Incorporation of reservations in an on-demand ridesharing system

N.K. Theodoridis

TU Delft

Incorporation of reservations in an on-demand ridesharing system

by

N.K. Theodoridis

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday November 28, 2022 at 15:00.

Student number:	4486897	
Date:	November 14, 2022	
Report number:	2022.MME.8728	
Thesis committee:	dr. A.S. Fielbaum Schnitzler,	TU Delft, supervisor
	dr. B. Atasoy,	TU Delft, supervisor
	dr. J. Alonso Mora	TU Delft, supervisor
	dr. F. Schulte	TU Delft, external committee member

Cover: Times square 20 September 2013 by Turtix (Modified)
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

The last year has been an interesting and educational experience. This thesis is the final project in my study program at the Delft University of Technology and marks the end of my student life in Delft. I am immensely grateful for all the opportunities, activities and interactions I have had over the past seven years in Delft both within my studies and outside. My time in Delft has allowed me to develop myself, get a grasp of how the world works, discover my interests and helped me become the person I am today.

I would like to take this opportunity to express my sincere gratitude to the people that were essential in my journey. First I would like to thank my supervisors for guiding me in my thesis at an interesting topic. Doing my thesis, I learned about modelling, running simulations and upgrading my programming skills. But also, and more importantly, about keeping the bigger picture in mind and taking time for reflection. I want to especially thank Bilge Atasoy and Andrès Fielbaum for pulling me out of all the interesting looking rabbit holes I came across along the way in this thesis.

Furthermore, I would like to thank my family and friends for their support and motivation during my studies. First of all, I would like to thank my parents and brother for their love and support my whole life. You raised me and have given me the confidence to choose my own path. I would also like to thank all my friends that have joined me for lunches, coffee breaks, walks on campus and study sessions during my thesis (and before!). Furthermore, I would like to thank my house mates where it was always nice to come home to and chat the evening away solving world problems; my study friends for the shared study sessions, help and ideas; my close group of friends from the rowing association Proteus whom I have seen at least weekly for the past 7 years plus.

Finally, I would like to thank my girlfriend for her kindness, love and positive energy I always cherish. I promise I will stop my daily briefing on ideas for ridesharing somewhere in the near future.

Thank you all!

Nander Theodoridis
Delft, November 2022

Abstract

With increasing urbanization and a need to reduce greenhouse gas emissions, new forms of urban mobility are studied worldwide. An interesting transportation method is ridesharing, where people can share parts of their rides with other passengers travelling in a similar direction in the same car. To improve the attraction of ridesharing systems, research into extensions to the existing ridesharing model formulation is required. The goal of this research is to make ridesharing more attractive to potential users by including the option to make reservations.

In this work, the inclusion of reservations (pre-bookings) into the on-demand ridesharing problem is studied. The inclusion of ridesharing has to potential to make better assignments by having knowledge on part of the future demand. The main challenge from reservations is the increased computational load generated by having more available information. This thesis proposes multiple adjustments and different heuristics for a state-of-the-art on-demand ridesharing system to deal with the requirements and challenges brought by the incorporation of reservations. An additional consideration is a trade-off between providing benefits for reservations and serving on-demand requests.

Numeric experiments are performed on the Manhattan street network with NYC taxicab request data. The results indicate that reservations can be included in on-demand ridesharing at a tolerable computational cost with the proposed framework. Passengers that make reservations shortly in advance can be guaranteed service once initially accepted by the system. Having passengers make reservations in advance does increase the overall service the system can provide.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
RV graph	Request Vehicle graph
RTV graph	Request Trip Vehicle graph
ILP	Integer linear program

