mixed-integer linear program solver.

As an extension, we provide space partitioning algorithms for limiting driver detours. Although we do balance the load of each driver in the assignment stage, drivers would be required to deviate considerably from their original routes, especially when the ratio of customers to drivers is high enough. If drivers have to make large detours, the model becomes inconvenient. We offer path-expansion and cost-threshold partitioning techniques, which offer maximum-detour guarantees to the drivers. Yet, limiting detours to private drivers comes with the price of hiring per-hour dedicated drivers, who may be

more expensive. We show that our solution can keep detours controlled and serve almost every customer with private drivers when the original paths of the drivers are expanded between 15% to 20% of their distance.

6.4 Future Work

6.4.1 Dynamic versions of our graph-theoretic problems

Both activity-based ride-sharing and crowdsourced delivery can be categorized as either static or dynamic. This characterization is given by how the approach deals with transportation requests. If the approach computes travel plans for the requests that are available at planning time only, it is considered as static. If the approach can schedule the requests that come up after the planning stage and still satisfies the constraints of the plans that are already in execution, the approach is within the dynamic category. Our proposed approaches are static for the following two reasons. First, by approaching the problems as static, we were able to reduce them to well-known graph-theoretic problems and thus, our solutions to them are more generic. Second, we can process the available transportation requests in batch, and the ones that become available later are stored to be processed in the next batch. Although this last approach enables our solutions to be ready for deployment, better plans may be possible should we process the requests dynamically.

There are interesting consequences at the graph-theoretic level. Our VST, C-VST and GHPPCP problems must shift to their dynamic versions. For example, within an activity-based ride-sharing scheme, new participants' requests would force to reschedule the already computed forest of VSTs. The goal is to minimally degrade the solution's quality since re-computation is not performed.

6.4.2 Our Models with User-Specific Constraints

In our contributions, time windows for drivers, passengers, and online customers are not considered. For example, in Section 3.4, simulations are performed among users who

depart at the same hour since it is more likely they ride-share. This assumption can be dropped and time budgets per user can be considered when solving instances of our problems. A user can provide her earliest departure time and latest arrival time. The difference between these bounds provides the user's time budget which can be used to constrain the search space even more. Many other variables like traffic conditions, user delays, parking availability, etc., have a great impact on time constraints. Some of these are purely stochastic phenomena that we also consider as a good candidate research direction in Section 6.5.

In Chapter 3, the suitability of a candidate hot-spot was measured through our *gain ratio* function whose domain is the set of users who may meet at such evaluated hot-spot. If we want to include users' time budgets, the set of candidate hot-spots as well as the set of users that are both input to the gain ratio function must be updated every time this function is called by considering whether a hot-spot is still available and users are within their time budgets. Similar considerations must be made in the case of the trading hours of the POIs. On the other hand, in Chapter 4, users' time budgets must be considered within our VST-RS algorithm. Every possible travel plan for subgroups of at most z users are computed in the last stage of VST-RS mimicking how the Dreyfus and Wagner's algorithm [21], the seminal work on the ST problem, works. Again, users must be filtered based on their time budgets before being considered as part of a subgroup for which the travel plan is to be computed. In Chapter 5, we already considered an ellipse-based "space" filter when a threshold upon the cost of the driver's delivery route required to be enforced. Such a filter may be extended to consider users' time budgets.

Chaube et al. [15] concluded that people would participate more in ride-sharing if issues like convenience and incentives are covered. Perhaps, a woman considers ride-sharing an insecure transportation option and thus she does not find it convenient. She would like to specify a woman as a driver as part of her preferences when she ride-shares. Other users may wish to ride-share with old people because they find elders more friendly. These are examples of user preferences that could also be included in our models. Contrary to users' time budgets where users are known to be part of an arrangement during the execution of the algorithm, gender- and age-based constraints are easier to en-

force since the feasibility of a match between users is known straight away by checking out their gender and/or age.

6.5 Stochastic Models

Traffic conditions are not the only variable that affects our deterministic models. Each of the components of our models, i.e. users, meeting points (resp. pick-up stores) and destination (resp. online customers), could add *uncertainty* to our optimization process.

