# ABSTRACT

Title of thesis: A RIDE-SHARING SYSTEM WITH HIGH CAPACITY VEHICLES AND FLEXIBLE MEETING POINTS

Anna Petrone 2016

Directed by: Professor Gang-Len Chang
Department of Civil and Environmental Engineering

In many major cities, fixed route transit systems such as bus and rail serve millions of trips per day. These systems have people collect at common locations (the station or stop), and board at common times (for example according to a predetermined schedule or headway). By using common service locations and times, these modes can consolidate many trips that have similar origins and destinations or overlapping routes. However, the routes are not sensitive to changing travel patterns, and have no way of identifying which trips are going unserved, or are poorly served, by the existing routes. On the opposite end of the spectrum, personal modes of transportation, such as a private vehicle or taxi, offer service to and from the exact origin and destination of a rider, at close to exactly the time they desire to travel. Despite the apparent increased convenience to users, the presence of a large number of small vehicles results in a disorganized, and potentially congested road network during high demand periods. The focus of the research presented in this paper is to develop a system that possesses both the on-demand nature of a personal mode,

with the efficiency of shared modes. In this system, users submit their request for travel, but are asked to make small compromises in their origin and destination location by walking to a nearby meeting point, as well as slightly modifying their time of travel, in order to accommodate other passengers. Because the origin and destination location of the request can be adjusted, this is a more general case of the Dial-a-Ride problem with time windows. The solution methodology uses a graph clustering algorithm coupled with a greedy insertion technique. A case study is presented using actual requests for taxi trips in Washington DC, and shows a significant decrease in the number of vehicles required to serve the demand.

# A RIDE-SHARING SYSTEM WITH HIGH CAPACITY VEHICLES AND FLEXIBLE MEETING POINTS

by

Anna Petrone

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2016

Advisory Committee:
Dr. Gang-Len Chang, Chair/Advisor
Dr. Bruce L. Golden
Dr. Paul M. Schonfeld

# Table of Contents

List of Tables iv

List of Figures v

1 Introduction 1
1.1 Motivation . . . 1
1.1.1 Organization of thesis . . . 3
1.1.2 Description of proposed service . . . 3
1.1.3 Benefits of proposed service . . . 7
1.2 Existing Flexible Transport Modes . . . 8
1.2.1 Informal transit . . . 8
1.2.2 Micro-transit . . . 10
1.2.3 Casual carpooling . . . 11
1.2.4 Web-based carpooling . . . 12
1.2.5 Shared taxi . . . 13
1.2.6 Vanpools . . . 13
1.2.7 Dial-a-Ride service . . . 14
1.2.8 Flexible route bus . . . 15

2 Literature Review 17
2.1 Dial-a-Ride problem . . . 17
2.1.1 Exact solution methods . . . 20
2.1.2 Insertion heuristics . . . 21
2.1.3 Tabu search heuristic . . . 24
2.2 Shared taxi problem . . . 26
2.3 Carpooling problem . . . 29
2.4 Ride sharing with transfers . . . 33
2.5 School bus routing problem . . . 34
2.6 Vanpool routing problem . . . 38
2.7 Literature review conclusion . . . 39

3 Solution Algorithm Overview 41
3.1 Introduction . . . 41
3.2 Solution method overview . . . 45
3.3 Model inputs, outputs, objective, and constraints . . . 46

4 Solution Algorithm Phase I: Spatial Similarity 52
4.1 Part 1: Pickup-dropoff pairs enumeration . . . 52
4.2 Part 2: Iterative shortest path calculation . . . 53
4.3 Phase I conclusion . . . 56

5 Solution Algorithm Phase II: Request Assignment 58
5.1 Part 1: Determine candidate pickup-dropoff pairs . . . 58
5.2 Part 2: Travel time overlap . . . 60
5.3 Part 3: Greedy request to route segment assignment . . . 63
5.4 Part 4: Improvement heuristic . . . 67
5.5 Part 5: Route segment chaining . . . 69
5.6 Phase II conclusion . . . 71

6 Case Study: Washington DC Taxi Trips 72
6.1 Data sources . . . 74
6.1.1 Road network . . . 74
6.1.2 Bus stop network . . . 75
6.1.3 Demand data . . . 76
6.1.4 Shortest path engine . . . 77
6.2 Model Performance . . . 78
6.2.1 Phase I . . . 78
6.2.2 Phase II . . . 80
6.2.3 Instance descriptions . . . 81
6.2.4 Measures of effectiveness . . . 84
6.2.5 Performance summary . . . 90

7 Conclusion and Future Work 94
7.1 Future extensions . . . 94
7.1.1 Further parameter testing . . . 94
7.1.2 Modeling extensions . . . 95
7.2 Conclusion . . . 96

Bibliography 98

# List of Tables


3.1 General terminology . . . . . 48
3.2 Phase I specific terminology . . . . . 49
3.3 Phase II specific terminology . . . . . 50
3.4 Phase II specific terminology (continued) . . . . . 51

6.1 Sizes of each request data subset. . . . . 82
6.2 Run times and objective values of all instances, flexible meeting points 91
6.3 Run times and objective values of all instances, closest meeting points 92

# List of Figures


1.1 Differentiating service types . . . . . 6

4.1 Finding overlapping routes . . . . . 55

5.1 Sample request and its candidates . . . . . 59
5.2 Relationship between departure, arrival, pickup and dropoff times . . 61
5.3 Greedy request to route segment assignment algorithm. . . . . 68

6.1 Case study stop network . . . . . 73
6.2 Matching bus stops to the road network . . . . . 76
6.3 Edge commonness illustration . . . . . 79
6.4 Improvements to vehicle utilization with respect to demand . . . . . 86
6.5 Vehicle capacity with respect to $\lambda$ and demand . . . . . 86
6.6 Increase to vehicle utilization with respect to inconvenience . . . . . 87
6.7 Impact of flexible meeting points vs closest points . . . . . 87
6.8 Comparison of routes with and without flexible meeting points . . . . 89
6.9 Comparing run times of Phase II, Part 3 . . . . . 93
6.10 Comparing run times of Phase II, Part 4 . . . . . 93

# Chapter 1: Introduction

## 1.1 Motivation

In cities around the world, millions of people are moved daily on fixed route transit services, such as bus and rail. These options have people collect at common locations (the station or stop), and board at common times (for example according to a predetermined schedule or headway). By using common service locations and times, these modes can consolidate many trips that have similar origins and destinations or overlapping routes. This allows the operator to charge lower fares, as many individuals are transported by a single vehicle. Of course there is an inherent trade-off between the degree of this consolidation, which can be considered a measure of efficiency, and convenience to the riders. The farther the service location – and time – are from what the rider desires, the less convenient the service becomes for him or her. There is also however another inherent trade-off: that between consolidation and street congestion. Larger vehicles carry more people in less space, and moreover they streamline the path of travel causing overall travel time to decrease.

On the opposite end of the spectrum, personal modes of transportation offer service to and from the exact origin and destination of a rider, at close to exactly the time they desire to travel. Despite the apparent increased convenience to users,

a large number of small vehicles traveling results in a disorganized road network, and potentially congestion during high demand periods. This type of service is offered by a personal vehicle or a taxi. In the case of a taxi, due to the high level of individual customization of service location and time, a higher fares is required in order to be financially sustainable. Moreover, the number of vehicles needed to serve the demand must be higher than with a collective option such as mass transit.

However, it is not always possible to realize the efficiencies of fixed route transit. For example, when demand for travel is not sufficiently large, or does not follow a significantly strong spatial pattern, it is not possible to consolidate trips into useful routes. Additionally, if trips are not sufficiently regular so that they cannot be well predicted, it is not possible to come up with a suitable fixed route. Some situations which experience these types of shortcomings include off-peak travel times, so-called "reverse commutes" (whereby an individual commutes in the opposite direction of the major commuting pattern), late-night travel, or any route that is not well served by the existing transit system (for example due to the requirement for multiple transfers or an excessive number of stops). Additionally, fixed routes are slow to respond (and costly in the case of rail) to ever-evolving travel patterns, for example due to development of a new residential or employment area.

### 1.1.1 Organization of thesis

This work proposes a new service that combines the customization of a personal mode with the efficiency of a transit system, by utilizing a network of meeting points. The remainder of this chapter describes the service is further detail. The rest of the thesis is organized as follows. Chapter 2 contains a review of related literature. Because this service contains elements of the Dial-a-Ride problem, as well as bus routing problems and shared taxi or carpool problems, several areas are discussed and their solution methods assessed. Chapter 3 fully states the problem definition and constraints, and Chapters 4 and 5 detail the two-phase solution heuristic employed by this research. Chapter 6 shows the model's performance using a publicly available data set of taxi trips taken in Washington DC. Chapter 7 concludes.

### 1.1.2 Description of proposed service

This work proposes a service that bears elements of both fixed-route transit and fully personal transportation. As in personal transit, users can request a ride from and to a particular location, and specify a desired time of arrival. Requests can be made through a website, smart-phone application, public kiosk, over the phone, or via text message, and must be requested in advance for future travel. This research does not address the topic of real-time requests, in which the system responds immediately to received requests. A fleet of vehicles with homogeneous capacity is available to serve the requests, though an extension of this problem

could consider a heterogeneous fleet of vehicles. Multiple requests may be assigned to be served by a single vehicle, which allows this system to realize some of the efficiencies of fixed-route transit, and allows the operator to charge a lower fare per rider since multiple individuals will utilize the vehicle. Note, it is not necessary that the individuals have exactly the same origin and destination. Just as with a transit vehicle, passengers can board and alight throughout the route.

In urban networks – which can be difficult to navigate due to congestion and restricted turning movements – it may be quite inconvenient to service each request at their precise origin and destination location. Therefore the proposed system, instead of offering door-to-door service, goes to locations close to a passenger's desired origin and destination. These locations are chosen to maximize the overall convenience of passengers, where convenience for a particular passenger can be thought of as having two components:

1. Difference between the requested versus offered location and time of service
2. Time spent on-board the vehicle

There is an inherent conflict between these two components: generally speaking, the closer a given passenger is served to their desired locations, the longer the in-vehicle time will be for other passengers. Therefore it is important to choose service locations that are within reasonable walking distance from each passenger's desired origin and destination, while integrating well into the vehicle's route. Figure 1.1 shows an example of the proposed service. Three individuals specify an origin and destination for their trip, shown as green and red dots, respectively. The first

panel of the figure shows the case in which each request is served by separate vehicles. The second panel shows the case in which the requests are served by the same vehicle, but the vehicle picks up and drops off the passengers at precisely the customers' specified locations. The final panel shows the service proposed by this work. The individuals are assigned "stops" along the route, and are required to walk from their origin location to the pick-up stop and from the drop-off stop to their destination location. Notice that compared to the second panel, the overall driving distance is much shorter. Therefore, the slight inconvenience from walking to and from the stops is counterbalanced by the decreased time of travel. This is especially true in urban areas where the presence of one-way streets, turning restrictions, and long delays at intersections make certain locations difficult to access, while going instead to a nearby location may be much easier.

It should also be evident that under the system shown in the third panel it is possible to accommodate a larger number of passengers without causing much inconvenience to the original passengers, whereas in the second panel, each additional customer is likely to add noticeable inconvenience to the original ones. This means that the third service has the potential to serve a large number of users with many fewer vehicles. This then yields a system-wide benefit, as the resultant decrease in vehicles on the road improves travel times for all users.

Though demand and vehicle size is taken as an input to this model, the model can be run with different inputs, and from this the operator can draw conclusions about how many vehicles to deploy and their sizes. The operator can then choose the fleet size and composition subject to their budget constraint. Consider a morning

Figure 1.1: Differentiating service types

Three ways to service requests: 1) With separate vehicles; 2) With a single vehicle, offering door-to-door service; 3) Single vehicle with users walking to and from stops

rush hour period as an example, where demand is forecast to be heavy and follows a strong directional pattern. The system should try to accommodate each request as well as possible, which will mean that each passenger will have longer walks to and from their pickup and dropoff stops, in order for the demand to be met with a limited number of vehicles. Due to the high degree of streamlining, simulation results will recommend using a fleet of high capacity vehicles. On the other hand, if demand is forecast to be low, say during an off-peak period, passengers will be able to be served closer to their desired locations, as there are few other users to inconvenience. In this case, the simulation results would suggest using a fleet of low capacity vehicles. Thus in periods of high demand, the system becomes more "bus-like," with longer wait times and walk times, and more streamlined routes,

and in periods of low demand it becomes more "taxi-like," with closer service stops, smaller vehicles and more customized routing.

### 1.1.3 Benefits of proposed service

**Improved service for non-recurrent or sparse trips** Due to the high operating cost of fixed-route services, low demand periods necessitate that they run at low frequency. As an example, consider a feeder route to a mass transit station in a suburban neighborhood. If the neighborhood is large, any fixed route would have to be quite long to serve the entire area, and thus quite infrequent as well. This means that passengers would experience long in-vehicle times, simply to complete the first leg of their trip. A flexible service is much better suited here, because smaller vehicles can be used and are dispatched only as needed. This system could also be useful for those who work late at night, where the demand is not high enough to run a fixed route.

**Reduced need for taxis** This system may also reduce the need for taxis in urban areas. The case study presented in Chapter 6 of this paper demonstrates that a large number of taxi trips can be consolidated, and thus served with fewer vehicles.

**Reduced cost and pollution** Large transit vehicles contribute to air pollution and are expensive to operate, so it is not desirable to run them below their capacity, for example during low demand periods. When integrated with a demand simulation tool, the proposed model can allow transit agencies to fluctuate their

fleet size and composition throughout the day, allowing trips to be served by a fleet of vehicles that is appropriate for the current demand.

**Adaptability to changing travel patterns** While fixed-route transit modes like bus or rail can serve large numbers of passengers, their routes are constructed based on past trends and are slow and/or expensive to adapt to change. On the other hand, a flexible service continuously evolves with changing travel habits, as each route is determined by user requests. A flexible service could therefore be implemented as a way to inform transit agencies of developing patterns, and if a particular pattern becomes sufficiently strong, the transit agency could then implement a fixed route.

## 1.2 Existing Flexible Transport Modes

### 1.2.1 Informal transit

Perhaps the earliest form of flexible transportation can be found in developing cities around the world. In cities such as Lagos, São Paulo, Mexico City, Nairobi, and Manila, a large number of privately operated small to medium-sized vehicles comprise an "informal" transportation network. The vehicles are generally referred to as *jitneys*, but also are called by local names depending on the city. For example, they are referred to as *coletivos* in Mexico City and *matatus* in Nairobi. The vehicles, typically small vans, operate on roughly pre-determined routes. However, unlike formalized transit, the drivers are free to modify the routes to adapt to changing travel patterns. Since vehicles are independently operated, they are more sensitive

to market forces, so are quick to add custom routes. This, along with their ability to navigate small streets and their typically high operating frequency, make them a highly flexible and desirable mode of transportation, heavily relied upon by those whose travel routes are not well served by the formal transit network, or who unable to afford traditional transit. For example, in Mexico City, residents who live on the periphery of the city rely on informal transportation, as the formal system does not reach them. Common trip destinations include places of work, health clinics, markets, as well as train stations or bus terminals (Cervero 2000 [1]).

Another vehicle type that is used in many countries including India, Indonesia, Sri Lanka, Madagascar, Nigeria and South Africa, is the auto-rickshaw (often referred to locally as tuk-tuks) which is a three-wheeled vehicle that seats 2-4 passengers. A bike-powered version called a pedi-cab also exist in some cities such as Yogyakarta Indonesia, and Dhaka Bangladesh. Unlike van service, which is well suited for commuting trips between the city and periphery, rickshaws offer convenient service for short trips within the city. However, since their service is decentralized, their large number can increase the level of congestion on the streets, and they may at times may be unsafe to passengers, especially those unfamiliar with the city.

Despite its flexibility and low fares, informal transit has several drawbacks. The vehicles are often old and in poor condition, causing them to operate at slow speeds and to produce a considerable amount of noise and air pollution. Additionally, since drivers are independent, they compete for passengers, which can incentivize aggressive driving behavior. Competition also means that people who live

in areas of low demand density may not be served, as drivers gravitate towards higher-traveled corridors. Another issue is that the lack of any central organization means routes are often undocumented, and users rely on local knowledge to learn which route they need to take. A handful of projects have utilized GPS technology with the help of a large number of university students to systematically map informal transit routes. These include a project by the World Bank and AusAid in Manila [2], the "digitalMatatus" project by the University of Nairobi's Computing for Development lab and MIT's Civic Data Design Lab [3], and the AccraMobile project in Accra, Ghana [4]. However, with the exception of the few cases in which maps have been painstakingly created, the majority of cities lack central knowledge of the routes served.

### 1.2.2 Micro-transit

Informal transit offers a purely organic form of transportation that can roughly accommodate and adapt to the majority of travel needs, though it is not without its drawbacks. Recently in the US, due to improved mobile phone technology, several so called "micro-transit" companies have started, which aim to mimic the flexibility of informal transportation networks, while using technology to alleviate issues of coordination, reliability, and safety. Two such companies include *Chariot* [5], operating in San Francisco, and *Bridj* [6], operating primarily in Boston, and recently in Washington DC. These companies collect requests for trip origins and destinations from users of their mobile app, and use these to derive a set of routes, both

companies serving just the weekday commuting hours. Passengers are required to purchase tickets in advance of their trip. Routes remain fixed for several weeks or months, and over time, in response to users submitting requests on their website, new routes are added, modified, or dropped. In both Boston and San Francisco, the respective services are quite popular since they have identified routes that are not well-served by existing transit.

A similar service has also been employed extensively by transit agencies in China, beginning in the cities of Qingdao and Beijing in 2013 [7]. The so-called "customized bus" has been praised for significantly decreasing commuting times, as the buses are allowed to travel in dedicated lanes. There are several variants present, including buses for school children and buses that feed transit stations. As of April 2015, customized bus service was operational in 22 Chinese cities and underway in eight more.

### 1.2.3 Casual carpooling

Similar services to informal van transportation also existed in the US and were known as *dollar vans*, but these were banned in the early 1900's. However, other organic modes of shared transportation are still in use today. In the suburban region surrounding Washington DC, a form of carpooling referred to as casual carpooling, or by its colloquial term "slugging," arose in the 1970's from commuters traveling to the Naval Research Lab. Drivers on their way to work who wish to take advantage of high occupancy vehicle (HOV) lanes on the congested I-66 and I-95 freeways stop

at predetermined collecting locations ("slug lines"). Upon arriving, they announce their destination (which is often also among a set of pre-determined locations), and any travelers waiting may board. Money is typically not exchanged, which is a cherished characteristic of the service. The Washington DC area currently has a fairly sophisticated slug line network used by thousands of riders per day. An online forum provides a means for riders to discuss slug line locations and suggest new routes. Thus the service achieves a degree of flexibility to travel patterns. Sluglines are also run in Houston and San Francisco [8].

### 1.2.4 Web-based carpooling

Several websites and mobile apps have been developed in the past few years to better facilitate the benefits of carpooling. The website "Bla bla car" [9] allows drivers, typically making intercity trips, to post their route and an optional fare, and for others to join their ride. The app Carma [10] offers a similar service, but focuses on commuting. Drivers post the time and location of trips they make, and others who need a ride can request to travel with them. The transportation network company Uber has recently begun to offer a product in some cities called uberCOMMUTE [11], with the same functionality as Carma except that riders and drivers are automatically matched and the fare is set by Uber, not the driver.

### 1.2.5 Shared taxi

Recently, two major transportation network companies Uber and Lyft have launched a shared version of their taxi service: UberPOOL [12] and Lyft Line [13], respectively. Drivers are assigned requests in real time, and when possible, travellers going in the same direction share a portion of the ride. Because the service uses small vehicles, at most three passengers can be matched into a shared ride. There is no notion of a meeting point; pickups and dropoffs occur at the exact requested locations.