Symbols

Symbol	Definition	Unit
G = (N, E)	Graph of the network of nodes and directed edges describing the road network	
$\mathbb{V}$	set of all vehicles	
v	single vehicle	
v_n	location of vehicle at node n	
v_t	time till the next node is reached	
w_k	waiting time for requests k	[s]
d_k	detour time for request k	[s]
δ_k	total delay time for request k, this is the sum of the waiting time w_k and the detour d_k time for request k	
Δ	maximum waiting time for each requests	
Ω	maximum detour time for each requests	
Π_k	rejection penalty for request k	[s]
$\mathcal{P}$	single passengers	
$\mathcal{P}_v$	passengers on board off vehicle v	
$\mathbb{R}$	set of requests	
r	single requests	
r_r	reveal time for requests	
r_e	emerge time for requests	
r_{min}	minimum travel time between origin and destination for request r	
r_o	origin node of requests	
r_g	goal node of request	
r_p	pickup time of request	
r_d	drop off time for request	
$r_{reservation}$	binary parameter if request r is a reservation	
$r_{assigned\ before}$	parameter if $r \in \mathbb{R}_{ok}$ at the last iteration	
$\mathcal{T}$	trip consisting of requests	
$\mathbb{T}$	set of all trips	
$\mathbb{T}_{v=j}$	set of trips that can be served by vehicle j	
$\mathbb{T}_{\mathcal{R}=k}$	set of trips that contain request k	
$\mathbb{V}_{\mathcal{T}=i}$	set of vehicles that can serve trip i	
$\mathbb{R}_{ok}$	set of requests assigned to a vehicle	
$\mathbb{R}_{no}$	set of requests not assigned to a vehicle	
$e(r, \mathcal{T})$	edge between request r and trip $\mathcal{T}$	
$e(\mathcal{T}, v)$	edge between trip $\mathcal{T}$ and vehicle v	

Symbol	Definition	Unit
$c_{i,j}$	cost to assign trip i to vehicle j	
$\mathcal{X}_k$	decision variable if request is assigned to trip or not (1 means not assigned)	
$\epsilon_{i,j}$	decision variable to assign trip i to vehicle j	

1 Introduction

With increasing urbanization and a need to reduce greenhouse gas emissions, new forms of urban mobility are studied worldwide. An interesting transportation method is ridesharing, where people can share parts of rides with other passengers travelling in a similar direction in the same car. Another common name for this type of ridesharing is ridepooling. The specific type of ridesharing employed in this thesis is one in which a fleet of vehicles is centrally controlled to provide transportation through ridesharing. Matches for fellow passengers are made based on overlapping trajectories. The exact route taken and the other passengers on the vehicle can differ each time the same route is requested.

The implementation of large-scale ridesharing has shown potential. With current technology, large numbers of passengers can be efficiently allocated for ridesharing Santi et al. (2014). In Alonso-Mora et al. (2017), 98% of taxi demand in Manhattan can be served with just 30% of the taxis, while staying within reasonable constraints on delays. Ridesharing shows a potential upside for users, operators and society as a whole in case of widespread adoption [Agatz et al. (2012); OECD (2015)]. These advantages do require a sufficiently high density of travel demand to make appropriate matches. Therefore, the application shows more promise for urban areas compared to rural areas. The advantages of ridesharing could potentially be amplified by the adoption of autonomous vehicles (AVs). Conditions for AVs to succeed and technological readiness are discussed in Luo et al. (2021).

From a societal perspective, ridesharing has clear benefits. The vehicle miles driven can be reduced compared to both taxi services and private transport, thus reducing energy consumption, noise pollution, etc. Mobility can be provided on-demand, with fewer vehicles needed to service demand, therefore creating more space in cities. For example, more greenspaces can be created instead of existing parking areas. Next to this, ridesharing can offer attractive door-to-door service. However, ridesharing is still not employed on a large scale in reality. For potential users (passengers), the (perceived) disadvantages apparently outweigh the offered advantages. Otherwise, ridesharing would be expected to already be more commonly used. Sharing has some potential downsides, leading to an unwillingness to share. For example reduced privacy, less convenience and perceived longer travel times.

To improve the attraction of ridesharing systems, research into extensions to the existing ridesharing model formulation is required. The goal would be to increase the advantages and/or reduce the perceived disadvantages. If successful, the real-world adoption and the corresponding benefits might become more widespread. One potentially relevant extension to the rideshare formulation is the inclusion of reservations. A reservation (pre-booking) is a request that is known to the system a certain time before its first pick-up time. Reservations would provide the system with additional knowledge about how parts of the demand in the future will look and thus allow for better assignment in the current iteration. If specific user benefits would be attached to placing reservations (e.g. guaranteed service or faster pick-up), the inclusion can be considered user-friendly.