A user must be modeled differently depending on the role she is willing to play within the model, i.e. driver, passenger or online customer. If a passenger does not show up in ride-sharing or if an online customer decides to cancel her order in crowdsourced delivery, the optimization process is not affected to the same extent as when a driver decides to drop-out. When a driver drops-out, other passengers (resp. orders) already assigned to the driver are affected too. This situation gives rise to the idea of inclusion of *dedicated drivers*—who were already added to the crowdsourced delivery model—to guarantee the quality of service despite the cost increment.

On the other hand, some hot-spots in a city are more popular than others, so the probability of finding parking spaces is not distributed uniformly across the candidate meeting points. Moreover, their availability also depends on the time of the day or whether it is a weekday or weekend day. A more realistic model must include the uncertainty related to hot-spots given the importance they have on making ride-sharing more convenient to users.

Multiple scenarios must be identified based on the uncertainty of the components of our models, i.e., users, meeting points (resp. pick-up stores) and destinations (resp. online customers). Each scenario has an associated probability and considers which components may occur. Based on the knowledge of this probability distribution, our algorithms must be adapted to plan ahead and ”buy” some edges that will be part of the travel (resp. delivery) plans. This is particularly clear in our congestion-aware ride-sharing model where resources become very expensive to purchase when the travel plans start to realize. Since our models generalize the destinations (resp. pick-up stores), i.e., they are not fixed,

this initial purchase of edges may not end up in a connected component—there is no fixed vertex from which start. This is a subtle but significant difference from previous works on the stochastic version of the ST problem [34, 35, 38] that consider a fixed root and their initial purchase can start from it. Of special interest is the work of Gupta and Pál [33], who presented a version of the stochastic ST *without root*, that resembles our activity-based ride-sharing model where the destination can be considered the missing root in their model. When a realization of the components is learned, our algorithms must complete the travel plans by purchasing augmenting edges.

Appendix A

CD-CRSS MILP Formulation

In Chapter 5, we proposed CD-CRSS, a novel crowdsourced delivery model where the selection of the pick-up store for each order, and the computation of the ad hoc driver's delivery route, are concurrent. CD-CRSS can be formulated as a Mixed-Integer Linear Program (MILP) over a directed graph $G \coloneqq \langle V, A \rangle$, where the set of vertices V correspond to stores S and customers C as well as ad hoc drivers' start and end locations. Besides ad hoc drivers, the model allows the use of dedicated drivers who may be hired from a courier company. Let $D \coloneqq H \cup F$ be the whole set of drivers where H and F are the sets of ad hoc and dedicated drivers, respectively. Let k^+ and k^- be the start and end locations of the ad hoc driver $k \in H$. Each dedicated driver starts and ends at an assigned store thus, $\forall k \in F : k^+ = k^- = s_k \in S$. For each arc $(i, j) \in A$, $c_{i,j}$ and $t_{i,j}$ are the shortest distance—cost—and time to travel from i to j in the road network, respectively. Let R be the set of retailers thus, each retailer $r \in R$ owns a set of stores S_r. Two types of decision variables are used: (a) boolean variables $x^k_{i,j}$, and (b) time variables B^k_i. If driver k travels through arc $(i, j) \in A$ in the solution, then $x^k_{i,j} = 1$, otherwise $x^k_{i,j} = 0$. Time variable B^k_i corresponds to the time at which driver k starts service at location i. We aim to minimize the total cost of ad hoc and dedicated drivers' delivery routes. The MILP formulation is as follows:

$$\min \sum_{k \in D} \sum_{(i,j) \in A} c_{i,j} x^k_{i,j} \tag{A.1}$$

such that,

$$\sum_{s \in S_r} \sum_{(s,j) \in A} x^k_{s,j} \leq 1, \ \forall r \in R, k \in H \tag{A.2}$$

$$\sum_{(s_k,j)\in A} x^k_{s_k,j} \leq 1, \ \forall k \in F \tag{A.3}$$

$$\sum_{k\in D} \sum_{(q,j)\in A} x^k_{q,j} = 1, \ \forall q \in C \tag{A.4}$$

$$\sum_{(q,j)\in A} x^k_{q,j} - \sum_{(q,j)\in A} x^k_{q,j} \times \sum_{s\in S_q} \sum_{(s,j)\in A} x^k_{s,j} = 0, \forall q \in C, k \in H \tag{A.5}$$