### 1.2.6 Vanpools

Vanpooling is a service, typically organized by an employer, in which employees can register for a shared ride to work with coworkers. Additionally, many transit agencies offer vanpools, including those in Washington DC, Virginia, Minneapolis, King County, Houston, Dallas, Austin, and many others. The vans are allowed to drive in HOV lanes when available, and members are given one or more "free ride home" benefits, in which the agency will pay for a return trip on another transportation mode, in the event of an emergency that leads the individual to miss their van home. Airport shuttles, which pick up several passengers on the way to the airport, or pick up passengers from the airport and take them to their destinations, can be classified as another type of vanpool, since either the origin or destination of trips is the same for all passengers.

### 1.2.7 Dial-a-Ride service

Dial-a-Ride service arose from the need for a transportation mode for elderly and disabled individuals, who find it difficult or are unable to use conventional transportation modes. Users tend to need special care, such as the requirement of a wheelchair ramp or other assistance. Under the Americans with Disabilities Act (ADA), public transit agencies that operate fixed route services are required by law to provide complementary *"para-transit or other special service to individuals with disabilities that is comparable to the level of service provided to individuals without disabilities who use the fixed route system"* [14]. Para-transit service commonly offers door-to-door trips made by large vans that are shared with multiple passengers. Typically, ride reservations must be made anywhere from one to three days prior to the trip in order for the agency to develop the schedule for rides. Requests are traditionally made over the phone, giving the service its name "Dial-a-Ride." Passengers specify their origin and destination of travel, what type of accommodation they need (for example, a wheelchair ramp), and their desired time of travel: either "leave at" or "arrive by." Thus the service is fully customized to the needs of the users; unlike micro-transit or casual carpooling, passengers are picked up and delivered from their request point of origin and destination, and are not asked to use common collecting points. Consequently, in-vehicle time for passengers can be high, and only a few individuals can be served per van.

### 1.2.8 Flexible route bus

As a compromise to this shortcoming of the Dial-a-Ride service, several cities offer some form of "flex bus" service. In general the idea is to have a bus with a fixed route, but that can deviate somewhat from this route. The way in which the deviations occur can be used to categorize the different variants of this service. A 2004 report by the Transit Cooperative Research Program [15] interviewed over 1,100 transit agencies and categorized the flexible services offered into six categories: route deviation, point deviation, demand responsive connector, request-stops, flexible-route segments, and zone route.

With a route-deviation system, which was the most common form in their survey, the bus travels along a pre-specified route, but has a margin around the route of a specific width in which it is allowed to serve custom requests. An example is the 570 bus in Salt Lake City, Utah, whose route has a deviation radius of $\frac{3}{4}$ of a mile. It allows for two custom requests to be served per trip, and the requests must be made in advance. With a point-deviation system there is no pre-specified route, but rather a set of available points from where individuals can make requests. The vehicle may serve the requests in any order. A demand-responsive connector serves requests on-demand that begin or end at specific transit stations. A request-stop system is similar to route deviation, but instead of being allowed to request service in any area surrounding the route, off-route service is only available at a set of pre-specified points. Flexible-route segments refers to a conventional bus route with one or more segments operating on a demand-responsive basis. In a zone route system,

the vehicles travel along a corridor and service requests on demand, however they are held to pre-specified arrival times at the beginning and end of the route. For an extensive list of examples of these services, see [15].

# Chapter 2: Literature Review

## 2.1 Dial-a-Ride problem

The most similar established problem to the one presented in this work is the Dial-a-Ride problem (DARP). This problem arose in the 1970's out of research done by the cities of Rochester, New York and Dade County, Florida [16]. The problem arises from the need for transportation services for elderly and disabled individuals, who find it difficult or are unable to use conventional transportation modes. Users tend to need special care, such as the requirement of a wheelchair ramp, or the need for an assistant.

Para-transit service typically consists of door-to-door trips made by large vans that are shared with multiple passengers. Typically, ride reservations must be made anywhere from one to three days prior to the trip in order for the agency to develop the schedule for rides. Passengers specify their origin and destination of travel, what type of accommodation they need (e.g. wheelchair, sign-language interpreter), and their desired time of travel: either "leave at" or "arrive by." While seemingly identical to the problem proposed by this research, the crucial difference is that the origin and destination points of each passenger are fixed, as it is a door-to-door service. The feasible region of the proposed service is therefore larger, since it

considers multiple possible pickup and dropoff locations for the passengers.

The Dial-a-Ride problem is a more constrained version of the Pickup and Delivery Problem with Time Windows (PDPTW). In the PDPTW, a fleet of vehicles needs to be routed to pick up and drop off goods from specific locations. Unlike the standard Vehicle Routing Problem (VRP) – where vehicles are required to bring goods to customers from a central depot – the PDPTW contains both pairing and precedence constraints: if a good is picked up by a vehicle it must be delivered by that vehicle, and the pickup location must be visited before the delivery location. Additionally, throughout every step of the route it must be checked that the load of goods on the vehicle does not exceed the vehicle's capacity; whereas in the VRP, the vehicles are loaded at the depot so the capacity constraint only has to be enforced one time. Each request has a desired time window for pickup service and a desired time window for delivery service. If these time windows are treated as "hard" constraints, the vehicle is not allowed to service a request late, and any solution with such a schedule will be considered infeasible. If instead they are "soft" time windows, solutions are allowed to violate the time windows, but incur a penalty in the objective function for doing so. It is typically assumed in either case that early arrival is permitted, but the vehicle must idle until the opening of the time window. Without this assumption many solutions become infeasible.

The DARP has an additional constraint, which is due to the fact that the "goods" being transported are people, so they must not remain on the vehicle for an excessive amount of time. Therefore a maximum ride length constraint is typically included. For further description of the many variations of routing problems and

their formulations, see Parragh, 2008 [17].

Lastly, it should be noted that there are two versions of the DARP: the static case and the dynamic case. In the static case, all requests are made before the planning horizon, for example an hour or a day in advance. The problem is then to determine the size and composition of the fleet, and to integrate all of the pre-specified requests into routes. Since the requests are known in advance and there is usually a sufficiently large amount of time available for computation, research on the static case tends to focus on exact methods or high performance meta-heuristics, as these methods produce optimal or near-optimal solutions at the expense of running time. In the dynamic case, some requests may be made in advance, but the system also responds to requests in real time. Since users must receive a response quickly about whether their request can be successfully accommodated, the dynamic case favors solution speed over quality. Research in this area therefore tends to focus on heuristic solutions, as will be shown in the section that follows.

It should now be apparent that the Dial-a-Ride problem is similar to the one addressed in this paper, with the difference that passengers are not necessarily picked up and dropped off precisely at their desired locations; instead they may be served close to these locations, in order to offer a more efficient overall service. A second difference is that in the problem addressed here, each passenger is assumed to be homogeneous, i.e. every seat in a vehicle is the same, whereas models specifically related to para-transit must consider the individual needs of each passenger. A wide body of literature exists on the Dial-a-Ride problem, and is summarized in the following sections.

### 2.1.1 Exact solution methods

A number of studies have used exact techniques to solve the Pickup and Delivery Problems, which is a less restricted version of the DARP. For the sake of brevity, those works will be left out of this review, and included will be only those that focus specifically on the DARP. For a survey of the literature regarding the Pickup and Delivery problem, see Parragh 2008 [17] and Berbeglia et al. 2010 [18].

Being a generalization of the Pickup and Delivery Problem, the DARP is NP-hard. The first exact solution to the DARP with time windows was developed by Psaraftis 1983 [19], which used a dynamic programming method to solve the single-vehicle case when there are fewer than 10 customers. Desrosiers et al. 1986 [20] used the column generation method to solve the single-vehicle problem with 40 customers.

Cordeau 2006 [21] presents a linear programming formulation for the static multi-vehicle DARP and develops a branch-and-cut solution method. It is able to achieve a reasonable running time by performing a pre-processing phase, which reduces the initial problem size, and generates several sets of valid inequalities in order to strengthen the LP relaxation at each of the nodes of the enumeration tree. The "families" of valid inequalities include: bounds on the load and service time variables, sub-tour elimination constraints, capacity and precedence constraints, generalized order constraints, order-matching constraints, and infeasible path constraints. Checking for violated constraints requires solving the corresponding separation problem, and to do this the authors employ heuristics, including a tabu search algorithm based on Augerat et al. 1999 [22]. The branch-and-bound algorithm with cuts incor-

porated requires significantly lower CPU time: an average of 1.81 minutes compared to 25.93, when considering the instances with 24 or fewer nodes. It solved a 20 node case in 42 minutes. The authors apply a similar model to the Pickup and Delivery Problem with time windows in Ropke 2007 [23], with the addition of new valid inequalities. In one variant of their model, they allow for requests to contain up to six passengers, and use vehicles with capacity of six. The largest instance of the PDPTW they solve has 96 requests. They also applied the model to a DARP of a similar size as their previous work.

Liu et al. 2015 [24] use a branch-and-cut formulation and develop a set of valid inequalities to solve a Dial-a-Ride problem that introduces constraints encountered in the real world. These include heterogeneous passengers who may require different accommodations such as wheelchairs, special seats, or stretchers. This in turn introduces the need for a heterogeneous vehicle fleet, in which each vehicle has varying capacities for each of these facilities. The largest instance solved to optimality had 22 requests and solved in 35 minutes.

### 2.1.2 Insertion heuristics

Early methods for solving the Dial-a-Ride problem relied on greedy insertion algorithms. For example, Jaw et al. 1986 [16] uses the "parallel insertion" algorithm. Parallel insertion algorithms can produce better solutions than sequential ones, which only consider one vehicle at a time. The parallel insertion algorithm is as follows. Pickup and dropoff time windows are derived for each request, based

on their requested pickup or arrival times, and the maximum ride time function, which relates direct travel time to the maximum allowable travel time in the shared service. The requests are then sorted by their earliest pickup times in ascending order. A vehicle is picked at random and the first two requests are added to it, and ordered in the least-cost way (Li and Quadrifoglio 2010 [25]). Next the third request is considered. If it can be inserted into the current route while respecting time windows, load capacity, and maximum ride time restrictions, then it is added in the least cost way. If not, another vehicle is selected to serve it. Then the fourth, fifth, etc. requests are considered for insertion into the existing routes, each time taking the previous ordering of stops as fixed. Since the number of vehicles is limited, if there are no vehicles available for which it is possible to serve a particular request, the request is "rejected" – it cannot be served. The algorithm is myopic – since once a ride is inserted to a route, its order with respect to the other requests is fixed – and thus potentially requires more vehicles to serve the requests than necessary. However, this method is able to solve large-sized problems (over 2,000 customers and 20 vehicles) in a reasonable amount of time. The authors note it can be made less myopic by instead of processing one request at a time, simultaneously processing a given number of requests, though this causes an increase in run time.

Diana and Dessouky 2004 [26] developed a regret-based model to address the myopic nature of the basic parallel insertion algorithm. The regret is calculated as the cost of inserting a given request in the current step as opposed to waiting. They are able to solve instances with up to 1,000 customers, while requiring significantly fewer vehicles, though it requires longer computation time since computing

the regret values is an $O(n^3)$ operation. Luo and Schonfeld 2007 [27] provides an improved algorithm by introducing a rejected-reinsertion heuristic that is able to handle a problem of equal size but requires shorter computation time. It produces better results in terms of number of vehicles needed, vehicle distance travelled, and vehicle idle time. Upon a request being rejected, the neighbors of that request (neighbors being defined to include spatial and temporal information) are each tentatively removed from their routes and inserted elsewhere. The least cost reinsertion is then performed. This algorithm leads to a 17% reduction in the number of vehicles needed as compared to the basic parallel insertion method.

Xiang et al. 2006 [28] develop a heuristic to solve a DARP that incorporates real world constraints, such as a maximum tour length (so no driver's shift is too long), built-in breaks for drivers, and drivers being compatible with the passengers (in terms of special training needed for handicapped or elderly users). An initial solution is constructed via a cluster-first sweep-second operation. The remainder of the algorithm includes an improvement phase, a local search strategy, a diversification strategy, and an intensification phase. The best performing version of the algorithm is able to solve an instance with 200 customers in about 20 seconds. However, when the number of customers reaches 1,000 or 2,000, the run time increases to several hours.

Häme 2011 [29] considers the single vehicle DARP and defines very narrow time windows for each request. He defines the notion of a priori infeasible: the move from $i$ to $j$ is a priori infeasible if it is not possible to leave $i$ at its earliest pickup time, and arrive at $j$ before its latest pickup time. These can be systematically calculated and

stored to determine whole clusters of mutually infeasible requests. Next, customers are inserted on the route one by one, and at each step all feasible orderings are considered, thus avoiding the myopic characteristic of the basic parallel insertion. However, the algorithm relies heavily on the narrow time windows (which restrict many insertions due to infeasibility) in order to run in a reasonable amount of time. The authors note that the time windows can be enlarged while maintaining a good run time by limiting the number of candidates considered at each step. The largest instance solved had 50 customers.

Coslovich 2006 [30] studies a version of the dynamic Dial-a-Ride problem with time windows. In the scenario studied, the vehicles are assigned to requests booked in advance, so it is based on the static problem. However, a new passenger may approach a vehicle on the street and request a ride in real time. The paper develops an algorithm that can tell the driver whether or not to accept the ride using a two-phase insertion technique. The first phase is run offline while the car is driving and creates a neighborhood of routes around the current route. The second phase is then run when a new request arrives, and tells the driver whether or not it can accept the request.

### 2.1.3 Tabu search heuristic

Several works have employed the tabu search (TS) algorithm to solve the Dial-a-Ride problem. Cordeau and Laporte 2003 [31] were the first to apply it to the DARP, though it had previously been applied to the Vehicle Routing Problem

[32], [33]. An initial solution is first found, and then the algorithm moves to a new solution in the neighborhood of the original. Solutions that contain too many similar characteristics to the previous solution are temporarily declared "tabu," and are unable to be used for a certain number of iterations; this allows the algorithm to explore the search space and avoid getting trapped in local extrema. A continuous diversification measure is also used to avoid getting trapped. An important feature of tabu search is that neighbors of a solution need not be feasible – that is, they may violate ride time or vehicle load constraints – but this is in fact a major strength of the tabu search (this also makes it trivial to come up with an initial solution). The objective function is a weighted sum of the total vehicle distance travelled, the excess ride time experienced by the passengers (travel time in excess of the direct trip length), violation of the load constraint, and violation of the time window constraints. The weights fluctuate over time to allow a more thorough exploration of the search space.

Additional papers have advanced the use of the tabu search heuristic for the DARP. Attanasio et al. 2004 [34] analyzes the success of using various tabu search algorithms developed for the static case of the DARP to solve the dynamic case. Though the TS algorithm is flexible and provides good solutions, its run time can be long (on the order of hours). This is acceptable when requests are made in advance, but in a dynamic setting, it is necessary to return a response in a matter of seconds. The authors consider techniques from parallel computing to decrease solution time. An initial solution is constructed by solving the static problem on any advance orders. Then upon the arrival of a real time requests, several parallel

processors insert the request into different random locations in the solution, and then run a tabu search to obtain feasibility. This process is capped at 30 seconds of run time.

Berbeglia et al. 2012 [35], considers the dynamic DARP, in which requests must be served in real time. It is a "hybrid" algorithm, as it combines a tabu search with constraint programming, which is a technique used to find and remove many infeasible variables when the search space is large. The constraint programming technique used is outlined in greater detail in Berbeglia et al. 2010 [36]. The model is tested on generated data, as well as a real data set from Denmark containing 200 customers.

Kirchler and Calvo 2013 [37] use a granular tabu search on the static problem, which differs from the classic tabu search in its increased focus on the local search step. Working with the assumption that optimal routes tend to not have very long edges, only moves are allowed that have reduced costs lower than a certain threshold value, which is gradually increased if no candidates exist.

## 2.2 Shared taxi problem

The shared taxi problem bears high resemblance to the Dial-a-Ride problem, though there is no heterogeneity among passengers, it almost exclusively considers the dynamic case, and it places a greater emphasis on large-scale instances. Users submit taxi requests through an online application and are matched to a taxi, which may contain additional passengers, or pick up passengers along the way.

Lee et al. 2005 [38] present a simple algorithm for integrating demand responsive service with the mass transit system in Taipei, focusing on rural areas. Riders specify their origin location, time of travel, and which transit station they need to access. The system is parameterized by an acceptable waiting time (for example, 5 minutes). The dispatching system checks which vehicle is closest to the location of the request. Once the vehicle is selected, it is checked if the request can be inserted into the vehicle's schedule. They performed a case study on an area with a 3 km radius surrounding Tinghsi MRT station. The middle 300 meters was removed from the service area, as it was assumed that it is better for users to simply walk to the station when they are within that distance from the station. It considered an instance with ten taxis and seven requests (each request could contain up to 3 persons).

In Ma et al. 2013 [39], a dynamic taxi sharing system is proposed, which has the ability to serve 720 thousand requests per hour. Every 20 seconds, all taxis in the system report their location, trajectory, and available capacity to the processing center. Upon receiving a request for a ride, the processing center determines which taxis are in a suitable vicinity of the request pickup location, determined by their respective distances to the location and the passenger's desired pickup time. Since it is prohibitively expensive to calculate a potentially large number of shortest paths for each request, the system uses a pre-calculated distance matrix, which is defined over a grid-partition of the road network. This greatly decreases the time needed to filter the taxis to those near the pickup location. Once the candidate taxis are determined, a scheduling module is run to find the best way to insert the request

into the taxi's schedule. Infeasible insertions – those that violate time windows or capacity constraints – are eliminated. The taxi with the most suitable service time for the passenger in question and the passengers already on-board is taken. While their approach is greedy, the authors argue that on-demand ride-sharing is inherently greedy: customers are expecting to be served as quickly as possible. They use a training set of 33,000 taxi trips in Beijing to simulate a stream of requests. They find that their ride-sharing system can serve 25% more passengers that a system without ride-sharing, and that the amount of miles saved leads to a reduction of 120 million liters of gasoline annually. Moreover, taxi drivers can increase their profits by 19% when the ratio of vehicles to request is 6 to 1.

The work proposed by this paper differs from Ma et al. 2013 in several ways. The first is that the research here solves a static optimization problem. In the dynamic setting, it is possible to filter the available taxis to only those that are near the customer, thus greatly reducing the size of the feasible region. However, in a static setting, any assignment of request to vehicle is possible, meaning the feasible region is much larger. Additionally, this model allows for the use of larger vehicles to serve requests. When the vehicle capacity increases, the scheduling module that determines where to insert the request in the schedule becomes more difficult, since the number of possibilities grows exponentially. Finally, the presence of multiple possible meeting points multiplies the number of options to consider.

## 2.3 Carpooling problem

The carpooling problem is similar to the shared taxi problem, except that the driver himself has a destination and a desired time of arrival. When an individual is driving to work, they may desire to pickup one or more additional passengers, either to offset costs (for tolls or gas), or to be able to use high occupancy vehicle lanes on highways. The matches can be made either in real time (the companies Waze, Lyft and Uber all have carpooling options; others include Carma, Flinc, Carticipate, EnergeticX/Zebigo, Avego, and Piggyback), or in advance, for example through an employer program.

Agatz et al. 2012 [40] provide a review of the literature on dynamic ride-sharing. They categorize different ride-sharing schemes with respect to several factors, such as how costs are shared among the passengers, whether passengers may be assigned to multiple drivers (i.e. their journey involves a transfer), and whether the ride-sharing system is integrated with other modes, such as mass transit. Additional topics include whether future demand is anticipated by the system, and whether to include monetary incentives to encourage individuals to use ride-sharing systems.

Amey 2010 [41] demonstrated that carpooling among employees of MIT could reduce the number of vehicles needed by between 9 and 27%, depending on the drivers' preferences for detours. It considers a match between at most one rider and driver.