From a model's perspective, incorporating reservations into the ridesharing problem formulation brings challenges. Firstly, a higher computational load can be expected since more knowledge of future

demand is available to the system. This is caused by having more potential combinations between passengers and vehicles. Secondly, it is reasonable to assume passengers will only make reservations if placing a reservation is associated with benefits. These benefits will impose constraints on the solution space and thus potentially reduce the quality of the passenger assignments.

In the research approach, an existing state-of-the-art model will be extended to study the effects from the inclusion of reservations in an on-demand ridesharing system. The extensions are focused on restricting the increase in computational load and reviewing the impact of reservations over different operational scenarios. An important objective of this research is to allow for reservations in a large-scale ridesharing setting.

For this research, reservations that are placed shortly in advance are considered. This is applicable in an on-demand ridesharing system where passengers can indicate if they want to be served as soon as possible or intend to leave in for example 10 minutes.

Numerical simulations will be performed with the Manhattan road network and the NYC yellow cab data set.

1.1. Research objective

The objective of this work is to assess the effectiveness and feasibility of the inclusion of reservations in a large-scale on-demand ridesharing system. This is done by developing a method to include reservations into an existing system and study their impact on the solution quality and the consequent trade-offs. The different effects and trade-offs that emerge because of reservations will be quantified to assess the effectiveness of including reservations in the model.
The research objectives for this thesis are listed below.

- Include reservations efficiently into a state-of-the-art on-demand ridesharing method, allowing for implementation on a large scale.
- Quantify the effects of having reservations available on multiple Key Performance Indicators (KPIs) and review the imposed computational burden.
- Study the trade-offs in providing benefits to reservations over normal requests.

1.2. Contribution statement

A state-of-the-art on-demand ridesharing system is modified to allow for reservations. The adjustments focus on decreasing the imposed computational load to allow for large-scale ridesharing systems with reservations and assign benefits to passengers making reservations. To reach this goal, reservations are only revealed to the system a short time in advance and heuristics are applied to reduce the computational load.

More specifically, the contributions are listed below:

- A method to reveal and assign reservations into an on-demand ridesharing system is developed and implemented, with the aim of being effective and lightweight.
- Detailed experiments are designed and subsequently performed to gain insights into the potential benefits from the inclusion of reservations, while at the same time assessing the computational costs of doing so.
- A numerical study assessing the potential trade-off between providing improved service for reservations over on-demand requests is performed, including an assessment of different types of benefits on the overall service level of the system.
- Multiple heuristics to limit computational burden are developed, applied, tested and evaluated for their effectiveness and efficiency.

1.3. Thesis overview

The work in the research is presented as follows. In chapter 2, relevant background literature on the working principles of ridesharing is discussed. Multiple research gaps with respect to ridesharing are

discussed. The inclusion of reservations is the topic selected for further research. The methodology of the research is described in chapter 3. The chapter consists of three main parts. The first part describes the problem formulation, the second part provides insights into a state-of-the-art method for on-demand ridesharing. The third part contains the methodical extensions used to include reservations. Chapter 4 contains the experimental setup for the research. The results are discussed in chapter 5. In this chapter, the effects of reservations, computational load, benefits for reservations and the effect of heuristics are discussed. Finally, chapter 6 consists of conclusions, discussions and recommendations for further work.

2

Literature review

In this section, core aspects of ridesharing will be discussed from both the passenger and operator perspectives.
First, different aspects of static and dynamic ridesharing are discussed. This section is followed by a review of matching algorithms. Matching is the process where passengers are assigned to vehicles.

The general concept of rideshare is to use fewer vehicles to service more demand. This can be achieved by assigning passengers to vehicles efficiently, thus reducing the vehicle miles to transport the passengers to their destinations.

2.1. Static and dynamic ridesharing

Ridesharing can be done statically and dynamically. In the static version, the assignment is made beforehand and is set in stone. This requires all information to be known beforehand so that passengers and vehicles can be matched. The most important information is the set of requests and the travel times in the network. In the dynamic version of ridesharing, the assignment is not determined beforehand but remains flexible over time, i.e. during the ride. This allows the assignment to respond to new situations, such as emerging demand or stochastic/varying travel times. The downside is that the assignment procedure should be run repeatedly to optimize for the new situation. In this section details regarding static or dynamic ridesharing are discussed.