$$\sum_{(q,j)\in A} x^k_{q,j} - \sum_{(q,j)\in A} x^k_{q,j} \times \sum_{(s_k,j)\in A} x^k_{s_k,j} = 0, \forall q \in C, k \in F_q \tag{A.6}$$

$$\sum_{(i,j)\in A} x^k_{i,j} - \sum_{(j,i)\in A} x^k_{j,i} = 0, \ \forall i \in S \cup C, k \in D \tag{A.7}$$

$$\sum_{(k^+,j)\in A} x^k_{k^+,j} = 1, \ \sum_{(j,k^-)\in A} x^k_{j,k^-} = 1, \ \forall k \in H \tag{A.8}$$

$$B^k_j \geq x^k_{i,j}(B^k_i + t_{i,j}), \ \forall (i,j) \in A, k \in D \tag{A.9}$$

$$B^k_q \geq B^k_{s_q} + t_{s_q,q}, \ \forall q \in C, s_q \in S_q, k \in D \tag{A.10}$$

$$\begin{gathered} x^k_{i,j} \in \{0,1\}, \ \forall (i,j) \in A, k \in D \\ B^k_i \geq 0, \ \forall i \in V, k \in D \end{gathered} \tag{A.11}$$

In (A.2), an ad hoc driver visits at most one store of a retailer. In (A.3), s_k represents the store from where dedicated driver k starts. Thus, a dedicated driver leaves a store at most once. In (A.4), every customer must be served, either by an ad hoc or dedicated driver. In (A.5), if an ad hoc driver visits a customer, this driver must have visited the store. S_q is the set of stores for customer q. Constraints (A.6) are similar for dedicated drivers. In (A.7), flow conservation is enforced for each driver and in each vertex, except in drivers' start and end locations. In (A.8), we enforce each ad hoc driver to leave her start location and to arrive at her end location. In (A.9), if a vertex is visited after another, their service times must be coherent. This makes subtours impossible. In (A.10), a customer must be visited after picking up the products from the store —precedence constraints. Integrality and nonnegativity are enforced in constraints (A.11). Sets of constraints (A.5), (A.6) and (A.9) must be linearized.

Bibliography

[1] N. Agatz, A. L. Erera, M. W. Savelsbergh, and X. Wang, "Dynamic ride-sharing: A simulation study in metro Atlanta," *Procedia-Social and Behavioral Sciences*, vol. 17, pp. 532–550, 2011.

[2] K. Aissat and A. Oulamara, "Dynamic Ridesharing with Intermediate Locations," in *2014 IEEE Symposium on Computational Intelligence in Vehicles and Transportation Systems (CIVTS)*. IEEE, 2014, pp. 36–42.

[3] U. M. Aïvodji, S. Gambs, M.-J. Huguet, and M.-O. Killijian, "Meeting points in ridesharing: A privacy-preserving approach," *Transportation Research Part C: Emerging Technologies*, vol. 72, pp. 239–253, 2016.

[4] L. Alarabi, B. Cao, L. Zhao, M. F. Mokbel, and A. Basalamah, "A Demonstration of SHAREK: An Efficient Matching Framework for Ride Sharing Systems," in *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2016, p. 95.

[5] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences*, vol. 114, no. 3, pp. 462–467, 2017.

[6] Amazon, "amazon FLEX," https://flex.amazon.com/, 2018, [Online; accessed 20-Nov-2018].

[7] J. Arbib and T. Seba, "Rethinking Transportation 2020-2030," *RethinkX, May*, 2017.

[8] A. M. Arslan, N. Agatz, L. Kroon, and R. Zuidwijk, "Crowdsourced delivery–A dynamic pickup and delivery problem with ad hoc drivers," *Transportation Science*, vol. 53, no. 1, pp. 222–235, 2019.