Mahmoudi 2016 [42] presents a unique, yet computationally intensive exact solution method for the ride-sharing problem. Drivers announce their origin and

destination and are matched to riders in real time. They begin with the network formulation of Cordeau et al. 2006 [21], with additional nodes defined for the passenger and drivers' origin and destination locations. They then create an expanded network, which has one node per each possible vehicle state, where a state is defined as the passengers currently in the vehicle. For example $(p_1, p_2, \_)$ is the state corresponding to passenger 1 and 2 in the vehicle, and the third seat empty (cars are assumed to have 3 available seats). Then feasible state transitions are identified, based on time window and precedence constraints. The method relies on tight time windows to reduce the size of the feasible region. On this new network, the problem reduces to a shortest path calculation which can be solved using a Lagrangian relaxation formulation with dynamic programming. The benefit of this approach is that the shortest paths can be computed in parallel. The author solves instances on the cities of Phoenix and Chicago, with 60 and 50 passengers respectively, and each with 15 vehicles. The optimal solution was found in just under two hours, however it should be noted that a particularly powerful computer was used (128 GB of RAM).

Hosni et al. 2014 [43] also solved the ride-sharing problem using Lagrangian relaxation, in a way that allows for the decomposition of the problem into separate sub-problems. They propose two heuristic methods. They solve an instance of the dynamic problem on the Manhattan road network with 20 taxis over 2 hour period. The largest instance considered had 60 requests, and when the heuristic was used, a solution was found in 1-10 minutes depending on the instance parameters.

Agatz et al. 2011 [44] present a ride sharing system where drivers and riders are matched on short-term notice (for example a few minutes). Users of the system

specify whether they intend to drive or take a ride, and enter their origin, destination, and parameters related to the time of travel. For simplicity, the authors assume that a driver will make only one pickup and dropoff stop along their journey. A driver may serve multiple passengers, but only if the passengers are travelling to and from the same location. An additional restriction is that only trips that represent a cost savings for both the rider and the driver are considered. The requests are made with short-notice, so in order to increase the chances of finding a match between a rider and driver, a rolling horizon is used, so that all incoming requests over a period of time are considered, up to a certain time limit, after which the selection is finalized. Their matching optimization algorithm is run at pre-specified intervals in order to match the requests in the current horizon. They use the 2008 travel demand model from the Atlanta metropolitan region to develop realistic instances. The model is solved as a maximum-weight bipartite matching problem with the objective of maximizing the sum of driver and rider benefits, defined as a network flow problem and solved in CPLEX. The largest static instance solved contain 29,000 requests and solved in 78 seconds. They however note that the online system with a rolling horizon approaches the success of the offline benchmark: the offline model has a 60% match rate, and the rolling horizon model has a 58% match rate, and both yield about a 19% reduction in vehicle miles traveled. When increasing the flexibility of riders and driver, in terms of wait time and acceptable route deviation respectively, the match rate increased up to 73% in one instance.

They also develop a greedy algorithm which picks the match with the largest savings and then fixes this value in the solution. Not surprisingly, the match rate is

much lower for the greedy algorithm, around 28% in most instances.

Stiglic et al. 2015 [45] show the benefit of using meeting points in the driver-rider matching problem, using the travel demand data from Agatz et al. 2011. Instead of servicing the passenger at their desired origin and destination, a pickup and dropoff point is used that is within 11 minutes of walking distance from their requested locations. The benefit is decreased driving distance, at the expense of walking time for the passenger. Indeed the results show that driving time increases by only 1%, but total travel time experienced by passengers increases by 12% due to walking, in comparison to the case without meeting points. However, the system benefits as a whole, because the match rate increases by 6.8 percentage points. However by including meetup points in the solution, the solution time increases from 150 seconds to 10 minutes, due to increased complexity.

This is perhaps the closest existing research to the work presented in this paper. The core of this research is to utilize meeting points to increase the efficiency of ride-sharing. There are however two differences with respect to Stiglic et al. 2015. The first is that their work considers a carpooling application, in which the driver has a fixed destination and arrival time, which restricts the set of feasible matches compared to the case when drivers' routes are open. The second is that it only allows for one pickup and dropoff stop in a journey. The results of this paper are promising, in that they indicate that meeting points can indeed increase ride-share efficiency, yet they point to the need for a different solution method that is able to handle the increased number of feasible solutions.

Stiglic et al. 2016 [46] measure the effect of an increase in flexibility on the

part of the drivers – in terms of acceptable route deviation – and of the passengers – in terms of acceptable waiting time. They use the matching algorithm from Agatz et al. 2011 on the Atlanta metropolitan region demand data set. The goal is to be able to inform ride-sharing planners when driver and rider flexibility will have the greatest impact on system performance. They find that even small increases in flexibility can have a large impact on the driver-rider matching rate, especially when the request volume is low.

## 2.4 Ride sharing with transfers

A few papers have considered the possibility of a ride-sharing system in which passengers transfer between vehicles. Clearly this is a significantly more complex problem than the single vehicle problem. However it can potentially improve the efficiency of the system. For example, short routes in small vehicles can be used to feed into a route on large vehicle. The work presented here does not consider the multiple-hop scenario.

Gruebele 2008 [47] outlines a vision of a ride-sharing system in which users may share with multiple drivers to complete their trip. By using multiple vehicles, the likelihood of completing the trip as a shared ride increases; for example, even a driver whose destination is far from the rider's destination, may still be able to offer the individual a ride for part of the trip. The road network could potentially identify certain intersections with high probability of drivers passing by as "hubs" in the network. The document is only an outline of the potential benefits from

multi-hop ride-sharing, and does not present a solution method.

Herbawi and Weber 2011 [48] solve the multi-hop ride-sharing problem using an evolutionary algorithm. They formulate the problem as a multi-objective minimization problem, where the objectives are cost, travel time, and number of vehicles needed. In order to obtain the Pareto Frontier (the set of solutions for which it is not possible to improve one objective without worsening another), they use the Nondominated Sorting Genetic Algorithm (NSGA-II), which is a fast greedy genetic algorithm. They develop their own genetic operators for crossover and mutation.

To test their methodology they create a network with 41 nodes, and generate instances with between 100 and 500 requests. They used a population size of 250 and maximum of 100 generations, which led to 25,000 calls to the objective function. The run time of their method is several seconds even for the largest instance. They compare the method to a Generalize Label Correcting algorithm (GLC), which is a deterministic method for discovering the Pareto frontier. For the largest instance, the GLC found a solution in 2 minutes.

## 2.5 School bus routing problem

Another problem similar to the one proposed here is the School Bus Routing Problem (SBRP). A fleet of school buses is required to pick up students, who have been assigned to stops near their house, and bring them to their school by a certain time. The stop selection aspect is similar to the work presented here, since the students' origin locations are known (their homes), but their stop assignment must be

determined, and ideally this is done concurrently with the route definition. The difference with the work presented here is that the students have the same destination, the school, and a common required arrival time.

Typical constraints in the SBRP include earliest pickup time and vehicle capacity. A variant of the problem considers the mixed-load School Bus Routing Problem, in which there are multiple schools, and students from different schools may ride together on the same bus. Clearly, the total number of vehicles required should be less than if each school used its own buses. Park and Kim 2010 [49] provide a review of the literature on school bus routing problems.

In the simplest version of the SBRP, the students are assigned to their stop locations in advance. This provides a substantial simplification, since the students' pickup locations are inputs to the model, and all that is needed is to route the buses through those fixed points. In this case, it becomes quite similar to a Dial-A-Ride problem in which the stops are the origin locations and there is a single dropoff location, namely the school (the only difference being that SBRP problems often need to consider the location of the vehicle depot). Two examples are Euchi and Mraihi 2012 [50], which uses ant colony optimization with a variable neighborhood descent improvement search, and Park et al. 2012 [51], which takes a single-load solution for the multiple-school case, and produces an improved mixed-load solution.

These works assume each student has already been assigned to a bus stop. However, there may be several locations near a student's house that could serve as a bus stop, so a desirable model would consider all of these candidate locations to determine the most efficient set of routes, instead of fixing the stops as an initial

step.

Early works in this area considered sequential approaches to the stop selection and routing problems. The method in Chapleau 1985 [52] clusters students into districts first, and then routes one vehicle through each district. Others who used similar sequential approach include Bodin and Berman 1979 [53], and Desrosiers et al. 1980 [54]. Such approaches were borrowed from location routing theory, and are referred to as Location Allocation Routing (LAR), which assigns students to stops and then derives the routes, and Allocation-Routing-Location (ARL), which defines clusters first and then derives the routes. ARL tends to produce better results than LAR.

Some works have considered the stop selection problem concurrently with route optimization. Bowerman 1995 [55] performed a clustering of student requests and then used set covering and travelling salesman problem algorithms to generate the routes. This paper also introduced a maximum walking distance for students, which determines the number of candidate stops a student has. Riera-Ledesma 2012 [56] formulate a mixed integer program to solve the school bus routing problem with stop selection, which they refer to as the multi-vehicle traveling purchaser problem. They use branch-and-bound to solve instances of varying size, but for the largest instance with 100 students a solution could not be found in under an hour. A paper by the same authors in 2013 solved the problem using column generation.

Schittekat et al. 2013 [57] develops a heuristic model to 1) choose stops for the buses to visit, 2) assign students to stops, and 3) determine the routes through the points. The algorithm is initialized with a greedy randomized adaptive search

procedure (GRASP), which performs a variant of the Clark and Wright algorithm. Each of the potential stops are initially served by a single route. Then the savings matrix between each pair of stops on different routes is computed as the time savings that would result from merging the routes. The pair of stops to be merged is chosen randomly by drawing from a distribution based on the savings values. After merging, the feasibility is checked by solving the 'student allocation sub-problem' which is assigns students to one of the stops (and therefore one of the routes) in the current solution by solving a constraint programming problem. Note, feasibility of a merge is only checked after that merge is selected, otherwise it would require solving this sub-problem many times. This process repeats, where pairs of stops are considered if one is the first stop in its route and the other is the last, and if the capacity of the vehicle would not be violated by merging the respective routes. After the GRASP solution is found, a variable deterministic descent search is performed to improve the solution. Neighborhoods of the solution are found via three operations. The first two, remove-insert within a route and remove-insert between routes are commonly used in vehicle routing problems. The third procedure, replace, is unique to the SBRP, and it removes a visited stop and replaces it with an unvisited one. The authors used their model to solve 112 different instances, in which they varied the number of students from 25 to 800, the number of stops from 5 to 80, the bus capacity of either 25 or 50, and walking distance from 5 to 40 minutes. The largest instances could not be solved in under two hours, while the small and medium sizes were solved in minutes.

## 2.6 Vanpool routing problem

A simplification of the school bus routing problem is the vanpool problem, which is to pick up passengers from a common meeting point, located reasonably close to their origin locations, and take them to a common destination, such as an office building. It represents a simplification of the SBRP because generally only one pickup is made along the route. Employers may facilitate vanpooling for their employees, which provides a cost savings for the participants, along with the ability to rotate the driving responsibility.

Kaan and Olinick 2013 [58] study the one-stop and two-stop vanpooling problem for a case study in the Dallas Fort-Worth area, where park-and-ride locations were considered as potential meeting points. They present an exact linear programming model using a three-index decision variable (an index for each passenger, origin point, and vehicle type), as well as a heuristic solution method.

The exact method was able to find a solution to the one-stop vanpooling assignment problem with up to 600 employees and 120 possible stop locations in a matter of seconds. However, an exact solution to the the two-stop model could not be found in under 12 hours. The heuristic method was therefore used, which works by combining a greedy covering algorithm with a relaxed LP solution. The relaxed heuristic is used to inform which park-and-ride locations seem most useful. The heuristic is reported to have suitably short solution time.

## 2.7 Literature review conclusion

The work presented here intends to take passengers from and to locations relatively close to their desired origin and destination. In that sense, it is similar to the Dial-a-Ride problem, but the pickup and delivery can occur at different locations than the original ones specified by the users. This allows for a more streamlined route, and the potential for multiple passengers to be served in the same location. Since stop selection is a crucial part of this system, the research also bears similarity to the School Bus Routing Problem, in the case in which the stops are not pre-assigned to students. Unlike the school bus problem however, passengers do not all have a common destination location (i.e., the school) and arrival time window (i.e., the morning bell). This means that the number of riders in a vehicle can increase and decrease throughout the journey, unlike a school bus which only accumulates passengers. It also means it is significantly more difficult to "cluster" requests, as both the pickup and dropoff location need to be considered.

The system proposed here would be similar to the mixed-load school bus problem, in which each student belongs to a different school. The mixed-load problem has been studied, but not with concurrent stop selection. It would also be similar to a multi-stop vanpooling problem where the destination is not the same for every passenger, but this has not been studied.

Carpooling and share-taxi research consider service to and from the desired origin and destination, with the exception of Stiglic et al. 2015 [45] which considers meeting points. Additionally, they consider vehicles with capacity of 3 or fewer

passengers. The work presented here allows for the use of high capacity vehicles, and offers service from meeting points, which are not necessarily at the desired origin and destination of the requests.

# Chapter 3: Solution Algorithm Overview

## 3.1 Introduction

The proposed system accepts requests for travel from the user's specified origin and destination location, and desired time of arrival at the destination. The solution algorithm presented here assumes that requests are submitted sufficiently far in advance of the desired time of travel (e.g. one hour or one day). An input to the model is a digital representation of the road network, converted into a directed graph. The edge weights of the graph can be the travel time along the edge. If desired, several versions of the graph can be used to represent periods of varying levels of traffic (e.g. a rush hour graph and an off-peak graph). However, it should be noted that since the solution algorithm here considers the scenario in which all requests are made ahead of time, it does not account for information about live traffic conditions or incidents. Such real-time changes would need to be handled in a heuristic way, and this is not in the scope of this work.

Lastly, the system requires the existence of a pre-defined network of stop locations. As an example, one might use the existing network of city bus stops. Alternatively, one could define a custom network, where the general idea is to have one stop per block in both directions of travel whenever possible. This grants the al-

gorithm the greatest flexibility to choose the most efficient set of stops when forming a route. Increasing the number of stops can only improve the solution, but comes at the cost of increased complexity to the solution algorithm. Having the stop network pre-defined is the key feature that enables the model to overcome some of the difficulties encountered by existing methods. Because they stop-network is determined ahead of time, one can analyze all possible routes in the system. Then when the schedules are being derived, the model is better informed about which routes to try to insert into the schedule. This is in contrast to a purely brute force insertion method, or a method that only looks at time windows.

**Model objective** In general, there are several objectives that can be considered for this problem depending on the needs of the system operator. In this paper, the objective taken is to minimize the number of vehicles used to serve the requests, subject to the travel time constraints of the passengers. However, another objective might be to consider the cost of providing the service: for instance, some requests may be exceedingly expensive to serve (if a particular request is distant from the majority of other requests), and therefore a cost minimization model or a model that had a budget constraint would involve rejecting some requests. Another objective would be to achieve an optimal quality of service, subject to a certain portion of the demand being satisfied. This could mean striving for the shortest total travel time, or the least amount of walking, while guaranteeing that a certain percentage of the demand be served. Moreover, any of these objectives could be used concurrently in a multi-objective framework, but this would require a more complex solution approach. As previously stated, this model tries to minimize the

number of vehicles used to serve the demand, subject to serving all of the requests, and obeying the vehicle capacity and ride time constraints.

**Model constraints** As with the Dial-a-Ride problem, time windows are specified for the pickup and dropoff of riders. The pickup and dropoff time windows are based on the user's desired arrival time, and their calculation is explained in the description of Phase II. Note that unlike the DARP, in which the pickup location is the same as the origin location and the dropoff location is the same as the destination, the amount of time spent walking from the origin to the pickup stop and from the dropoff stop to the destination must be accounted for when computing the time windows at each candidate stop.

Additionally, a constraint on the ride time for each passenger must be enforced. This can either be handled by imposing a system-wide maximum allowed ride time, or by setting the maximum ride time separately for each request as a function of the direct ride time. The simple function used in this research is to allow rides to be no longer than the direct ride time plus some value $\lambda$, a model parameter.

Constraints with respect to the vehicles include maximum load constraints, defined by the number of seats available on each vehicle, as well as precedence and pairing constraints. In this research, vehicles are assumed to have equal capacity, but necessary considerations for incorporating heterogeneous vehicles are discussed in the Conclusion chapter. This research does not address the problem of crew scheduling, in which routes are found that incorporate time for drivers to switch out or take breaks, or maximum shift constraints, which put a limit on the longest driver shift. It also assumes that the vehicles may start and end in any location,

which is to say there is no notion of a "depot." More realistic assumptions with regards to the fleet can be imposed in future work; the goal of this research is to explore the potential capabilities of such a system.

**Model formulation** The objective addressed by this work is to minimize the number of vehicles $K$ used to serve the demand:

$$\min K \tag{3.1}$$

For each request, there may be multiple possible pairs of pickup and dropoff stops. The term *request-candidate* is used to refer to a unique combination of request, pickup stop, and dropoff stop. Denote a request candidate $ra$ by this triple $ra = (r, a = (p, d))$, where $r$ is the request index, and $a = (p, d)$ is the *pickup-dropoff pair*, determined by a unique combination of pickup and dropoff stops.

The ride time constraints are represented as follows, where $T^V_{ra} = \tau^d_{ra} - \tau^p_{ra}$ is the in-vehicle time from the pickup stop $p$ to the dropoff stop $d$, equal to the dropoff time minus the pickup time assigned to $ra$. $T^{W_p}_{ra}$ and $T^{W_d}_{ra}$ are respectively the time needed to walk to the pickup and dropoff stops associated with $ra$. $\delta_r$ is the direct travel time from the origin to the destination of request $r$.

$$T^V_{ra} + T^{W_p}_{ra} + T^{W_d}_{ra} \leq \delta_r + \lambda \tag{3.2}$$

This states that the total travel time, equal to the sum of the walking from the origin to the pickup stop, the in-vehicle time, and the walk from the dropoff stop to

the destination, must not exceed the direct driving time from the origin destination by more than $\lambda$. A limit is placed on the values of $T_{ra}^{W_p}$ and $T_{ra}^{W_d}$, limiting the amount of time passengers will be asked to walk to each stop:

$$T_{ra}^{W_p} \leq \gamma_i \tag{3.3}$$

$$T_{ra}^{W_d} \leq \gamma_i \tag{3.4}$$

As previously mentioned, all requests $r \in R$ must be served. Lower and upper bounds are placed on the pickup and dropoff times, and are defined in the next section. Lastly, the number of passengers on board a vehicle at any given time must not exceed the capacity of the vehicle.

## 3.2 Solution method overview

Since the problem presented here generalizes the Dial-a-Ride problem, it would be impractical to seek an exact solution method that could handle an instance of realistic size. This paper presents a heuristic that can find good solutions in a reasonable amount of time.

The main idea of the solution method is to define an undirected graph in which the nodes are request candidates, and find clusters on this graph of request candidates that are sufficiently similar and assign them to the same vehicle in a greedy fashion. Two components determine the similarity of request candidates: 1) *spatial similarity*, referring to the location and shape of their routes, and 2) *temporal similarity* in terms of the users' requested time of arrival at their destinations and

the implied earliest departure time from their origin.

The solution method includes two phases. The first initializes the system and the second is used to respond to requests from users. The first phase consists of several calculations that may take a considerable amount of time but only need to be done once, and their results are referenced every time the second phase runs.

Since there are a discrete number of stops, the routes between all possible pickup and dropoff pickup-dropoff pairs can be enumerated ahead of time, and therefore the spatial similarity between these pairs can be pre-calculated; this is the task of Phase I. With the spatial similarity already calculated, when requests from users arrive, Phase II needs only compare their requested travel times in order to determine their temporal similarity. Then the clustering algorithm can be executed.

For simplicity, users are required to submit their requests sufficiently far in advance, and it is further assumed that all vehicles initially have no passengers on board.

A detailed explanation of Phase I and Phase II is presented in the next chapters. Several notations and terms will be used, which are first defined here for clarity.

## 3.3 Model inputs, outputs, objective, and constraints

For clarity, the following are the model inputs and outputs:

**Inputs**

- Road network: a digital representation of the road network is required, with

the cost (distance or travel time) of each link specified.

- Stop network: location and bearing of service locations. This can be taken as the existing bus stop network, or defined in another way.

- Vehicle capacity: This research assumes all vehicles are of equal size. The system operator is free to run the model under different assumptions of vehicle fleet size and vehicle capacity to compare the outcomes.