The assignment of passengers to vehicles can be done statically where all information over the planning horizon is known beforehand. Having all information over the planning horizon allows for a solution to be non-myopic (optimized over the entire planning horizon). To achieve the most benefit from the available knowledge, sufficient time should be available to make the best possible overall planning. The required time to utilize the information is highly dependent on the size of the problem, this will be discussed in more detail in section 2.3.1.

A reason to use static assignments is to solve for often occurring instances. In this case, finding a better solution is worth the effort as the solution is used multiple times. An example could be classical carpooling, where the solution is used regularly with the same people. As most people prefer to use the same mode of transport each day, optimizing for this behaviour is relevant (Aarts, Verplanken, and Knippenberg (1998)).

Another option is to assign passengers dynamically to vehicles. An often studied problem is the on-demand assignment of passengers. In the on-demand scenario requests for trips can emerge at any time. Another aspect of ridesharing that could benefit from re-optimization is if stochastic travel times in the network are taken into account. This scenario is discussed in detail in section 2.5.3. In this current section, the focus will be on-demand assignment.

In case on-demand assignment is used, this is usually processed in batches of incoming requests instead of instantaneous. This allows for more combinations to be considered and thus better matches to be made compared to assigning every passenger one by one in order of arrival. Batch time can vary

from 10 seconds (Simonetto, Monteil, and Gambella (2019)) to multiple minutes. This means that all current demand is known once the assignment is made, but future demand is unknown. In this situation, methods have to make a trade-off between possible matches and usability from a customer point of view. A longer batch time allows for better matches to be made at the cost of increased complexity during the assignment for a batch. On the other hand, a shorter batch time means passengers have a shorter waiting period before they are assigned to a vehicle. Direct assignment to a vehicle for emerging demand is also possible, which is described by Fagnant and Kockelman (2018). A downside of direct assignment is that it is myopic and greedy, thus not reaching optimum system-wide performance.

The advantage of dynamic assignment comes from the usability side since it allows for on-demand ridesharing and/or the consideration of stochastic parameters, for example travel times. The advantage of on-demand ridesharing is that it offers an easy way for passengers to use the service. Since dynamic assignment does not have information about future demand, the current optimum does not have to be the global optimum if more information would have been available. As a consequence, the assignment is not necessarily the best in the long run. Steps taken to account for future demand are discussed in section 2.5.1.

The assignment of passengers to vehicles can be solved in two ways, either exact or with heuristics. The advantage of exact solutions is that the optimum can be found, however, this might take a lot of computational power. Therefore, exact solutions are mostly used for small problem sizes. Heuristics don't necessarily provide optimal solutions, but can reach good results in a limited time. This makes it an obvious candidate for dynamic problems where batches of requests have to be assigned quickly. A possible heuristic is a local search algorithm, as described in Beraldi et al. (2010).

An important method to dynamically assign passengers to vehicles is provided in Alonso-Mora et al. (2017). The method contains multiple steps that lead to an integer linear optimization problem (ILP). The method is anytime optimal, meaning it converges to the optimum over time. The first step in the method is focused on creating groups of riders that can share a ride within the constraints of the problem. Secondly, the ILP problem is solved to assign groups of passengers to the best vehicles. The ILP is initialized with a greedy solution and optimizes over time. Lastly, rebalancing empty vehicles is added to better accommodate for future demand. Passengers that are not yet picked up by a vehicle might still be re-assigned, as long as their waiting time does not increase as a consequence. New demand can thus be used to improve the solution and the routing for passengers.

A different approach is proposed by Simonetto, Monteil, and Gambella (2019), where a linear assignment formulation is used. In this concept, cars make a bid to pick up a single passenger. The bids are then combined and the best overall outcome is looked for. The method works by having cars make a bid to pick up a certain passenger. To calculate the costs, cars calculate the resulting detour time for the current passengers and the waiting time for the new passenger. The algorithm can reach good results by having short batch times. These batch times are very short, therefore grouping has no additional value. The method is thus computationally lighter compared to the method by Alonso-Mora et al. (2017) as no groups are made. In the next batch, the bids can lead to similar groups forming without going over as many possibilities. To reduce computational complexity, the only vehicles that can make a bid for a passenger are those within a certain range from the passenger. The order of pick-up and drop-off for the new passenger is only inserted in the order of the vehicle for vehicles with a capacity larger than 4. If the capacity is 4 or lower, all possible orders of pick-up and drop-off are checked. This leads to a limited number of possible combinations to check and thus a reduction in the required time.