[9] N. Ascheuer, L. F. Escudero, M. Grötschel, and M. Stoer, "A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing)," *SIAM Journal on Optimization*, vol. 3, no. 1, pp. 25–42, 1993.

[10] A. Baltz and A. Srivastav, "Fast approximation of minimum multicast congestion - Implementation VERSUS Theory," *RAIRO - Operations Research*, vol. 38, no. 4, pp. 319–344, 2004. [Online]. Available: https://doi.org/10.1051/ro:2004028

[11] Bringg Team, "Flexible Dispatch Software Tailored to your Companys Needs," https://www.bringg.com/platform/modules/flexible-dispatch/, 2019.

[12] Bringg Team, "[Techcrunch] Walmart partners with delivery logistics platform Bringg on last-mile grocery delivery," https://www.bringg.com/in-the-news/highlights/techcrunch-walmart-partners-with-delivery-logistics-platform-bringg-on-last-mile-grocery-delivery/, 2019.

[13] Bureau of Infrastructure, Transport and Regional Economics (BITRE), "Traffic and congestion cost trends for Australian capital cities," BITRE, Tech. Rep., 2015.

[14] B. Cao, L. Alarabi, M. F. Mokbel, and A. Basalamah, "Sharek: A Scalable Dynamic Ride Sharing System," in *2015 16th IEEE International Conference on Mobile Data Management*, vol. 1. IEEE, 2015, pp. 4–13.

[15] 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*. IEEE, 2010, pp. 1–8.

[16] P. Chen and S. Chankov, "Crowdsourced delivery for last-mile distribution: An agent-based modelling and simulation approach," in *2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. IEEE, 2017, pp. 1271–1275.

[17] W. Chen, M. Mes, and M. Schutten, "Multi-hop driver-parcel matching problem with time windows," *Flexible services and manufacturing journal*, vol. 30, no. 3, pp. 517–553, 2018.

[18] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, Tech. Rep., 1976.

[19] DHL, "DHL crowd sources deliveries in Stockholm with MyWays," http://www.dhl.com/en/press/releases/releases_2013/logistics/dhl_crowd_sources_deliveries_in_stockholm_with_myways.html, 2013, [Online; accessed 20-Nov-2018].

[20] F. Drews and D. Luxen, "Multi-hop ride sharing," in *Sixth annual symposium on combinatorial search*. Citeseer, 2013.

[21] S. E. Dreyfus and R. A. Wagner, "The Steiner problem in graphs," *Networks*, vol. 1, no. 3, pp. 195–207, 1972.

[22] G. D. Erhardt, S. Roy, D. Cooper, B. Sana, M. Chen, and J. Castiglione, "Do transportation network companies decrease or increase congestion?" *Science advances*, vol. 5, no. 5, p. eaau2670, 2019.

[23] L. F. Escudero, "An inexact algorithm for the sequential ordering problem," *European Journal of Operational Research*, vol. 37, no. 2, pp. 236–249, 1988.

[24] L. F. Escudero, M. Guignard, and K. Malik, "A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships," *Annals of Operations Research*, vol. 50, no. 1, pp. 219–237, 1994.

[25] D. J. Fagnant and K. M. Kockelman, "Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas," *Transportation*, vol. 45, no. 1, pp. 143–158, 2018.

[26] E. Ferguson, "The rise and fall of the American carpool: 1970–1990," *Transportation*, vol. 24, no. 4, pp. 349–376, 1997.

[27] B. Fuchs, W. Kern, and X. Wang, "Speeding up the Dreyfus–Wagner algorithm for minimum Steiner trees," *Mathematical methods of operations research*, vol. 66, no. 1, pp. 117–125, 2007.

[28] M. Furuhata, M. Dessouky, F. Ordóñez, M.-E. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," *Transportation Research Part B: Methodological*, vol. 57, pp. 28–46, 2013.

[29] L. M. Gambardella and M. Dorigo, "An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem," *INFORMS Journal on Computing*, vol. 12, no. 3, pp. 237–255, 2000.

[30] M. R. Garey, R. L. Graham, and D. S. Johnson, "The complexity of computing Steiner minimal trees," *SIAM journal on applied mathematics*, vol. 32, no. 4, pp. 835–859, 1977.