- Demand: Demand for trips is specified by individual users, and includes the origin and destination of travel and the desired arrival time at the destination. Each request is assumed to be for one person, but it would be straightforward to extend the model to accommodate requests for multiple passengers.

**Model Output**

- Request-to-vehicle assignment: Each passenger will be told the Id number of the vehicle to which they are assigned.

- Request service locations: Each user will be assigned a location for pickup and dropoff, that will be within a suitable walking distance from their requested origin and destination locations.

- Request pickup and dropoff time: Each user will be informed of the time that the vehicle will arrive at the pick up stop, and the time that it will drop them off at the dropoff stop.

- Vehicle routes: Vehicles will be given a schedule to follow, consisting of the times and locations of pickups and dropoffs.

Table 3.1: General terminology

| | |
|---|---|
| $G$ | The directed graph representing the road network. If multiple graphs are to be used for different times of day/ days of week, use the notation $G_t$ where $t$ is the time period index. |
| *users, passengers* | Refers to individuals who submit a request for travel to the system (these terms used interchangeably). |
| $r = (O_r, D_r, \tau_r^D)$ | A request. $O_r$ and $D_r$ are requested origin and destination location respectively, and $\tau_r^D$ is the desired arrival time at the destination. |
| $\delta_r$ | The direct travel time from $O_r$ to $D_r$. |
| $R$ | The set of all requests. |
| *stop, meeting point* | A fixed location that has been determined during system initialization to be suitable for service by a vehicle. Users will be assigned a pickup and dropoff stop within reasonable walking distance of their desired origin and destination. |
| $P$ | The set of all stops |
| $N$ | The number of stops, i.e. $\|P\|$ |
| *pickup-dropoff pair* $(p, d)$ | A pair of stops, which may be used to pick up and drop off a passenger. |
| $D(p, q)$ | The distance (or travel time) between stop $p$ and stop $q$. |
| $M$ | The set of all pickup-dropoff pairs which are sufficiently far from each other, i.e. $\{(p, d) : p, d \in P, \mathcal{D}(p, d) \geq \beta\}$, where the value of $\beta$ is left to the discretion of the system operator. |
| *spatial similarity* | A metric describing the compatibility of two pickup-dropoff pairs in terms of their direction and path of travel. |
| *temporal similarity* | A metric describing the compatibility of two pickup-dropoff pairs in terms of their desired travel times. |

Table 3.2: Phase I specific terminology

| | |
|---|---|
| $W$ | The spatial similarity matrix between pickup-dropoff pairs. |
| $G'$ | The modified road network graph. It is initialized to $G$, and then the edge weights are iteratively decreased based on the number of shortest paths that route through each edge. |
| $S^X(a, b)$ | The overlap measure for two pickup-dropoff pairs $a$ and $b$ |
| $\kappa$ | The number of iterations to perform in the iterative shortest path component of Phase I |
| $\alpha$ | The proportionality factor by which to reduce the edge weights in the iterative shortest path component of Phase I |

Table 3.3: Phase II specific terminology

| | |
|---|---|
| $C_\ell$ | The number of passengers vehicle $\ell$ can carry. In this research this is taken as a constant value for all vehicles. |
| $\gamma_i$ | The maximum amount of time passenger $i$ should have to walk to their assigned pickup and dropoff stops. This value can be the same for all users, though for example users with a disability may have $\gamma_i = 0$. |
| *Maximum inconvenience amount* $\lambda$ | For a request with origin $O$ and destination $D$, the amount of time in excess of the direct route time that a ride in this system is allowed to take. For example $\lambda = 10$ means that the total travel time (walking plus in vehicle time) cannot exceed 10 minutes plus the direct travel time. |
| *Candidate pickup/dropoff stop* | A pickup or dropoff stop within $\gamma$ distance of the passenger's desired origin and destination, respectively. |
| *Request-candidate* | A request-candidate consists of a request $r$ and one candidate pickup-dropoff pair |
| $RM$ | The set of all request-candidates, $\{(r, a) : r \in R, a \in M$and pickup-dropoff pair $a$ can serve request $r\}$ |
| $h$ | Similarity threshold to consider two pickup-dropoff pairs for clustering. |
| $\epsilon$ | Value of the buffer on the latest arrival time. For example $\epsilon = 2$ minutes allows dropoffs to occur up to two minutes outside of the time window. |
| $W^R$ | The similarity matrix between pairs of request-candidates. |
| $\Delta$ | The amount by which to artificially increase the similarity between request-candidates if they use the same pickup and dropoff stops. |
| $\eta_1$ | Number of neighborhood levels to take around a cluster after all cluster members have been tried. |
| $\eta_2$ | Number of consecutive infeasible attempts to insert a node before starting the next vehicle. |
| $\eta_3$ | Number of assigned segments after which to re-run the graph clustering. |
| $K$ | Number of vehicles used in the solution to serve the demand. |
| *Vehicle utilization* | The number of requests served divided by the number of vehicles used: $\lvert R\rvert/K$. |

Table 3.4: Phase II specific terminology (continued)

| *Defined for each request-candidate* $(r, a)$, $r = (O, D, \tau^D), a = (p, d)$ : | |
|---|---|
| *Departure time* | The time at which the passenger leaves from their origin location $O$ and begins to walk to the pickup stop $p$. |
| *Departure time* | The time at which the passenger leaves their origin location $O$ to begin walking to their pickup stop. |
| $\tau^p$ *Pickup time* | The time at which the passenger is picked up at their pickup stop. |
| $\tau^d$ *Dropoff time* | The time at which the passenger is dropped off at their dropoff stop. |
| *Arrival time* | The time at which the passenger arrives at their destination $D$, after having walked from their dropoff stop. |
| $T^V$ *In-vehicle time* | The time spent in the vehicle by passenger, equal to the dropoff time minus the pickup time |
| $T^{W_p}$ | The time it takes to walk from the origin to the pickup stop |
| $T^{W_d}$ | The time it takes to walk from the dropoff stop to the destination |
| *Latest arrival time* | The latest time the passenger can arrive at their destination $D$, after having walked from their dropoff stop. Equal to the desired arrival time, plus an optional small margin, $\epsilon$. |
| *Latest dropoff time* | The latest time the passenger can be dropped off at the dropoff stop, in order to be able to walk to their destination $D$ and arrive by the latest arrival time. |
| *Latest pickup time* | The time the passenger would have to be picked up at the pickup stop, if travelling on a direct route to the dropoff stop in order to arrive by the latest dropoff time. |
| *Earliest departure time* | The time the passenger would have to depart from their origin $O$ in order to arrive at their destination $D$ by $\tau^D$, assuming that the route takes $\lambda$ plus the direct ride time. |
| *Earliest pickup time* | The time the passenger would arrive, via walking, at the pickup stop $p$ if leaving her origin at the earliest departure time. |
| *Earliest dropoff/ arrival time* | It is assumed passengers do not mind arriving early to their destination, provided it is within reason. This is secured however, since the earliest departure time is fixed. Therefore the earliest dropoff and arrival times are set to the trivial value of the earliest departure time. |

# Chapter 4: Solution Algorithm Phase I: Spatial Similarity

Because users are served from a discrete set of stops, many calculations can be done in advance to speed up the passenger to vehicle assignment at run-time. This section describes a set of calculations that are time-consuming, but can be done ahead of time and their results stored for repeated use later. The goal of this phase is to calculate the amount of "similarity" that exists among pickup-dropoff pairs. These values can be stored and queried later in response to user requests (Phase II). The components of the pre-calculation phase are:

1. Pickup-dropoff pairs enumeration
2. Iterative shortest path calculation
3. Store spatial similarity matrix

## 4.1 Part 1: Pickup-dropoff pairs enumeration

In the first step, consider the set $P$ which contains the $N$ pre-defined stops. Then form the set of all pairs of stops in the system, removing stops that are too close to each other: $M = \{(p, d) : p, d \in P, \mathcal{D}(p, d) \geq \beta\}$. The planner can decide the value of $\beta$ to determine the minimum allowable ride length, and the distance

function can be defined in terms of travel length or time. Defining the number of stops in the system as $N$, then there are $O(N^2)$ allowable pickup-dropoff pairs.

It is now necessary to determine how "similar" two pickup-dropoff pairs are. A brute-force method would to look at every pair of pickup-dropoff pairs, and calculate the additional travel time incurred by inserting in the least-cost way one pickup-dropoff pair into the other. However since there are $O(N^4)$ such pairs, this method is impractical unless the number of stops in the system is small. Therefore, the next two steps use a heuristic method to implicitly identify similar routes.

It should be noted that both Luo and Schonfeld 2007 [27] and Santos and Xavier [59] compute the insertion cost for pairs of requests. This is practical if 1) the model considers live requests only, and 2) each request has only one possible pickup and dropoff location. In the work presented here, Phase I calculates the similarity between all pairs of pickup and dropoff stops. If only the live requests were considered, there would be many additional insertion costs to calculate, since each request may have multiple feasible pickup and dropoff locations.

## 4.2 Part 2: Iterative shortest path calculation

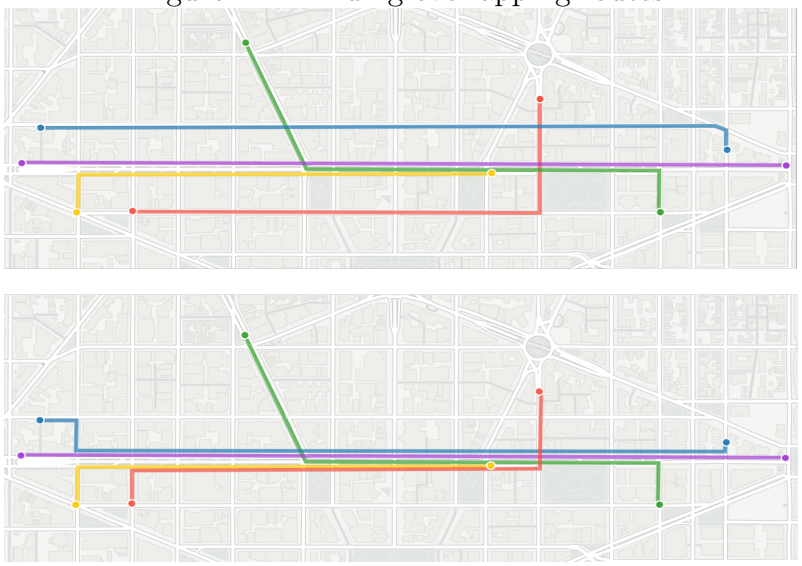
For each pickup-dropoff pair $(p, d)$, calculate the directed shortest path between $p$ and $d$, and save the edges traversed by each shortest path. To begin the first iteration, start with the list of edges used in any shortest path, and calculate the total distance of overlap among the pickup-dropoff pairs. Note, it is not necessary to consider all $O(N^4)$ pairs, since the vast majority of them will not possess any

overlapping edges. Keep track of the amount of overlap between the pickup-dropoff pairs, and note that they overlapped on the initial iteration ($k$=0). There still may be two pickup-dropoff pairs that are quite similar, but do not explicitly overlap on the first iteration, as is shown in Figure 4.1. This is the motivation behind the subsequent steps.

Next, for each edge in the graph, calculate the number of pickup-dropoff pairs that used it in their shortest path, and call this value the *commonness* of the edge. Some edges will be more common than others, typically corresponding to major roads. Reduce the cost of each edge by a factor $\alpha$ times its commonness. Call the resultant graph the *modified graph* $G'$ because its edge weights differ from those of the true graph. Then recalculate the shortest paths of the pickup-dropoff pairs in $M$ on the modified graph $G'$. The idea is that in the second iteration, pickup-dropoff pairs will be attracted to the more commonly used edges. This in turn increases the commonness of those edges further, and even more pickup-dropoff pairs are attracted to them in the subsequent iterations. A sketch of this idea is shown in Figure 4.1. Nearby routes that did not explicitly overlap in the first iteration are made to overlap by this process, making it possible to capture their similarity.

At every iteration, calculate the amount of overlap between pickup-dropoff pairs $X_k(a, b)$, defined as the sum of the length of the edges that are present in both shortest paths. When calculating the amount of overlap, use the edges of the shortest paths found on the modified graph, but substitute the true edge costs. Note that there is a corresponding iteration number $k$ that was required to achieve the overlap. The idea is that pickup-dropoff pairs that take several iterations to overlap

Figure 4.1: Finding overlapping routes



Demonstration of reducing edge costs on links with higher commonness. On subsequent iterations, more pickup-dropoff pairs then use these links, and so on.

are not as similar to each other as those that overlap early on. The module can continue for a $\kappa$ iterations, which is a parameter set by the modeler, and if desired, the factor $\alpha$ can be adjusted after each iteration.

Note that the routes calculated in each iteration of this process are merely used to determine similarity among pickup-dropoff pairs. They are *not* indicative of the route that will be taken in the final solution, as this depends on all of the requests selected to be served by a vehicle. Indeed, the routes calculated in this phase are generally not the most efficient routes, but the assumption is that if a group of pickup-dropoff pairs can be make to overlap over a large portion of their length, then they likely have many alternate routes in common as well. This point will be further demonstrated in the Case Study chapter.

After the iterations finish, the modified graph $G'$ contains routes for the pickup-dropoff pairs in $M$, but many more routes will overlap than did when using the edge

weights of the true graph $G$. At each iteration, the amount of overlap between routes is calculated, and the iteration counter $k$ at which that overlap occurred is stored. For pickup-dropoff pairs $a$ and $b$ that overlap, the overlap similarity $S$ can be defined as follows:

$$S(a,b) = \max_{k=1\ldots\kappa} \Big[ X_k(a,b) \cdot f(k) \Big] / L(a) \tag{4.1}$$

where the maximum amount of overlap weighed by the number of iterations is taken. $f$ is a function that transforms the iteration counter $k$ into a value between zero and one, and $L(a)$ is the length of route $a$. Thus $S(a,b)$ is the overlap similarity of pickup-dropoff pair $a$ to pickup-dropoff pair $b$.

Note that $S(a,b)$ need not equal $S(b,a)$, since route $b$ may be longer or shorter than route $a$, in other words $L(a)$ likely does not equal $L(b)$. For the clustering algorithm described in Phase II, it will be necessary to use symmetric costs (i.e. $S(a,b) = S(b,a)$). To accomplish this, the larger of the two values is taken.

The similarity matrix is defined by the elements $S(a,b)$ and stored for use in the second phase. If multiple graphs are used, for example, if a separate set of edge weights are used during rush hour and off-peak times, then separate similarity matrices should be calculated for each one.

## 4.3 Phase I conclusion

The spatial similarity calculation uses an iterative approach to discover overlapping pickup-dropoff pairs, and develops a measure for determining the degree of similarity between them. The method works by reducing the weights on more

commonly used edges of the graph and re-computing the shortest paths of all pickup-dropoff pairs. Doing so leads similar routes to overlap, and as the iterations progress, more and more routes overlap. This method is used in order to avoid the brute-force calculation of the insertion cost between all pickup-dropoff pairs.

There are two parameters to the similarity calculation just described: $\alpha$, the amount by which to decrease the edge cost at each iteration, as well as $\kappa$, the number of iterations to perform. These should be tested with different values, as the most suitable values will likely vary for different road networks. Increasing the amount by which edge cost is reduced at each iteration and/or performing a greater number of iterations will lead to a high degree of overlap among routes, potentially between even those that are not realistically similar. On the other hand, low values for these parameters may result in too few routes being found to overlap.

After the system has been operational and receiving user requests for some time, it will be possible to improve the initialization step of the similarity module. When first performed, edge weights are reduced based on their commonness, which is defined by how many pickup-dropoff pairs used the edge in its shortest path, with each pickup-dropoff pair being equally weighted. However, as travel patterns become known, each pickup-dropoff pair could be weighted by the frequency of its use by actual passengers. In this way, the system can "learn" to prefer more commonly requested routes as it is used.

# Chapter 5: Solution Algorithm Phase II: Request Assignment

Once the pre-calculation phase is complete, the system can go into operation and begin accepting requests. For the scope of this work, requests are assumed to arrive sufficiently in advance of the desired time of travel. Clearly, the earlier the requests are known, the greater is the potential to find highly efficient routes.

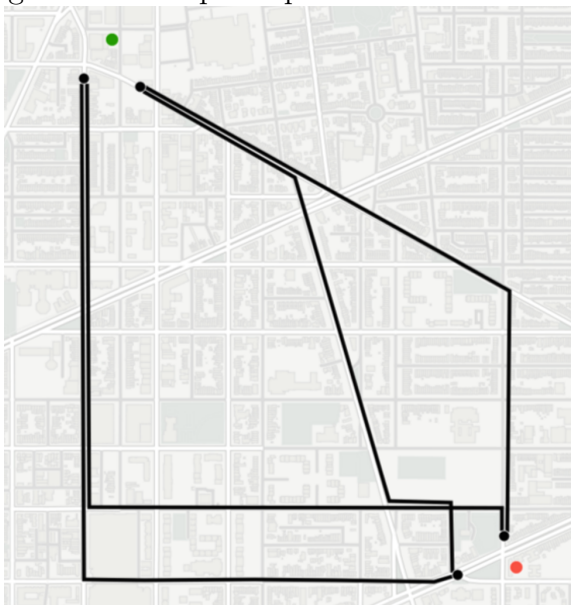
The second phase of the solution algorithm is run to match the received requests to the fleet of vehicles. It contains the following steps:

1. Identify request candidates
2. Identify similar request candidates
3. Greedy request to route segment assignment
4. Improvement heuristic
5. Route segment chaining

## 5.1 Part 1: Determine candidate pickup-dropoff pairs

Each request received will specify the user's desired origin and destination location, and desired arrival time at the destination. The set of requests can be

Figure 5.1: Sample request and its candidates



denoted as $R = \{r = (O_r, D_r, \tau_r^D)\}$. For each request, all stops within $\gamma$ distance from the origin and destination are selected as *candidate stops* for that request. An example is shown in Figure 5.1. The origin and destination locations are shown in green and red respectively. There are two candidate pickup stops and two candidate dropoff stops, meaning there are a total of four candidate pickup-dropoff pairs, which are shown as the black lines.

The walking time threshold $\gamma$ is therefore a parameter of the solution method. This parameter measures where the service is located on the spectrum between a personal mode and a collective mode. A lower $\gamma$ means passengers will be served closer to their requested locations, meaning the service will behave more like a taxi system. However, as was motivated by Figure 1.1, if demand is high this may lead to an increase in passenger travel time. A higher walking threshold gives the algorithm greater flexibility in forming routes, making the service more like a bus system,

however passengers may dislike the increased walking time. Certain passengers, for example those with a disability, could however be allowed an exception in order to be given a zero-minute walking time threshold.

The set $M$ contains all possible pairs of pickup and dropoff stops. The set $RM$ of *request-candidates* can be formed

$$RM = \{(r, a) : r \in R, a = (p, d) \in M, \text{and pickup-dropoff pair } a \text{ can serve request } r\}.$$

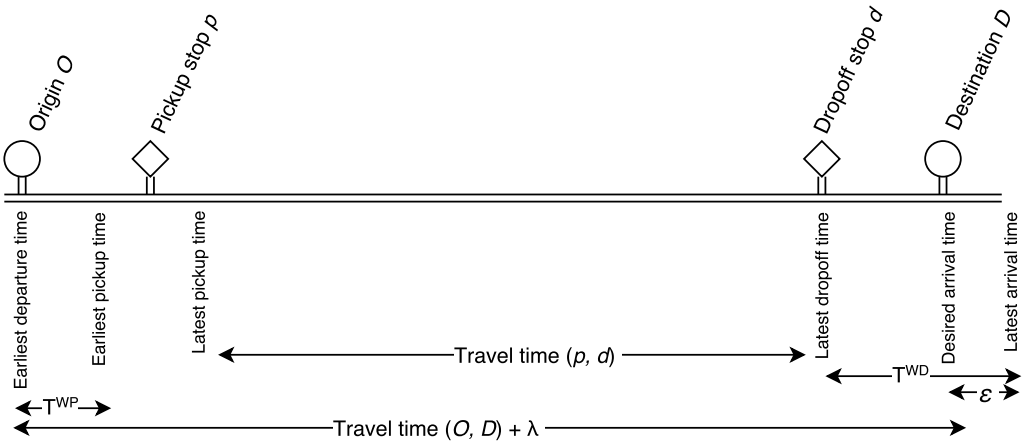
## 5.2 Part 2: Travel time overlap

If two requests are to be served with the same vehicle, it is necessary that not only their pickup-dropoff pairs are similar, but also that their desired times of travel are compatible.