Exact algorithms for dynamic ridesharing are also available. The previously mentioned difficulties at large-scale problems have to be addressed before an exact algorithm be executed fast enough. An option is to reduce the number of potential matches to be evaluated. This is described in Kucharski and Cats (2020). Here a pre-selection is made for good matches, only these good options are then solved exactly. The pre-selection is based on the concept of utility, which will be discussed in section 2.2.

To summarise, the trend in literature is towards the dynamic assignment of ridesharing. This trend is enabled by increases in computational power required for quicker optimization methods. Most re-

search focuses on assigning on-demand requests. Requests are often bundled within a batch time to improve the assignment.

2.2. Matching methods

Matchmaking in ridesharing is the action where passengers are assigned to vehicles. Matching can be done both for groups or for individual requests. There are two fundamental ways in which these matches can be made.

The first and most used option is based on time windows. In this method, there are time constraints on how much waiting time and detour time are allowed. An important reason to use time windows is to limit the number of possible combinations of riders. This is necessary since otherwise the number of possible groups grows exponentially with the number of passengers. The assumption is made that the constraints on waiting and detour time are acceptable for users. The objective function for matchmaking in this case usually consists of minimizing the waiting + detour time for the passengers. Time windows are used in Alonso-Mora et al. (2017) and Simonetto, Monteil, and Gambella (2019). In some studies, monetary costs are assigned to waiting and driving. In practice, the difference is small as usually there is a linear scaling in time and monetary value.

A different basis for matchmaking is the concept of utility. Utility is how attractive a match is from the passenger's perspective. In Kucharski and Cats (2020) utility is based on the required time, incurred cost, perceived delay and sharing discomfort. These factors combined lead to how attractive a certain match is. This can be combined between different potential matches and also with different modes of transport. The method assumes fixed values for the value of time, cost, etc for all passengers in the network. As indicated in the paper, fixed values are not ideal, but are easy to use. The objective function for this method consists of maximizing the utility of transportation. An important advantage of this methodology is that the attractiveness of ridesharing can be compared to other modes of transport. The method by Kucharski and Cats (2020) allows the calculation of the fare reduction required to make ridesharing attractive from a passenger perspective relative to the alternatives. By the nature of ridesharing, the trip can take longer than expected once the trip has been accepted. This happens when another passenger pick-up has been inserted between acceptance of the ride and drop off. Therefore, the value of delay perception should also be affected by delays once the trip is accepted. Effects and consequences of this unpredictability are described by Fielbaum and Alonso-Mora (2020). Here a trade-off between unreliability in prediction and system performance is identified. There is also a differentiation between one-time unreliability (within a single trip, due to new demand being inserted in a ride) and daily unreliability (differences between multiple trips).

The differences between utility and time windows are in effect limited as smaller waiting times also lead to better utility and higher waiting time to lower utility (at the same fare for the trip).

In short, matching is mostly done based on time windows. An important advantage of this method is that the number of options is more limited, as will be discussed further in the following section. Matching based on utility has the advantage that it allows to study competition with other forms of transport as people will take the option that is best for them.

2.3. Scaleability of ridesharing

In this section, the effects of problem size in a ridesharing assignment are discussed. Firstly, the combinatorial effects caused by the scale of the rideshare problem are evaluated. Secondly, the effects on performance due to scale are reviewed. With an increase in possible combinations it becomes more difficult to find good solutions as more options could be considered.

The scale of the problem is defined by multiple parameters. The most important ones are the road network, number of passengers, shareability between passengers, number of vehicles and their capacity.

2.3.1. Combinatorial effects of scale

The road network is represented as graph $G = (N, E)$, with N the pick-up/drop-off locations that are in the network. E represents the edges (roads) between these locations. Most nodes are connected

to multiple other nodes, so the number of edges is considerably larger than the number of nodes. The nodes and edges together can be used to create a graph of the network. This graph can, but does not have to be, bi-directional. A reason could be one-way streets, so that travel one way is different from the other way.

Closely related to the number of nodes and edges is the geographical size of the area for which the system is designed. The size of the area corresponds with the travel time from location A to location B and thus with the fleet size required for a certain level of service. The scale of the assignment problem is further defined by the number of passengers.