[31] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[32] P. Goel, L. Kulik, and K. Ramamohanarao, "Optimal Pick up Point Selection for Effective Ride Sharing," *IEEE Transactions on Big Data*, 2016.

[33] A. Gupta and M. Pál, "Stochastic Steiner Trees without a Root," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2005, pp. 1051–1063.

[34] A. Gupta, M. Pál, R. Ravi, and A. Sinha, "Boosted Sampling: Approximation Algorithms for Stochastic Optimization," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, 2004, pp. 417–426.

[35] A. Gupta, R. Ravi, and A. Sinha, "An Edge in Time Saves Nine: LP Rounding Approximation Algorithms for Stochastic Network Design," in *45th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2004, pp. 218–227.

[36] S. Hougardy and H. J. Priomel, "A 1.598 Approximation Algorithm for the Steiner Problem in Graphs," in *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999, pp. 448–453.

[37] Y. Huang, F. Bastani, R. Jin, and X. S. Wang, "Large Scale Real-time Ridesharing with Service Guarantee on Road Networks," *VLDB*, vol. 7, no. 14, pp. 2017–2028, 2014.

[38] N. Immorlica, D. Karger, M. Minkoff, and V. S. Mirrokni, "On the Costs and Benefits of Procrastination: Approximation Algorithms for Stochastic Combinatorial Optimization Problems," in *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2004, pp. 691–700.

[39] K. Jansen and H. Zhang, "An Approximation Algorithm for the Multicast Congestion Problem via Minimum Steiner Trees," 2002.

[40] K. Jansen and H. Zhang, "Approximation algorithms for general packing problems and their application to the multicast congestion problem," *Mathematical Programming*, vol. 114, no. 1, pp. 183–206, 2008.

[41] N. Kafle, B. Zou, and J. Lin, "Design and modeling of a crowdsource-enabled system for urban parcel relay and delivery," *Transportation research part B: methodological*, vol. 99, pp. 62–82, 2017.

[42] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[43] M. Karpinski and A. Zelikovsky, "New Approximation Algorithms for the Steiner Tree Problems," *Journal of Combinatorial Optimization*, vol. 1, no. 1, pp. 47–65, 1997.

[44] A. K. M. M. R. Khan, 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, GIS 2017, Redondo Beach, CA, USA, November 7-10, 2017*, 2017, pp. 6:1–6:10. [Online]. Available: http://doi.acm.org/10.1145/3139958.3139972

[45] Le Petit, Yoann and Earl, Thomas, "Europe's giant "taxi" company: is Uber part of the problem or the solution?" Transport & Environment, Tech. Rep., 2019.

[46] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *ICDE*, 2013, pp. 410–421.

[47] S. Ma, Y. Zheng, and O. Wolfson, "Real-Time City-Scale Taxi Ridesharing," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 27, no. 7, pp. 1782–1795, 2015.

[48] M. T. Mahin and T. Hashem, "Activity-aware Ridesharing Group Trip Planning Queries for Flexible POIs," *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, vol. 5, no. 3, p. 20, 2019.

[49] T. A. Manual, "Bureau of Public Roads," *US Department of Commerce*, 1964.

[50] D. Mölle, S. Richter, and P. Rossmanith, "A Faster Algorithm for the Steiner Tree Problem," *Lecture notes in computer science*, pp. 561–570, 2006.

[51] M. Olsen, "On the Complexity of Computing Optimal Private Park-and-Ride Plans," in *International Conference on Computational Logistics*. Springer, 2013, pp. 73–82.

[52] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ," https://www.openstreetmap.org, 2017.

[53] M. Patriksson, *The Traffic Assignment Problem: Models and Methods*. Courier Dover Publications, 2015.

[54] G. Robins and A. Zelikovsky, "Tighter bounds for graph Steiner tree approximation," *SIAM Journal on Discrete Mathematics*, vol. 19, no. 1, pp. 122–134, 2005.

[55] Roy Morgan, "Over 5 million Australians consider buying groceries online," http://www.roymorgan.com/findings/7911-australian-online-grocery-shopping-march-2019-201903220623, 2019, [Online; accessed 22-Nov-2019].