For each request-candidate $ra = (r = (O, D, \tau^D), a = (p, d))$, it is possible to compute acceptable time windows for the pickup and dropoff stops, based on the user's desired arrival time at the destination. In the definitions that follow, the word *departure* refers to the passenger walking from their origin location to the pickup stop, pickup refers to when the vehicle arrives at the pickup stop, *dropoff* refers to when the vehicle drops off the passenger at the stop, and *arrival* refers to when the customer arrives at their final destination after having walked from the dropoff stop. A diagram relating the time windows is shown in Figure 5.2.

First, the maximum allowable time for the customer to arrive late to their destination must be established. One option would be to take the desired arrival

Figure 5.2: Relationship between departure, arrival, pickup and dropoff times




time $\tau^D$. However, the feasible region can be expanded by considering solutions in which the customer may arrive up to a few minutes after their desired arrival time. Call this time the *latest arrival time*, and let $\epsilon$ be the amount of time by which it is permissible to arrive after the desired time window. This is a parameter of the solution algorithm, for which different values may be tested. Generally speaking, the larger the value of $\epsilon$, the more requests will be able to be accommodated, at the expense of rider convenience.

Next, the *latest dropoff time* can be calculated as the latest time a passenger can be dropped off at $d$, in order to be able to walk to their destination $D$ and arrive by the latest arrival time. The *latest pickup time* is when the passenger would have to be picked up at $p$, if travelling on a direct route to $d$, in order to arrive by the latest dropoff time.

The *earliest departure time* is when the passenger would have to depart from their origin $O$ in order to arrive at their destination $D$ by $\tau^D$, assuming that the route takes $\lambda$ plus the direct ride time. This is based on the principal that a ride in

this service should not take more time than the direct route in excess of a certain value. For example, it might be desired that no trip take more than 10 minutes of the direct route time. In this case, the earliest departure time is set to the desired arrival time less the direct route time plus ten minutes. The amount of ten minutes is denoted as $\lambda$, the *maximum inconvenience amount*. Note that setting a high $\lambda$ value will allow a greater level of trip consolidation, but the longer trip lengths may be unfavorable to passengers. On the other hand, too low a value of $\lambda$ may lead to a high number of vehicles needed to serve the demand. The *earliest pickup time* is the time the passenger would arrive, via walking, at the pickup stop $p$ if leaving her origin at the earliest departure time.

There is no need to define an earliest dropoff or arrival time, as a customer can be assumed to always prefer an early dropoff whenever possible, provided it is within reason. This is secured however, since the earliest departure time is fixed. Therefore the earliest dropoff and arrival times are set to the trivial value of the earliest departure time.

Define the travel time range $T_{ra}$ of a request-candidate $ra$ as the earliest departure time to the latest arrival time.

Next, the similarity is measured between pairs of request-candidates. This is taken to be the spatial similarity of their routes, which is pulled from the stored spatial similarity matrix, $W$, with the additional comparison of their travel time ranges. A new similarity matrix, $W^R$ will be created considering the route-requests, and will have dimension $|RM| \times |RM|$, with one row and column for each request-candidates.

The elements of the request similarity matrix $W^R$ are computed as follows. For each request-candidate $(r, a = (p, d))$, obtain the pickup-dropoff pairs that are similar to $a$, in other words, $b \in M, b \neq a$ such that $W(a, b) > h$. The parameter $h$ can be set to 0 to include even routes that overlap slightly. For faster solution time, a higher value of $h$ can be used. Then for each pickup-dropoff pair $b$, retrieve the requests that has $b$ as a candidate, in other words $q \in R$ such that $(q, b) \in RM$. For these requests, compare their travel time ranges $T_{ra}$ and $T_{qb}$. If these time ranges overlap, $W^R\big((r, a), (q, b)\big) = W(a, b)$, otherwise 0. It should be noted that even if two requests have temporal similarity of 1, they may not be suitable for consolidation (for example, if they only overlap by a small amount, and/or during the overlap time, the vehicle would have to be in two different locations). It would be possible to make the temporal similarity value continuous instead of binary to better capture temporal compatibility. However it will be shown in the greedy assignment step that using the binary indicator still produces good solutions, and is much less expensive to compute than a continuous value. To increase the likelihood of trip consolidation, if two request-candidates use the same pickup-dropoff pair, their similarity is multiplied by $1 + \Delta$, where $\Delta$ can be set by the modeler.

## 5.3 Part 3: Greedy request to route segment assignment

Using the similarity matrix for request-candidates $W^R$, it is possible to identify clusters of request-candidates that are highly similar. The work presented here uses a greedy graph clustering algorithm, but other graph clustering algorithms

are available [60]. In this case, the nodes of the graph correspond to request-candidates, and two request-candidates are connected by an edge if they are spatially and temporally similar. Since, as previously noted, it might be the case that $W^R((r,a),(q,b)) \neq W^R((q,b),(r,a))$, the original directed graph is converted to an undirected graph by setting the arc cost as the larger of the two cost.

Details of the greedy graph clustering algorithm can be found in [61]. Upon converting the request-candidate graph to an undirected graph, each request is assigned to a cluster by this algorithm. Note that if a request has multiple candidate pickup-dropoff pairs, it may appear in multiple clusters for each of its candidates.

It is then necessary to assign time windows for each node: the pickup time window is (*earliest pickup time*, *latest pickup time*), and the dropoff time window is (*earliest dropoff time*, *latest dropoff time*). Then a tentative service time $t_x$ is assigned to each pickup and dropoff. This value will be updated throughout the algorithm, and is initialized to the earliest pickup time for pickups, and the latest dropoff time for dropoffs.

The next step is to identify the importance of each node within its cluster. Many *graph centrality* metrics exist that can be used to calculate this [62]. This research calculates the *weighted degree* of each node, which is equal to the sum of the costs of all edges incident to the node. The idea is that the node with highest weighted degree will be more likely to be similar to other nodes in the graph. Starting with the largest cluster, the request-candidate with the highest weighted degree is assigned to a route segment. After the first request-candidate is selected, any other candidates for that request are removed from the solution pool. Then the request-

candidate with the next highest weighted degree is attempted to be inserted. The insertion test module is described next. If the insertion is feasible, then the request-candidate is added to the schedule, and if it is not, it is removed from the list of request-candidates to try. After a successful insertion, the weighted degrees of the nodes in the graph are recalculated, since some nodes have now been removed.

Once all of the nodes in the cluster have been tested, any nodes that are adjacent to a node in the cluster, referred to as the *cluster neighbors*, are then tested, in decreasing order of the highest cost edge weight that connects the neighbor node to the cluster. After these have been tested, any neighbors of the neighbors are tested, and so on for $\eta_1$ iterations. The number of neighbor levels to search $\eta_1$ can be set by the modeler. At any point, if $\eta_2$ consecutive infeasible attempts to insert a node are made, a new route segment is started and the process repeats.

As requests are assigned to route segments, their respective request-candidate nodes are removed from the graph. This means that the originally defined clusters may no longer be the best clustering of the remaining nodes. To alleviate this, every $\eta_3$ route segments, the clustering algorithm is repeated to assign nodes to updated clusters. Depending on the size of the instance considered, the re-clustering may be desired to run less frequently, and this is left to the discretion of the modeler. For the case study that follows, the re-clustering step was set to every 10 route segments.

The overall structure of the greedy request to route segment assignment is shown in Figure 5.3.

**Feasibility checking modules**

Two modules are needed to insert request candidates into route segments. The first finds the least cost way to insert the pickup and dropoff stops, and requires use of the second module to determine feasible times to service each stop.

Insertion test module

Given an existing route segment schedule, a request-candidate is tested for insertion into the schedule in the following way. First, the pickup stop is considered for insertion at every point in the schedule, and the position that results in the route with the shortest distance is taken. Then every position after this one is tested for the dropoff stop, and again the one resulting in the lowest route length is taken. Now with the tentative schedule, the service time module is run to assign arrival times to each stop in the schedule that obey the time windows. The service time module is described in the next section. If a feasible set of service times is found, then the capacity constraint is checked: that at each step in the route the number of passengers on board does not exceed the capacity of the vehicle. Next, two local improvement procedures are tried: the first one attempts to combine users to the same stop whenever this is feasible for them, and the second tests switching each stop one at a time, in the order of the schedule, to a very nearby stop (e.g. across the street) to see if the route length is improved, and time window feasibility is not affected.

Service times module

Given a stop ordering, the service time module begins by setting the service time of the first stop to its current $t_x$ value. It then calculates the travel time between all

subsequent stops in the route, and updates their $t_x$ values accordingly. If the vehicle arrives at a stop before the start of a time window, it waits until the time window starts before travelling to the next stop. If any service time violate the end of a time window, the initial $t_x$ value is decremented by 1 minute, and the stop times are recalculated. The starting time can be pushed back until the beginning of its time window. If a time window is still violated, then the schedule is flagged as infeasible.

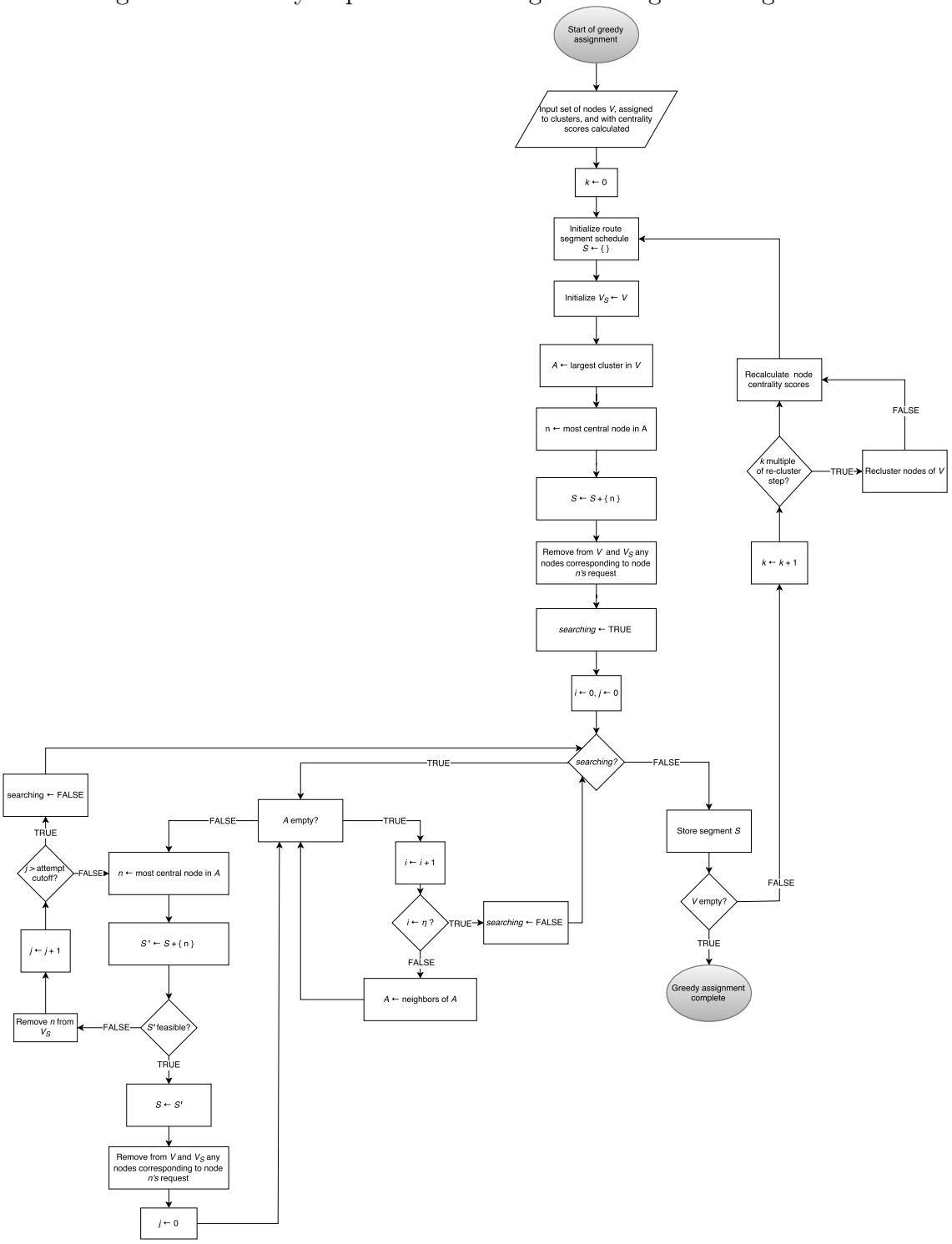
## 5.4 Part 4: Improvement heuristic

After the route segments have been created, there may be several requests that were unable to be inserted into a segment with any other requests, in other words the request is the only one in the segment. This can happen due to the greedy nature of the route insertion algorithm. If a request does not have an overwhelmingly high centrality score, then it may get skipped while its neighbors are being added.

This module tries to insert any of these "single" requests into another route segment, in order to reduce the total number of segments. It starts by enumerating the single requests, and sorting them based on the number of candidates they have. Requests with candidates are attempted in order of increasing number of candidates, as it is more likely to find feasible insertion for requests with lots of candidates. Requests with no candidates are tried at the end, since there is no information available about which segments might be compatible with it.

The module begins by selecting the single request with the fewest pickup-dropoff pair candidates. It then checks if it has any similar request-candidates, by

Figure 5.3: Greedy request to route segment assignment algorithm.



referring to the similarity matrix $W^R$. If it does, it attempts to insert the request into the smallest existing segment to which it is similar to, and if insertion fails, the second largest is tried, and so on. If there are no request-candidates similar to the single request, then all sections are tried, in order of increasing size. This is repeated until all single requests have received an attempt at reinsertion.

Additional local search improvement heuristics could be performed at this point, but this research only considers the step described.

## 5.5 Part 5: Route segment chaining

The output of the previous step is a set of route segments, which may or may not be served by distinct vehicles. For example, there may be two sections for which the last dropoff time of one is sufficiently earlier than the first pickup time of the other. These then could be served by the same vehicle. In order to reduce the number of required vehicles, this step "chains" together route segments to be served by a single vehicle. Another greedy heuristic is used, though it would be possible to formulate the chaining problem as an integer program. Doing so would also allow for crew-scheduling and maximum shift constraints to be imposed.

The first step of the chaining process is to find all pairs of compatible route sections: these are sections for which it's possible to leave the final dropoff stop of one and arrive either on time or early to the start of the other. At this point, the amount of compatibility could be quantified as a continuous value, for example as some combination of the distance required to travel and the amount of time the

vehicle would be early to the next pickup. However in this research, compatibility is treated as a binary condition. Next, an undirected graph is formed using the route sections as nodes and their compatibility as arcs. Note there is obvious asymmetry in the arcs: if segment $Y$ can be served before segment $Z$, then it's not possible for segment $Z$ to be served before $Y$. However, as will soon be apparent, this step does not need to consider the arc direction, so the arcs are treated as undirected.

The next step is to identify the cliques in this graph. A clique represents a set of segments that are mutually compatible (i.e., every member of the clique is compatible with every other member). Thus, a single vehicle can be assigned to all the members of a clique. Computing all cliques, however, is quite memory intensive for a solution with many route segments that covers a long period of time. For instance, if the sample time period is one hour, there may be many possible pairs of segments that can be served one before the other.

To reduce the cost of running this step, the following modification was made. For each pair of compatible segments $Y$ and $Z$, define the *idle time* between them as the amount of time after travelling from $Y$ and arriving at $Z$ before $Z$'s service start time. Remove any edges with idle time above a certain threshold. Then compute the cliques on this much smaller graph. Starting with the largest clique, assign its member route segments to the same vehicle, and proceed until all clique members have been assigned. Since a segment may belong to multiple cliques, once it has been assigned, the clique membership table is updated. Any segments that are not members of a clique – meaning they are not compatible with any other segments – must be served by separate vehicles.

After the first iteration, many route segments will have been chained together. Repeat the chaining process until there are no more cliques. At this point, the final vehicle assignment has been determined.

## 5.6 Phase II conclusion

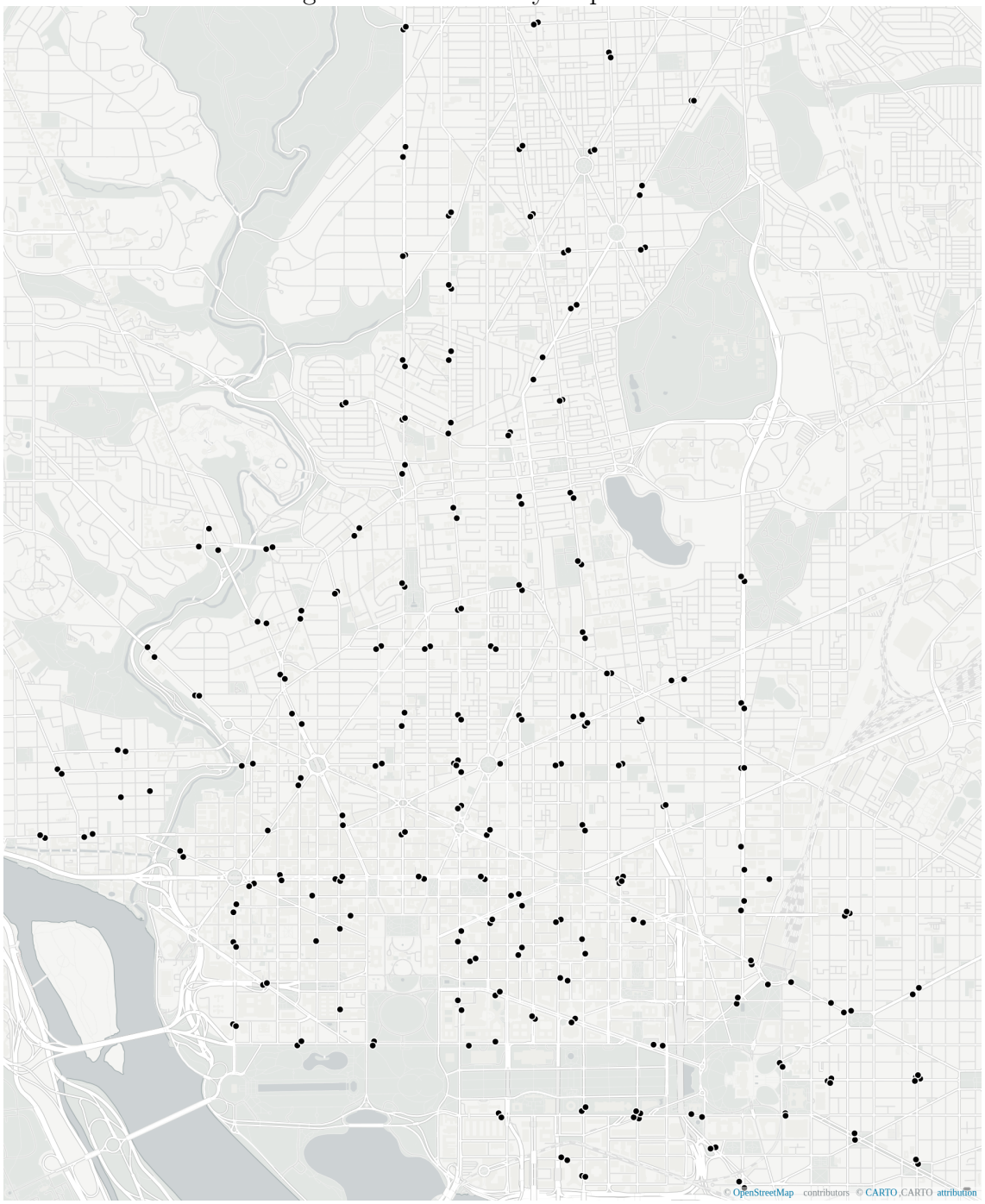
This completes the request-vehicle assignment algorithm. In summary, the algorithm references the similarity matrix from Phase I to create a measure of similarity between request-candidates, which are objects corresponding to the possible pickup-dropoff pairs that are feasible for each request. The similarity is set to 0 if the time ranges of the requests do not overlap. The similarity defines a directed graph, which can be made undirected by taking the larger of the forward and backward arc costs. Then a fast greedy clustering algorithm is run on the graph to determine request-candidates that are similar. Within each cluster, the weighted degree is calculated for each node, equal to the sum of the costs of the incident edges. Higher scores correspond to more "important" nodes, i.e., ones that are likely to be compatible with the most number of other nodes. A greedy insertion technique is used to generate segments of routes. Local perturbations are checked at each insertion, including grouping requests to the same stops and testing nearby stops. After the route sections are obtained, any single requests are attempted to be inserted into a more full section. Finally, the route sections are chained together, yielding a route corresponding to each vehicle.