[56] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," *Proceedings of the National Academy of Sciences*, vol. 111, no. 37, pp. 13 290–13 294, 2014.

[57] M. W. Savelsbergh and M. Sol, "The General Pickup and Delivery Problem," *Transportation science*, vol. 29, no. 1, pp. 17–29, 1995.

[58] B. Schaller, "Unsustainable? The growth of app-based ride services and traffic, travel and the future of New York City," *Schaller Consulting*, 2017.

[59] S. Spieckermann, K. Gutenschwager, and S. Voß, "A sequential ordering problem in automotive paint shops," *International journal of production research*, vol. 42, no. 9, pp. 1865–1878, 2004.

[60] M. Stiglic, N. Agatz, M. Savelsbergh, and M. Gradisar, "The benefits of meeting points in ride-sharing systems," *Transportation Research Part B: Methodological*, vol. 82, pp. 36–53, 2015.

[61] M. Stiglic, N. Agatz, M. Savelsbergh, and M. Gradisar, "Making dynamic ride-sharing work: The impact of driver and rider flexibility," *Transportation Research Part E: Logistics and Transportation Review*, vol. 91, pp. 190–207, 2016.

[62] M. E. Sutherland, "Lyft and Uber increase congestion in San Francisco," *Nature Human Behaviour*, p. 1, 2019.

[63] R. Tachet, O. Sagarra, P. Santi, G. Resta, M. Szell, S. Strogatz, and C. Ratti, "Scaling Law of Urban Ride Sharing," *Scientific reports*, vol. 7, p. 42868, 2017.

[64] 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*. ACM, 2016, p. 72.

[65] C. Tian, Y. Huang, Z. Liu, F. Bastani, and R. Jin, "Noah: A Dynamic Ridesharing System," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 985–988.

[66] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A Unified Approach to Route Planning for Shared Mobility," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1633–1646, 2018.

[67] Uber, "UberPool vs. UberX - How Does UberPool Work?" https://www.uber.com/au/en/ride/uberpool/, 2019, [Online; accessed 5-Nov-2019].

[68] D. Van Vliet, "Road assignment: A case for incremental loading," in *GLTS Note 30*. Department of Planning and Transportation, Greater London Council, 1973.

[69] S. Vempala and B. Vöcking, "Approximating Multicast Congestion," in *International Symposium on Algorithms and Computation*. Springer, 1999, pp. 367–372.

[70] Victorian Department of Transport, "Victorian Integrated Survey of Travel and Activity (2009-2010)," 2011.

[71] Walmart, "Walmart Tests New Last-Mile Grocery Delivery Service," https://corporate.walmart.com/newsroom/2018/09/05/walmart-tests-new-last-mile-grocery-delivery-service, 2018, [Online; accessed 4-Oct-2019].

[72] Walmart Labs, "Crowdsourced Delivery," https://www.walmartlabs.com/case-studies/crowdsourced-delivery-for-online-grocery, 2019, [Online; accessed 22-Nov-2019].

[73] 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*. ACM, 2016, pp. 1:1–1:10.

[74] Y. Wang, S. Winter, and N. Ronald, "How much is trust: The cost and benefit of ridesharing with friends," *Computers, Environment and Urban Systems*, vol. 65, pp. 103–112, 2017.

[75] Y. Wang, S. Winter, and M. Tomko, "Collaborative activity-based ridesharing," *Journal of Transport Geography*, vol. 72, pp. 131–138, 2018.

[76] S. Winter and S. Nittel, "Ad hoc shared-ride trip planning by mobile geosensor networks," *International Journal of Geographical Information Science*, vol. 20, no. 8, pp. 899–916, 2006.

[77] X. Xing, T. Warden, T. Nicolai, and O. Herzog, "Smize: A Spontaneous Ride-Sharing System for Individual Urban Transit," in *German Conference on Multiagent System Technologies*. Springer, 2009, pp. 165–176.

[78] X. Zhang, Y. Asano, and M. Yoshikawa, "Mutually Beneficial Confluent Routing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 10, pp. 2681–2696, 2016.