# Chapter 6: Case Study: Washington DC Taxi Trips

To test the model's performance with a real scenario, a data set of all taxi trips taken in the month of May 2015 was obtained from the Washington DC Department of For-Hire Vehicles [63]. A particularly busy hour in the sample – 6 pm on Wednesday May $13^{\text{th}}$ – was taken as the test period. A network of stop locations was defined by using the existing locations of bus stops in the city, and filtering them to a more manageable number. The stop network and the area they cover is shown in Figure 6.1. The following sections detail the steps taken to implement the model, and the results of several instances with varying parameters.

All of Phase I was implemented in a PostgreSQL database. For Phase II, the selection of candidates for each request was performed in the database, and the remaining steps were performed using the R programming language [64]. The R `igraph` library was used for the graph clustering and centrality calculations [65]. All code was executed on a computer running Ubuntu with an Intel Core i7 processor and 16 GB of RAM.

Figure 6.1: Case study stop network




The stop network used in the case study contains 270 stops and covers an area of about 13.5 square miles.

## 6.1 Data sources

### 6.1.1 Road network

For this research, a digital representation of the street network was obtained from OpenStreetMap [66], an open source map database. The extract file was loaded into a PostgreSQL database [67] and parsed from its XML format into SQL tables. The road network is represented by two types of objects: nodes and ways, where nodes are 1-dimensional points, and ways are collections of points. Roads are represented as ways, and contain a node every time the road bends. This means that the segment between two successive nodes is always a straight line. The extract area contained 15,193 road type ways, which had an associated 121,526 nodes.

This research made use of the PostGIS extension for PostgreSQL [68], which provides an efficient way to perform geometric operations such as calculating the length of a line, even if it is specified in non-Euclidean coordinates such as latitude and longitude. The length of each straight-line segment was calculated and then the total length of each road segment was taken as the sum of its straight-line sub-segments. The bearing was also calculated for each sub-segment, which is the degrees differing from north, increasing in the clockwise direction. Any unnecessary nodes were removed in the following way: a node was kept if it represented an intersection of road segments, if it represented a bus stop, or if it was the start or end point of a road segment, and otherwise it was discarded. Any edges that contain sub-segments defined by the nodes that were removed were combined to form one

long edge, with length equal to the sum of the sub-segment lengths.

Next, the road data needed to be further processed to be used as a directed graph. Streets that were marked as one-way were not modified. For the remaining streets, a reverse edge was created, but different Id numbers were used for the new edges and respective nodes. This was done to prohibit taking U-turns at every node. Then artificial edges of length zero were added to connect the new edges to the original ones, but only if the angle of the implied turn was not larger than 135 degrees.

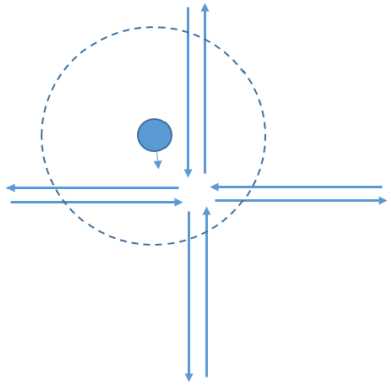
Lastly, a spatial index was created on the road segment table. This instructs the database to internally organize the location of each segment based on its physical location. The spatial index is used to reduce the amount of time needed to find objects close to one another.

### 6.1.2 Bus stop network

This research used the location of existing bus stops to form the stop network. The locations were obtained from the website of the Washington Metropolitan Area Transit Authority (WMATA) [69]. The data set contains the location coordinates in longitude and latitude of the stops, the stop name, and the bearing angle of the stop. As with the road segments, a spatial index was created for the stop locations.

The bus stops were inserted into the road network in the following way. For each bus stop, any of the straight-line road sub-segments within a small radius of the stop were considered. Finding nearby edge sub-segments is the step that benefited

Figure 6.2: Matching bus stops to the road network



from having a spatial index on both tables. Then from these candidates, the road sub-segment with the bearing most similar to the bearing of the stop was taken as the matched road segment. Then two artificial edges were created in the graph, connecting the start point of the matched edge to the stop, and then connecting the stop to the end point of the matched edge. This matching process is depicted in Figure 6.2.

A subset of the full set of stops was defined manually, with the goal of obtaining an even distribution of stops over the service area. Additionally, if a stop was selected on a two-way street, it was ensured that the stop on the opposite side of the street was also chosen. The network tested contained a total of 270 stops, and covered an area of 13.5 square miles. The map of the bus stop network is shown in Figure 6.1.

### 6.1.3 Demand data

The request data was taken to be real requests for taxi trips during the period of 6:00 PM to 7:00 PM on May 13, 2015. The data set contains the longitude and latitude coordinates of the origin and destination for each request. Any trips

that did not originate or end within $\gamma$ of a stop in the system were discarded. The mileage of the trip is contained in the data set as well, but in order to compare to the results of the model, the requests origin and destination locations were integrated into the road network in the same manner that the bus stops were matched, but taking the "bearing" as the angle defined by the straight line between the origin and destination. The taxi data set also contains the license plate Id of the taxi that served each request. Therefore it is possible to count how many taxi vehicles were used to meet the demand.

### 6.1.4 Shortest path engine

In order to perform the large number of shortest path calculations needed for Phase I of the algorithm, the `pgRouting` [70] extension for PostgreSQL was used. The convenience of this extension is that it simply requires the graph edges to be listed in a table having the columns "source node id," "target node id," and "cost." Then a number of functions are available to calculate either a one-to-one shortest path or one-to-many shortest paths. The function `pgr_kdijkstrapath` calculates one-to-many shortest paths and returns the edges in each of the paths. For each stop in the network, this function is called to get the shortest path edges to all other stops in the network.

## 6.2 Model Performance

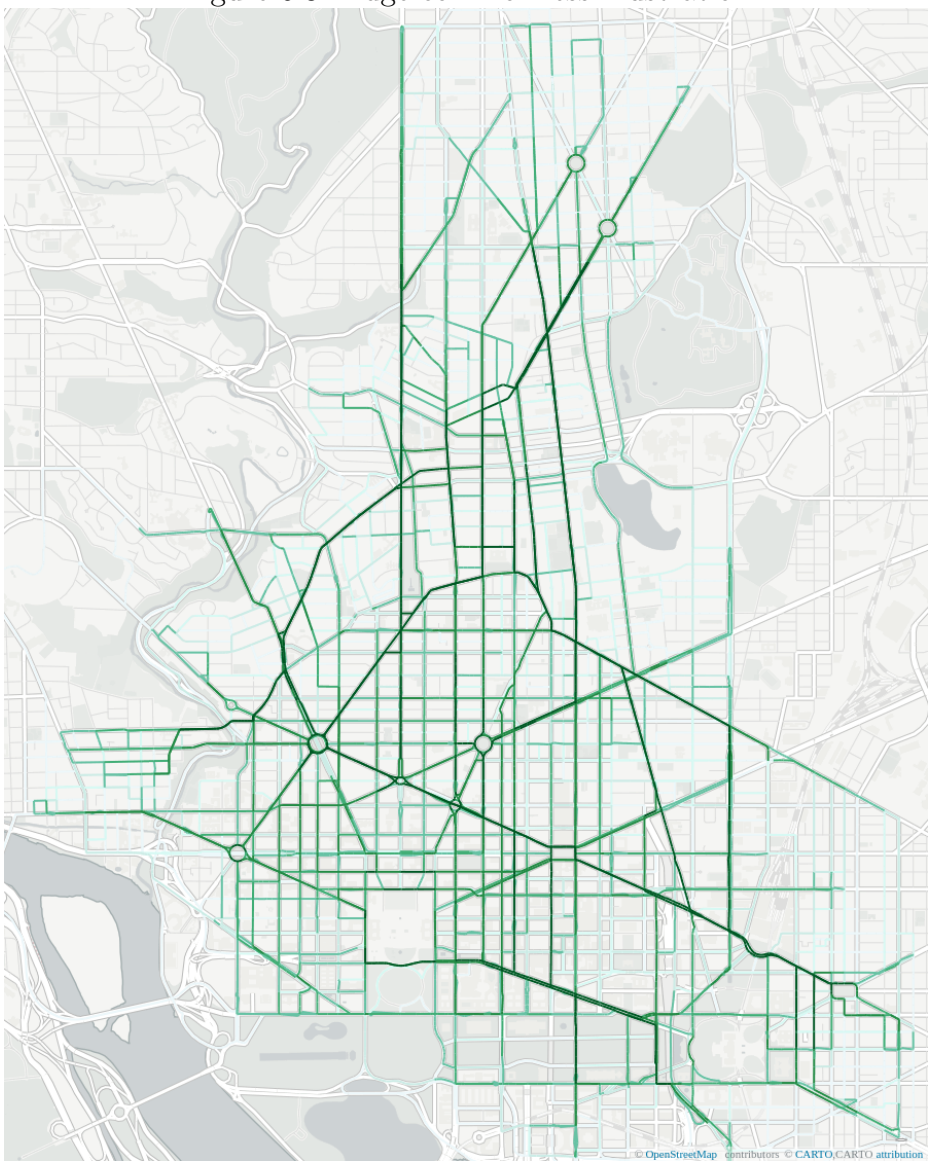
The model's performance is documented in the following sections. Table 6.2 summarizes the run time of both phases and their sub-parts.

### 6.2.1 Phase I

The stop network contains 270 stops. The set $M$ contains all feasible pickup-dropoff pairs, which are pairs of stops that are at least $\beta$ distance apart. The minimum route distance $\beta$ was set to 800 meters. Whereas the set of all pickup-dropoff pairs would have $270 \times 270 = 72,900$, the $\beta$ filter reduced the number to 67,118. It took approximately 30 minutes to generate all of the routes and compute their shortest paths. Then Phase I began by calculating the commonness of each edge used in the shortest paths. Figure 6.3 shows how the commonness is greater on the major roads. Three iterations were used for Phase I ($\kappa = 3$), and together had a run time of about 7 hours.

The most computationally intensive step of Phase I is the calculation of the similarity matrix. The results of the shortest path calculations are stored in a table containing the Id of the pickup-dropoff pair, and the corresponding Id's of each of the edges in the shortest paths. Determining similar pickup-dropoff pairs involves joining this table to itself. For the three iterations, the number of rows in this table is 6,597,541; 6,765,471; and 6,996,295, respectively. Two steps were taken to improve performance of these joins. The first was to draw a uniform grid over the service area map, and categorize each edge by its membership to a particular cell of

Figure 6.3: Edge commonness illustration




The commonness of each edge (proportional to the number of times an edge is used in a shortest path) is represented by the color of the edge. Darker colors indicate greater commonness. The commonness shown here is with respect to the third iteration of the Phase I iterative shortest path calculation.

the grid. If an edge overlaps multiple cells, the one in which the largest portion of its length falls into is taken. For this instance, a grid of squares with side length 600 meters was drawn over the service area, resulting in 357 cells. Then the shortest path table is indexed by the grid cell Id and the edge Id. Since there are much fewer grid cells than edges, the join is able to quickly eliminate a lot of possibilities, and thus run time is greatly improved. The second performance improvement step was to calculate the similarity matrix in segments: the 67,118 pickup-dropoff pairs were subdivided into 34 segments, and the similarity for each segment was calculated one at a time. The parameter $h$ determines the minimum amount of similarity two pickup-dropoff pairs must have in order to be recorded in the similarity matrix. For this case study, $h$ was set to 0.6.

### 6.2.2 Phase II

A total of 36 instances were created to test the effect of the following factors on system efficiency:

- The number of requests
- The inconvenience level $\lambda$
- The average driving speed in the network
- The effect of considering multiple candidate stops for each request

Two parameters are not tested by this case study. The first is the value of the maximum allowable walking distance, $\gamma$. This is not tested due to the fact

that the stop network is not sufficiently dense. Many requests only had one or two candidate stops, and the next closest stops were quite far. So an increase to $\gamma$ would not impact the number of candidates much, unless it was a large increase. Alternatively, setting $\gamma$ too low would make some requests ineligible for any stop, meaning that the number of requests under the different $\gamma$ settings would not be equal, and a fair comparison could not be made. Therefore it would not be possible to see a smooth response from increasing or decreasing $\gamma$. In order to assess the impact of this parameter, it would be necessary to have a much more dense stop network. Instead, the value of $\gamma$ was fixed to 400 meters for all of the instances.

The second factor that was not varied for testing is the vehicle capacity. This is so that the model can inform the vehicle sizes needed to serve the demand. For the instances studied, the capacity of all vehicles was set to the arbitrarily large value of 50.

### 6.2.3 Instance descriptions

For the one hour period of taxi data available, subsets of requests where generated by randomly removing a percentage of the requests. The full data set contained 1,291 requests with origin and destination within walking distance of stops in the network. Smaller samples were created by randomly removing 25, 50, and 75% of records from the full data set. The purpose of comparing the outcome under different demand levels is to show that the system is more efficient when demand is higher, as the chance to combine trips increases. Table 6.1 shows the number of

| Percentage of original data | Number of requests | Requests per vehicle |
|---|---|---|
| Full | 1,291 | 1.07 |
| 75% | 970 | 1.07 |
| 50% | 646 | 1.06 |
| 25% | 335 | 1.07 |

Table 6.1: Sizes of each request data subset.

requests in each of the request data subsets. It also provides the ratio of the number of requests to the number of taxi vehicles used, where a value of 1 indicates each request was served by a unique vehicle. This is the key metric used to compare the performance of the model on the various instances. While it is much greater for the proposed system, it is not a completely fair comparison since the model here is static – the requests are known in advance – whereas a taxi system is dynamic. Nonetheless, these values are shown for reference.

Next, for each of the request sets, three values of $\lambda$ were tested: 5, 10, and 15 minutes. The parameter $\lambda$ puts a limit on the amount of additional travel time allowed with respect to the direct travel time. Referring to Figure 5.2, decreasing lambda shifts the earliest pickup time closer to the latest pickup time. Therefore a larger $\lambda$ value means wider time windows for the requests, and therefore a larger feasible region.

There is a technicality that should be explained: when increasing the value of $\lambda$, for example from 5 to 10 minutes, it may become possible to serve more requests than with the lower value. This can happen when all of the candidates of a request require a total walking time greater than 5 minutes. With $\gamma = 5$, they would not be able to use any pickup-dropoff pair, however with $\gamma = 10$, they will have some

candidates. In order to be able to make a comparison across the different values of $\lambda$, only the requests that are feasible with $\lambda = 5$ are considered for all of the instances.

In addition to the value of $\lambda$ and the volume of the requests, the effectiveness of a shared ride system clearly also relies on the average travel speed on the road network. The faster a vehicle is able to move between stops, the more it is possible to combine trips without violating the passenger time windows. To measure this effect, for each of the four requests sets and three $\lambda$ values, the model was solved for two values of the driving speed: 11 mph and 20 mph.

An additional set of instances was run to test the importance of considering multiple possible pickup and dropoff locations. In the standard Dial-a-Ride problem, passengers are picked up at their requested origin and taken to their requested destination. In the proposed system, passengers may be served from one of many possible nearby stops. As a way to show the effectiveness of these flexible meeting points, the model was run assuming that the passengers were served from the pickup and dropoff stops closest to them. In other words, these instances considered the objective of minimizing the amount of walking required of passengers. These instances were performed assuming an average driving speed of 11 mph.

This results in a total of 36 instances. Their objective values are presented and discussed in the next section, and Tables 6.2 and 6.3 in the subsequent section reports their run times.

### 6.2.4 Measures of effectiveness

Two obvious measures of effectiveness for a shared ride service are the vehicle utilization rate (number of requests served per vehicle) and the amount of inconvenience experienced by the passengers. However, it is also important to evaluate the scalability of the request assignment algorithm. The model presented here exhibits a nice scaling property, in that:

1. Vehicle utilization increases with respect to increasing the demand (*Economies of scale*)

2. As demand grows, it becomes possible to have a low inconvenience level and maintain high utilization

The first result can be shown by comparing the vehicle utilization as a function of the request volume, controlling for the inconvenience level $\lambda$. This relationship is shown in Figure 6.4. Each line represents a constant value of $\lambda$: either 5, 10, or 15 minutes. It can be seen that the vehicle utilization (requests per vehicle) *increases* as request volume increases. For example, with $\lambda = 10$ and an average travel speed of 11 mph, it requires 68 vehicles to serve 335 requests, giving a utilization rate of 4.93. However, with the same parameter settings it requires only 199 vehicles to serve 1,291 requests, giving a utilization rate of 7.51. Increasing the number of requests by about four times resulted in only about a three time increase in the number of vehicles needed. This means that each additional unit demand because less costly to serve.

The increase in utilization as a function of requests can be explained by two factors. First, the vehicle capacity is allowed to adjust to accommodate a greater number of requests. This is shown in Figure 6.5, which shows the average capacity of the vehicles in the solution to each instance. The capacity for each vehicle is found by taking the greatest number of passengers simultaneously on board at any point in the vehicle's schedule.

The second result requires plotting the vehicle utilization, which is the model's objective, against the inconvenience level, controlling for the number of requests. As discussed, increasing $\lambda$ leads to a larger feasible region, since it becomes possible to combine trips that for lower values of $\lambda$ were not feasible to combine. As a result, vehicle utilization increases with $\lambda$. Depending on the needs of the system operator, it may be desirable to offer a more convenient service if cost is less important. Then, for example, if request volume is high enough, it is possible to still obtain utilization of about 5, which may be considered good enough, even with $\lambda$ equal to 5 minutes.

The two panels of Figures 6.4 and 6.6 show the effect of increasing the average travel speed from 11 mph to 20 mph. Note that the instances with 335 requests obtained a utilization between 3.94 and 5.78 when travel speed is 11 mph, but when speed is increased, the utilization increases to between 5.20 and 9.32. This means that in a low demand scenario, being able to increase travel speed can boost the utilization to a level experienced under high demand. Therefore, if a city has low request volume, they can still run an efficient shared ride service if they increase travel speeds, for example by letting the vehicles use HOV or dedicated lanes.

The two panels of Figure 6.7 show the impact of choosing to minimize the

Figure 6.4: Improvements to vehicle utilization with respect to demand

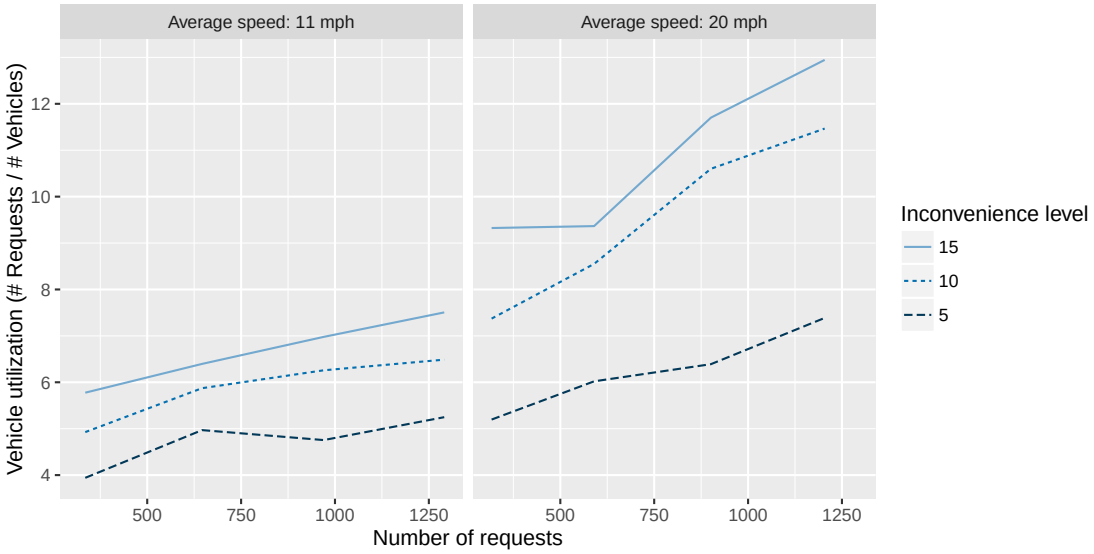



Figure 6.5: Vehicle capacity with respect to $\lambda$ and demand

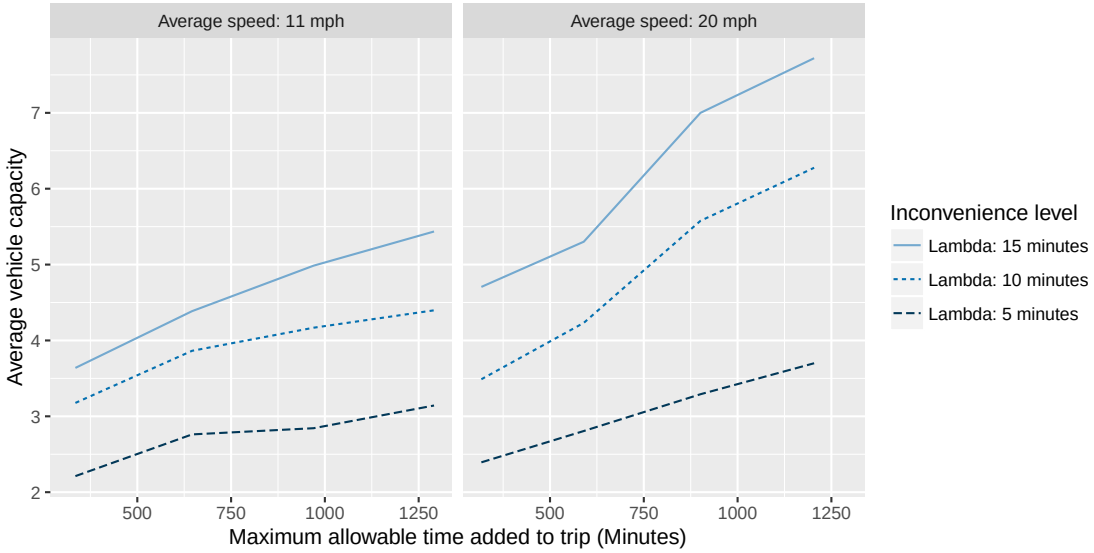


Figure 6.6: Increase to vehicle utilization with respect to inconvenience

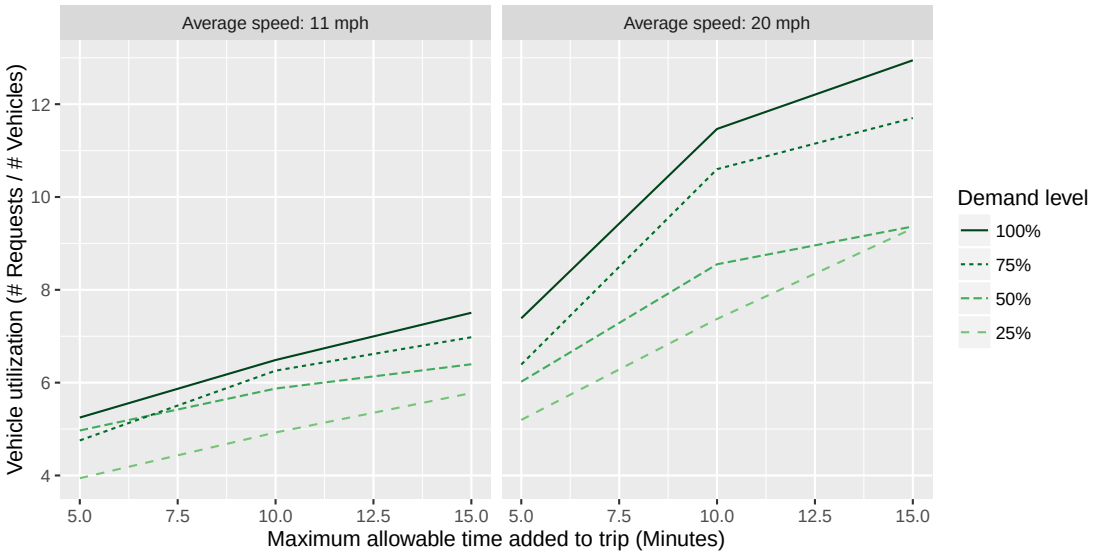
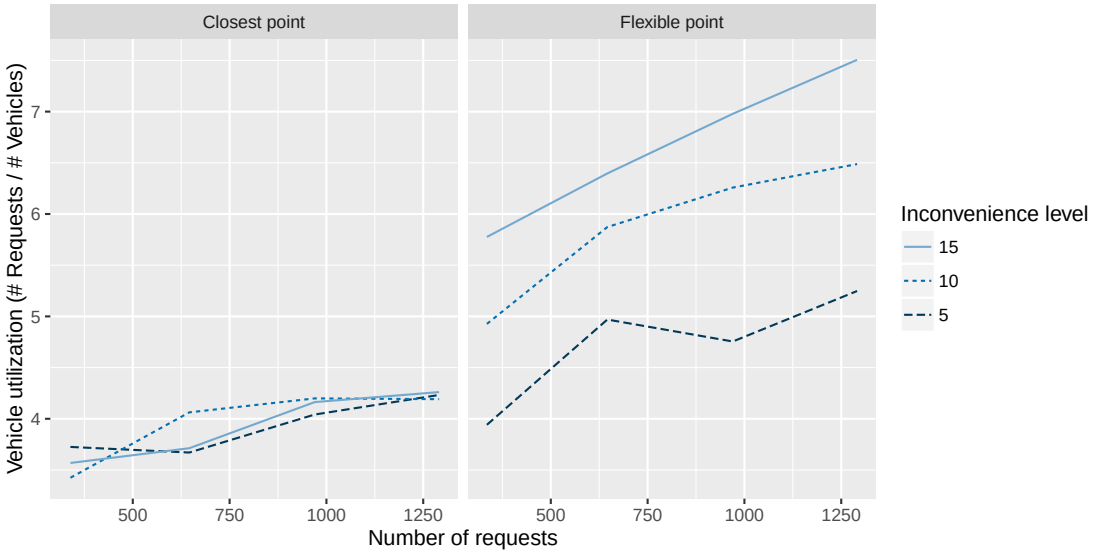



Figure 6.7: Impact of flexible meeting points vs closest points
Average speed: 11 mph

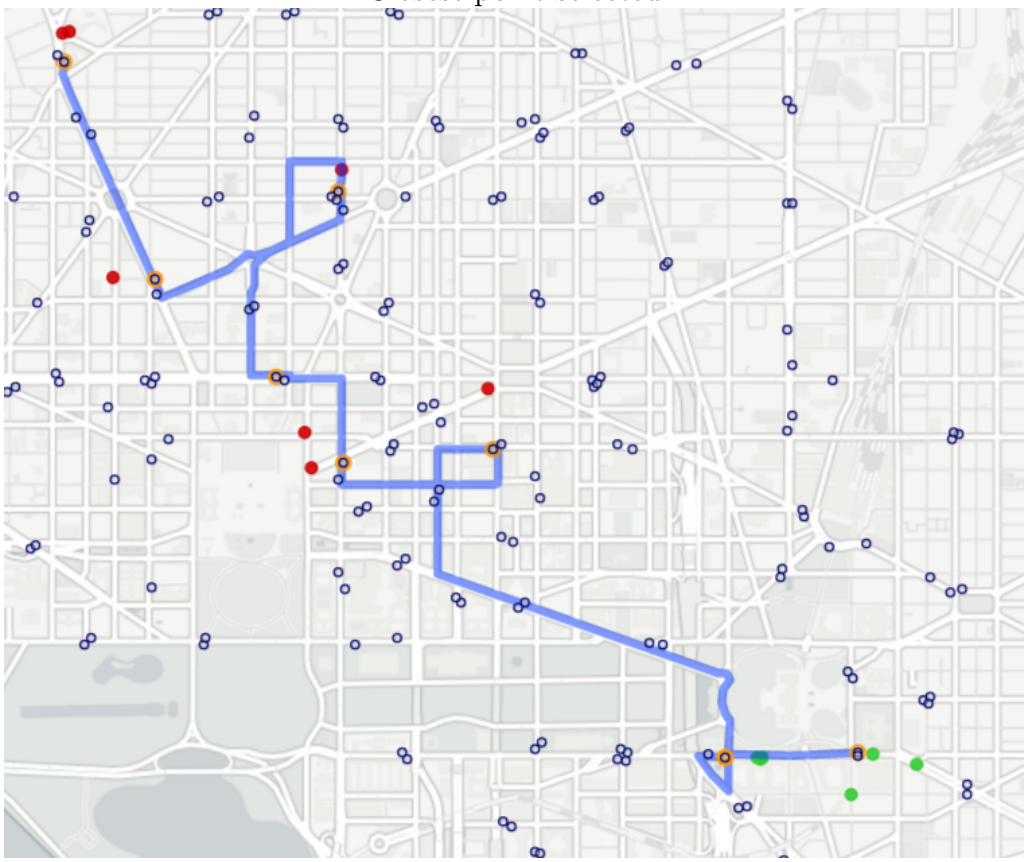


passengers' walking distance instead of the number of vehicles used. In the left panel, only the closet pickup and dropoff stop were considered for each request. Thus vehicles are likely required to driver farther to pick up and drop off each individual passenger. This means that time window constraints will be violated much sooner (i.e., with a lower number of passengers), therefore requiring a larger fleet to serve the same level of demand. While the flexible scenarios achieved a vehicle utilization of up to 7.51, the instances that only considered the closest stops attained a maximum utilization of 4.26. Two sample route segments from the solution with and without flexible meeting points are shown in Figure 6.8. Green and red dots show the origin an destination locations, respectively, of requests. The upper panel shows the case when the closest points are chosen. The route must make many time consuming maneuvers to reach the rider. On the other hand, in the route with flexible meeting points, the vehicle can take a much more streamlined path, resulting in an improved travel time for all passengers.

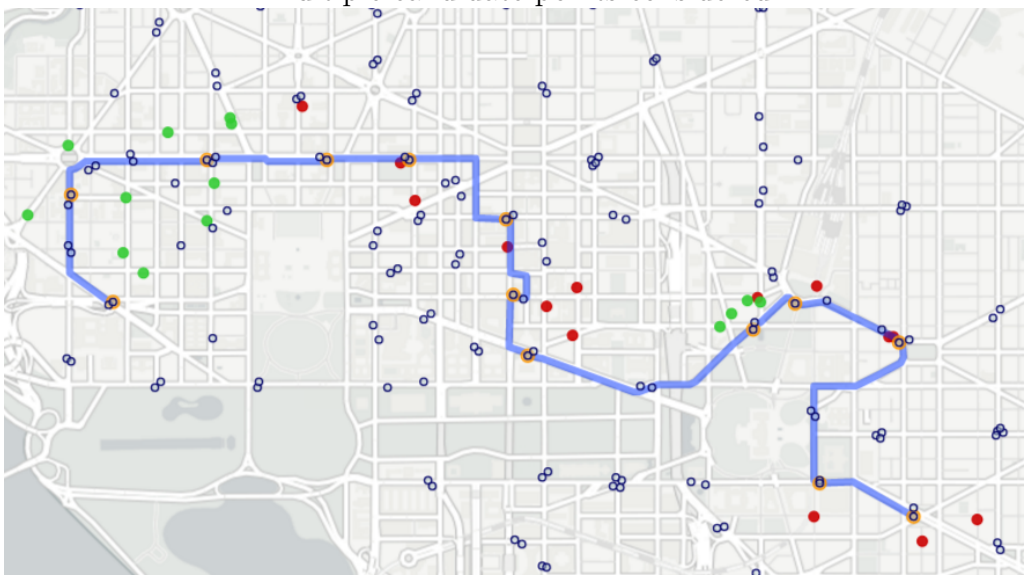
It should be noted that the impact of choosing the closest stop varies greatly depending on the density of the stop network. If the stop network is fairly sparse, then the chance of multiple riders being served by the same stop increases, and system efficiency may not be impacted much. However for a dense stop network, choosing the closest stop leads the system to approximate a taxi service, and the likelihood of trip combining falls.

Figure 6.8: Comparison of routes with and without flexible meeting points

Closest point selected



Multiple candidate points considered



### 6.2.5 Performance summary

Table 6.2 summarizes the parameter settings for each instance, as well as the average number of available candidates for each request. This is a function of the inconvenience level $\lambda$: for larger values of $\lambda$ there are more feasible pickup-dropoff pairs available for each request. Thus the problem size grows as $\lambda$ increases. The center columns in the table present number of vehicles used in the solution, $K$, as well as the vehicle utilization (number of requests per vehicle used), and the average capacity of the vehicles in the solution. Table 6.3 shows the same information but for the twelve instances for which only the closest pickup and dropoff stops were considered (thus the number of candidates per request is 1).

The rightmost group of columns show the solution time for each of the instances, in units of seconds. The five parts of the Phase II solution algorithm are 1) identifying request candidates, 2) defining the graph and performing clustering and centrality calculations, 3) greedy request to route segment assignment, 4) improvement heuristic, and 5) route segment chaining. The run times of parts 3 and 4 dominate the others. Two observations can be made. The run time of part 3 increases steadily with the number of requests, as is shown in Figure 6.9. However, the run time of part 4 is more sensitive to the $\lambda$ parameter. The task of part 4 is to attempt to insert requests that failed to be combined with any others in part 3. For lower $\lambda$ or driving speed there is less similarity among requests which leads the clustering step to yield many small clusters, making single requests more common. This leads part 4 to have a higher run time for lower $\lambda$, as shown in Figure 6.10.

Table 6.2: Run times and objective values of all instances, flexible meeting points

| Id | Demand | $\|R\|$ | $\lambda$ | Speed | Cand./Req. | $K$ | $\|R\|/K$ | Avg. Cap. | Part 1 | Part 2 | Part 3 | Part 4 | Part 5 | Total time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 25 | 335 | 5 | 11 | 6 | 85 | 3.94 | 2.21 | 2 | 2 | 113 | 1337 | 29 | 1483 |
| 2 | 25 | 335 | 10 | 11 | 17 | 68 | 4.93 | 3.18 | 2 | 6 | 274 | 1273 | 32 | 1587 |
| 3 | 25 | 335 | 15 | 11 | 17 | 58 | 5.78 | 3.64 | 2 | 5 | 384 | 791 | 23 | 1205 |
| 4 | 50 | 646 | 5 | 11 | 6 | 130 | 4.97 | 2.76 | 3 | 4 | 304 | 3000 | 92 | 3403 |
| 5 | 50 | 646 | 10 | 11 | 17 | 110 | 5.87 | 3.86 | 4 | 12 | 888 | 2755 | 100 | 3759 |
| 6 | 50 | 646 | 15 | 11 | 18 | 101 | 6.40 | 4.39 | 5 | 11 | 1134 | 1797 | 82 | 3029 |
| 7 | 75 | 970 | 5 | 11 | 6 | 204 | 4.75 | 2.84 | 4 | 5 | 784 | 2711 | 55 | 3559 |
| 8 | 75 | 970 | 10 | 11 | 16 | 155 | 6.26 | 4.17 | 6 | 17 | 2189 | 2793 | 60 | 5065 |
| 9 | 75 | 970 | 15 | 11 | 17 | 139 | 6.98 | 4.99 | 8 | 16 | 2287 | 2305 | 40 | 4656 |
| 10 | 100 | 1,291 | 5 | 11 | 6 | 246 | 5.25 | 3.14 | 22 | 7 | 1132 | 3707 | 102 | 4970 |
| 11 | 100 | 1,291 | 10 | 11 | 16 | 199 | 6.49 | 4.40 | 15 | 26 | 3925 | 3223 | 98 | 7287 |
| 12 | 100 | 1,291 | 15 | 11 | 17 | 172 | 7.51 | 5.44 | 8 | 26 | 3758 | 3153 | 85 | 7030 |
| 13 | 25 | 317 | 5 | 20 | 6 | 61 | 5.20 | 2.39 | 2 | 2 | 88 | 822 | 73 | 987 |
| 14 | 25 | 317 | 10 | 20 | 17 | 43 | 6.89 | 3.49 | 2 | 6 | 293 | 982 | 67 | 1350 |
| 15 | 25 | 317 | 15 | 20 | 18 | 34 | 9.32 | 4.71 | 2 | 5 | 277 | 387 | 68 | 739 |
| 16 | 50 | 590 | 5 | 20 | 6 | 98 | 5.78 | 2.81 | 3 | 4 | 200 | 1706 | 91 | 2004 |
| 17 | 50 | 590 | 10 | 20 | 18 | 69 | 8.19 | 4.23 | 4 | 12 | 781 | 1335 | 85 | 2217 |
| 18 | 50 | 590 | 15 | 20 | 18 | 63 | 9.36 | 5.30 | 4 | 11 | 872 | 1108 | 91 | 2086 |
| 19 | 75 | 901 | 5 | 20 | 5 | 141 | 6.63 | 3.29 | 4 | 5 | 483 | 1512 | 40 | 2044 |
| 20 | 75 | 901 | 10 | 20 | 17 | 85 | 10.60 | 5.58 | 5 | 18 | 1584 | 1881 | 48 | 3536 |
| 21 | 75 | 901 | 15 | 20 | 18 | 77 | 12.01 | 7.00 | 7 | 16 | 2121 | 1149 | 53 | 3346 |
| 22 | 100 | 1,204 | 5 | 20 | 6 | 163 | 7.39 | 3.70 | 6 | 7 | 865 | 2184 | 107 | 3169 |
| 23 | 100 | 1,204 | 10 | 20 | 17 | 105 | 11.36 | 6.28 | 9 | 27 | 2850 | 2421 | 117 | 5424 |
| 24 | 100 | 1,204 | 15 | 20 | 18 | 93 | 12.29 | 7.72 | 12 | 25 | 2693 | 1948 | 116 | 4794 |

Table 6.3: Run times and objective values of all instances, closest meeting points

| Id | Demand | $\|R\|$ | $\lambda$ | Speed | Cand./Req. | $K$ | $\|R\|/K$ | Avg. Cap. | Part 1 | Part 2 | Part 3 | Part 4 | Part 5 | Total time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 25 | 25 | 339 | 5 | 11 | 1 | 91 | 3.73 | 1.96 | 3 | 1 | 53 | 760 | 26 | 843 |
| 26 | 25 | 339 | 10 | 11 | 1 | 99 | 3.42 | 2.11 | 3 | 1 | 60 | 583 | 23 | 670 |
| 27 | 25 | 339 | 15 | 11 | 1 | 95 | 3.57 | 2.26 | 3 | 1 | 53 | 392 | 23 | 472 |
| 28 | 50 | 646 | 5 | 11 | 1 | 176 | 3.67 | 1.98 | 4 | 1 | 118 | 1812 | 34 | 1969 |
| 29 | 50 | 646 | 10 | 11 | 1 | 159 | 4.06 | 2.31 | 4 | 1 | 125 | 1168 | 31 | 1329 |
| 30 | 50 | 646 | 15 | 11 | 1 | 174 | 3.71 | 2.32 | 4 | 1 | 116 | 932 | 29 | 1082 |
| 31 | 75 | 970 | 5 | 11 | 1 | 240 | 4.04 | 2.29 | 4 | 1 | 219 | 2307 | 29 | 2560 |
| 32 | 75 | 970 | 10 | 11 | 1 | 231 | 4.20 | 2.49 | 5 | 1 | 207 | 1871 | 26 | 2110 |
| 33 | 75 | 970 | 15 | 11 | 1 | 233 | 4.16 | 2.64 | 5 | 1 | 211 | 1435 | 22 | 1674 |
| 34 | 100 | 1291 | 5 | 11 | 1 | 305 | 4.23 | 2.36 | 6 | 1 | 354 | 3113 | 47 | 3521 |
| 35 | 100 | 1291 | 10 | 11 | 1 | 308 | 4.19 | 2.58 | 6 | 1 | 272 | 2247 | 42 | 2568 |
| 36 | 100 | 1291 | 15 | 11 | 1 | 303 | 4.26 | 2.70 | 6 | 1 | 315 | 1654 | 36 | 2012 |

Figure 6.9: Comparing run times of Phase II, Part 3

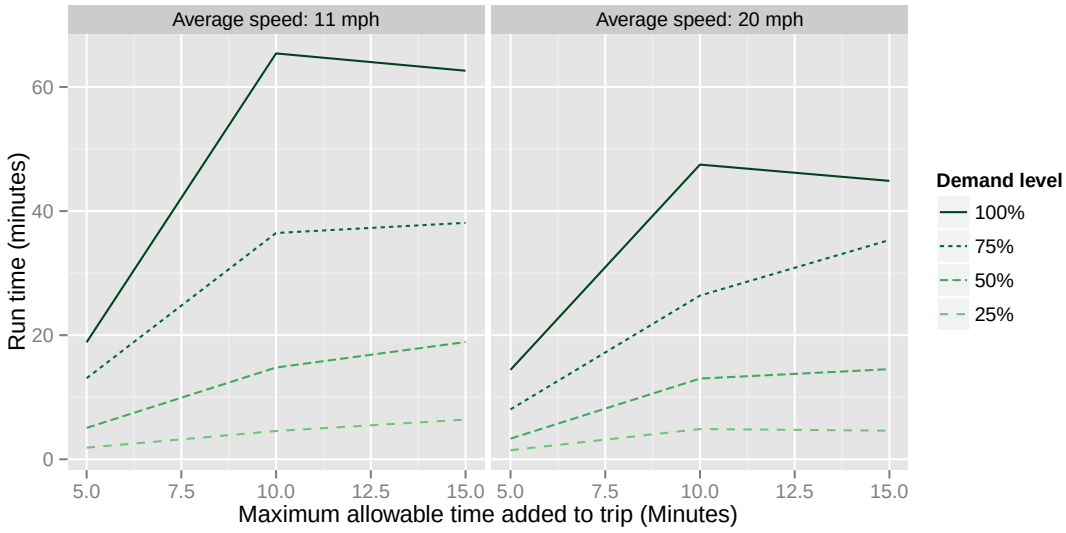
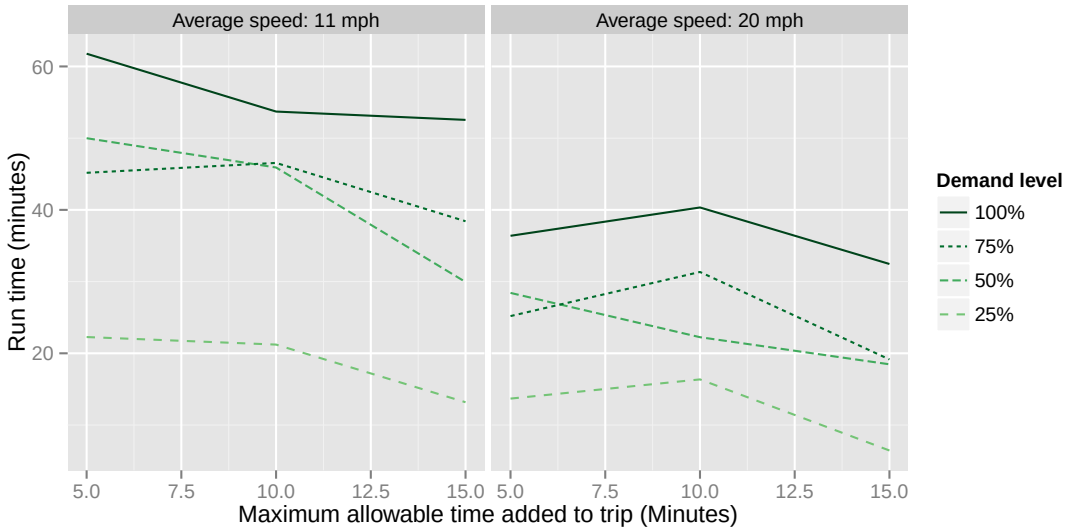



Figure 6.10: Comparing run times of Phase II, Part 4



# Chapter 7: Conclusion and Future Work

## 7.1 Future extensions

### 7.1.1 Further parameter testing

The case study presented here tested different values of the inconvenience level $\lambda$, the request volume, and the average travel speed in the network. Two parameters were left out of this analysis: vehicle capacity and the walking limit. Future work could study the impact of using a fleet of small, medium, or large vehicles on the vehicle utilization objective.

In order to meaningfully test the walking limit, however, it would be necessary to use a much more dense stop network. Without a sufficiently dense stop network it is not possible to measure smooth responses to changes in the walking limit. Increasing the number of stops would increase the run time of Phase I, since every new stop introduces $N$ new pickup-dropoff pairs. Therefore new techniques, from both a computing and algorithmic standpoint, would need to be developed to handle large networks.

A more subtle parameter that was not tested is the value of $h$, which determines how similar two pickup-dropoff routes must be before adding their cor-

responding edge to the graph. In this case study $h$ was set to 0.6, meaning that any request-candidates with less than this similarity did not have an edge between them. The higher the value of $h$, the smaller the graph, which may be desirable from a computational standpoint. However, the resulting clusters may be smaller and more numerous, which will make it difficult for the greedy-assignment phase to form route segments. Experimenting with a lower $h$ value may yield even higher vehicle utilization, at the expense of increased computational requirement.

An additional factor that is critical to the success of a shared ride system is to what degree there are strong spatial and temporal patterns in the users' requests. A controlled experiment could be done, in which requests are uniformly generated over space and time. A completely random demand pattern represents the "worst case" scenario for this system, since trips are much less likely to be similar.

### 7.1.2 Modeling extensions

The model considered here had the objective of maximizing vehicle utilization, subject to serving all of the demand. It may be desirable however, to include a financial constraint. This would allow for the possibility of rejecting certain requests that are too costly to serve. This research did not impose a fleet size constraint, but such a constraint would make the model more realistic. This would be possible within the current modeling framework by simply removing the vehicles with lowest utilization until the fleet size constraint is satisfied.

An additional advancement would be to allow for a mixed fleet, for example

with some small vehicles and some high capacity vehicles. This may be most beneficial in an area that has a dense urban core and a sprawling surrounding residential area. The variable vehicle capacity would have to be taken into account in the greedy insertion step. Instead of initializing an arbitrary vehicle for a new request, care would need to be taken to assign larger vehicles to requests that are expected to overlap with many others, and smaller vehicles to requests to which few others are similar. Lastly, crew scheduling considerations should be considered, in order to allow drivers to take breaks or end their shift.

## 7.2 Conclusion

This research has approached a new problem, which seeks to pick up and drop off passengers in an efficient manner. It bears similarity to the Dial-a-Ride problem and the School Bus Routing Problem. With respect to the former, a solution is sought that assigns a timetable for vehicles to pick up and drop off passengers, subject to time window and vehicle capacity constraints. However, passengers are not necessarily picked up or dropped off at their precise origin and destination locations. Instead, they are asked to walk to a stop in order to meet the vehicle. This is done to reduce the amount of in-vehicle time, which in turn allows a greater number of trips to be combined, than compared with a traditional carpool or shared taxi system.

The concurrent optimization of the vehicle route and the passenger stop assignment makes this problem similar to the School Bus Routing Problem. The key

difference is that in the SBRP, passengers are taken to a common destination at a common time.

The model proposed here uses a graph clustering algorithm to find groups of request candidates that are similar. The edge weights for the graph are derived in the first phase of the algorithm, which may be done ahead of time since it is time consuming. The second phase assigns similar request candidates to vehicles in a greedy fashion, and then applies some improvement heuristics.

The system in general is able to attain high vehicle utilization thanks to meeting points, which allow multiple passengers to be served from the same location at no additional cost to the route. Moreover, the results show that such a system has the property of *economies of scale*: as the demand for the service increases, the vehicle utilization rate increases. Put another way, when demand increases, the number of vehicles required increases, but by (considerably) less than the demand increase, meaning it becomes less costly to serve each unit of demand. This is due to two factors. First, the vehicle capacity is allowed to adjust to accommodate more passengers. Second, as the request volume increases, the likelihood of encountering similar requests increases. This makes it easier to combine more trips without incurring much additional inconvenience to the other riders. As a corollary, when the number of requests becomes sufficiently high it becomes possible to achieve reasonable vehicle utilization with a low inconvenience to passengers. Additional efficiency gains can be realized with an increased travel speed in the system, which can be achieved with dedicated lanes for the shared vehicle fleet. This should encourage city transportation officials to consider offering such a service.

# Bibliography


[1] Robert Cervero. Informal transport in the developing world. *United Nations Centre for Human Settlements*, 2000.

[2] Holly Krambek and Li Qu. Toward an open transit service data standard in developing Asian countries. *Transportation Research Record*, 6:30–36, 2015.

[3] Sarah Williams, Adam White, Peter Waiganjo, Daniel Orwa, and Jacqueline Klopp. The Digital Matatu project: Using cell phones to create an open source data for Nairobi's semi-formal bus system. *Journal of Transport Geography*, 49:39 – 51, 2015.

[4] Simon Saddier, Zachary Patterson, Alex Johnson, and Megan Chan. Mapping the jitney network with smartphones in Accra, Ghana: The AccraMobile experiment. *Transportation Research Board*, 2016.

[5] Chariot. `http://www.chariot.com`. Accessed: 2016-07-21.

[6] Bridj. `http://www.bridj.com`. Accessed: 2016-07-21.

[7] Tao Liu and Avishai Ceder. Analysis of a new public-transport-service concept: Customized bus in China. *Transport Policy*, 39:63–76, 2015.

[8] N. Chan and S. Shaheen. Ridesharing in north america: Past, present, and future. *Transportation Research Board*, 2010.

[9] Bla bla car. `https://www.blablacar.com/`. Accessed: 2016-05-12.

[10] Carma. `https://www.gocarma.com/`. Accessed: 2016-05-12.

[11] ubercommute. `http://www.uberpartnerschicago.com/ubercommute/`. Accessed: 2016-05-12.

[12] uberPool. `https://www.uber.com/ride/uberpool/`. Accessed: 2016-05-12.

[13] Lyft Line. `https://www.lyft.com/line`. Accessed: 2016-05-12.

[14] Americans With Disabilities Act of 1990, Subpart F., Section 37.121 (1990).

[15] D. Kauffman. Operational experiences with flexible transit services: A synthesis of transit practices. Technical report, Transportation Research Board, 2004.

[16] Jang-Jei Jaw, Amedeo R. Odoni, Harilaos N. Psaraftis, and Nigel H. M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B*, 20B:243–257, 1986.

[17] Sophie N. Parragh, Karl F. Doerner, and Richard F. Hartl. A survey on pickup and delivery problems, Part II: Transportation between pickup and delivery. *Journal of Business Management*, 58:81–117, June 2008.

[18] Gerardo Berbeglia, Jean-Franois Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8 – 15, 2010.

[19] Harilaos N. Psaraftis. An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, 17:351–257, August 1983.

[20] Jacques Desrosiers, Yvan Dumas, and François Soumis. A dynamic progrmming method for the large scale single vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6:301–325, February 1986.

[21] Jean-François Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54:573–586, May 2006.

[22] P. Augerat, J. M. Belenguer, E. Benavent, and D. Naddef A. Corberán. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, 106:546–557, 1998.

[23] Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49:258–272, July 2007.

[24] Mengyang Liu and Andrew Lim Zhixing Luo. A branch-and-cut algorithm for a realistic dial-a-ride problem. *Transportation Research Part B*, 81:267–288, November 2015.

[25] Xiugang Li and Luca Quadrifoglio. Feeder transit services: Choosing between fixed and demand responsive policy. *Transportation Research Part C*, 18:770–780, October 2010.

[26] Marco Diana and Maged M. Dessouky. A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. *Transportation Research Part B*, 38:539–557, 2004.

[27] Ying Luo and Paul Schonfeld. A rejected-reinsertion heuristic for the static dial-a-ride problem. *Transportation Research Part B*, 41:736–755, July 2007.

[28] Zhihai Xiang, Chengbin Chu, and Haoxun Chen. A fast heuristic for solving a large-scale static dial-a-ride problem under complex constraints. *European Journal of Operations Research*, 174:1117–1139, January 2006.

[29] Lauri Háme. An adaptive insertion algorithm for the single-vehicle dial-a-ride problem with narrow time windows. *European Journal of Operational Research*, 209:11–22, 2011.

[30] Luca Coslovich, Raffaele Pesenti, and Walter Ukovich. A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operations Research*, 175:1605–1615, November 2006.

[31] Jean-François Cordeau and Gilbert Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B*, 37:579–594, 2003.

[32] Gulay Barbarosoglu and Demet Ozgur. A tabu search algorithm for the vehicle routing problem. *Computers and Operations Research*, 26:255–270, 1999.

[33] Michel Gendreau, Gilbert Laporte, Christophe Musaraganyi, and Eric Taillard. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers and Operations Research*, 26:1153–1173, 1999.

[34] Andrea Attanasio, Jean-François Cordeau, Gianpaolo Ghiani, and Gilbert Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30:377–387, March 2004.

[35] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem. *INFORMS Journal on Computing*, 24:343–355, 2012.

[36] G. Berbeglia, G. Pesant, and L.-M. Rousseau. Checking the feasibility of dial-a-ride instances using constraint programming. *Transportation Science*, 2010.

[37] Dominik Kirchler and Roberto Wolfler Calvo. A granular tabu search dial-a-ride problem. *Transportation Research Part B: Methodological*, 56:12–135, October 2013.

[38] Ker-Tsung Lee, Pei-Ju Wo, and Shih-Han Wang. The planning and design of taxipooling on feeder system. *Proceedings of the 2004 IEEE International Conference on Networking, Sensing and Control*, March 2004.

[39] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 410–421, April 2013.

[40] Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223:295303, 2012.

[41] Andrew Amey. Real-time ridesharing: Exploring the opportunities and challenges of designing a technology-based rideshare trial for the MIT community. Master's thesis, Massachusetts Institute of Technology, May 2010.

[42] Monirehalsadat Mahmoudi and Xuesong Zhou. Finding optimal solutions for vehicle routing problem with pickup and delivery services with time windows: A dynamic programming approach based on statespacetime network representations. *Transportation Research Part B: Methodological*, 89:19–42, July 2016.

[43] Hadi Hosni, Joe Naoum-Sawaya, and Hassan Artail. The shared-taxi problem: Formulation and solution methods. *Transportation Research Part B*, 70:303–318, 2014.

[44] Niels Agatz, Alan L. Erera, Martin W.P. Savelsbergh, and Xing Wang. Dynamic ride-sharing: A simulation study in metro Atlanta. *Transportation Research Part B*, 45:1450–1464, 2011.

[45] Mitja Stiglic, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar. The benefits of meeting points in ride-sharing systems. *Transportation Research Part B*, 82:36–53, 2015.

[46] Mitja Stiglic, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar. Making dynamic ride-sharing work: The impact of driver and rider flexibility. *Transportation Research Part E*, 91:190–207, 2016.

[47] Philip Gruebele. Interactive system for real time dynamic multi-hop carpooling. 2008.

[48] Wesam Herbawi and Michael Weber. *Evolutionary Multiobjective Route Planning in Dynamic Multi-hop Ridesharing*, pages 84–95. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[49] Junhyuk Park and Byung-In Kim. The school bus routing problem: A review. *European Journal of Operational Research*, 202:311–319, 2010.

[50] Jalel Euchi and Rafaa Mraihi. The urban bus routing problem in the Tunisian case by the hybrid artificial ant colony algorithm. *Swarm and Evolutionary Computation*, 2:15 – 24, 2012.

[51] Junhyuk Park, Hyunchul Tae, and Byung-In Kim. A post-improvement procedure for the mixed load school bus routing problem. *European Journal of Operational Research*, 217(1):204 – 213, 2012.

[52] Luc Chapleau, Jacques-A. Ferland, and Jean-Marc Rousseau. Clustering for routing in densely populated areas. *European Journal of Operational Research*, 20(1):48 – 57, 1985.

[53] Lawrence Bodin and Lon Berman. Routing and scheduling of school buses by computer. *Transportation Science*, 13(2):113–129, 1979.

[54] Jacques Desrosiers, Jacques Ferland, Jean-Marc Rousseau, G. Lapalme, and Luc Chapleau. An overview of a school busing system. In *, Internal Conference on Transportation. Scientific Management of Transportation, vol. IX*, pages 235–243, 1980.

[55] Robert Bowerman, Brent Hall, and Paul Calamai. A multi-objective optimization approach to urban school bus routing: Formulation and solution method. *Transportation Research Part A: Policy and Practice*, 29(2):107 – 123, 1995.

[56] Jore Riera-Ledesma and Juan-José Salazar-González. Solving school bus routing using the multiple vehicle traveling purchaser problem: A branch-and-cut approach. *Computers and Operations Research*, 39:391–404, 2012.

[57] Patrick Schittekat, Joris Kinable, Kenneth Srensen, Marc Sevaux, Frits Spieksma, and Johan Springael. A metaheuristic for the school bus routing problem with bus stop selection. *European Journal of Operational Research*, 229(2):518 – 528, 2013.

[58] Levent Kaan and Eli V. Olinick. The vanpool assignment problem: Optimization models and solution algorithms. *Computers and Industrial Engineering*, 66(1):24 – 40, 2013.

[59] Douglas Santos and Eduardo Xavier. Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Applications*, 42:6728–6737, 2015.

[60] Fragkiskos D. Malliaros and Michalis Vazirgiannis. Clustering and community detection in directed networks: A survey. *Physics Reports*, 533(4):95 – 142, 2013. Clustering and Community Detection in Directed Networks: A Survey.

[61] Ruchen Duan Yingbin Liang. Bounds and capacity theorems for cognitive interference channels with state. *CoRR*, abs/1207.0016, 2012.

[62] Peter V. Marsden. Network centrality, measures of. In James D. Wright, editor, *International Encyclopedia of the Social and Behavioral Sciences (Second Edition)*, pages 532 – 539. Elsevier, Oxford, second edition edition, 2015.

[63] Department of for-hire vehicles. `http://dfhv.dc.gov/`. Accessed: 2015-04-28.

[64] The R Project for Statistical Computing. `https://www.r-project.org/`. Accessed: 2015-04-25.

[65] igraph R package. http://igraph.org/r/. Accessed: 2015-01-05.

[66] Openstreetmap. https://www.openstreetmap.org/about. Accessed: 2015-11-03.

[67] Postgresql – The world's most advanced open source database. https://www.postgresql.org/. Accessed: 2015-10-25.

[68] PostGIS – Spatial and geographic objects for PostgreSQL. http://www.postgis.net/. Accessed: 2016-01-19.

[69] DC open data. http://www.opendata.dc.gov/. Accessed: 2016-05-05.

[70] pgRouting project. http://www.pgrouting.org/. Accessed: 2016-04-07.