The number of passengers and the number of vehicles both influence the number of possible combinations. The number of passengers leads to a (up to exponential) growth in the possible combinations of passengers. These groups are then assigned to vehicles which is another combinatorial problem.

Vehicle capacity is an important parameter as it influences the maximum size of groups and thus the number of potential groups. Additionally, the vehicle capacity affects how many options there are for the pick-up/drop-off order for the passengers in the vehicle. As discussed earlier, time window constraints are an effective way of reducing the number of groups that can be created.

To improve service, Fu and Chow (2022) considers the option of transfers between different ridesharing vehicles. The inclusion of transfers in ridesharing would increase the number of possible combinations. Passengers that could not be in the same group due to time window constraints might now be matched because they could transfer to other vehicles. Next to this, there are multiple possible locations where transfers could occur thus increasing the number of routing options. This effect is amplified as multiple transfers within the system would be allowed. The added complexity due to transfers could be constrained with limits on the number of transfers, maximum times between transfers and other limits. Fu and Chow conclude that gains in travel time can be achieved by allowing transfers, however, the assignment procedure scales poorly for larger ridesharing instances due to the combinatorial problem.

In Fielbaum, Bai, and Alonso-Mora (2021), the option to include walking in the rideshare assignment is considered. Here the pick-up and drop-off locations are not only the exact origin and destination but also include a set of points within a certain range that could easily be walked. The advantage of allowing a small bit of walking is that more efficient routes can be used. For example around one-way streets or by not entering streets with lower maximum speeds. This means less detour time for all other passengers and less waiting time for future passengers. These advantages in travel time for multiple passengers could outweigh the (potential) additional time and discomfort for the passenger that has to walk. Walking here can also be seen as a transfer to a different mode of transport with specific costs and benefits (no transfer time in this case). The added complexity comes in the form of a lot of additional combinations between origin and destination pairs to be considered (a set of points around the original origin and destination). The increase in combinations is even larger for rideshare, as multiple sets of pick-up and drop-off points (for the different passengers) have to be compared.

A possible solution for very large problems is provided in Pelzer et al. (2015), where a method to split up the problem into smaller sections and later combine these is proposed.

2.3.2. Performance effects of scale

The scale of a rideshare problem affects the efficiency of the assignments in the model. As described in Fielbaum, Tirachini, and Mora (2021), the Mohring effect is present for ridesharing vehicles. The Mohring effect is an effect originating from public transport research. The main concept of the paper is that low demand leads to low supply. At low demand there is little possibility to share as there is little overlap in origin and destination pairs, therefore most rides will be private. At higher demand more detours will become necessary to deliver passengers to their destination, while the shared distance travelled is still limited. This deteriorates the service to the customers. At higher demand, the shareability increases and thus the effectiveness of ridesharing becomes evident. The Mohring effect is also studied in Kaddoura and Schlenther (2021). In this paper, the scalability of a simulation is researched. The paper describes how the effects of ridesharing can be studied on smaller problem instances and how these would scale to large problem instances. The goal is to gain insights from simpler and thus easier to solve problems.

To summarize, the scale of problems assessed in ridesharing has significant implications. On the one hand, there are effects on the number of combinations that can be assigned. A larger number of possible assignments could lead to better matches as more overlap in requests emerges. On the other hand, having additional options within a set computational budget could lead to worse performance as less convergence in the solution can be reached.

2.4. Ridesharing and public transport

The combination of ridesharing and public transport is interesting to review as the two systems have different strengths and could work symbiotically. Public transport (PT) offers the advantage of being efficient in miles travelled compared to passengers transported. Furthermore, PT requires limited public space relative to the number of passengers transported and provides a large capacity. The downside of PT is that it doesn't have a dense first/last mile system. This could potentially be provided by ridesharing systems.

Combining ridesharing with public transport could provide mutual benefits as the strengths of the one can cover parts of the weaknesses of the other (Ma et al. (2019)). People from areas around a transit station could walk or use ridesharing to arrive at the transit station. The high capacity transit could then be used for long-distance transport and walking/rideshare could be used as a last-mile transport mode. Origin and destination pairs far away from transit systems can be served with rideshare. A schematic version is given in figure 2.1. The letters R, T and W stand for ridesharing, (public) transport and walking respectively.

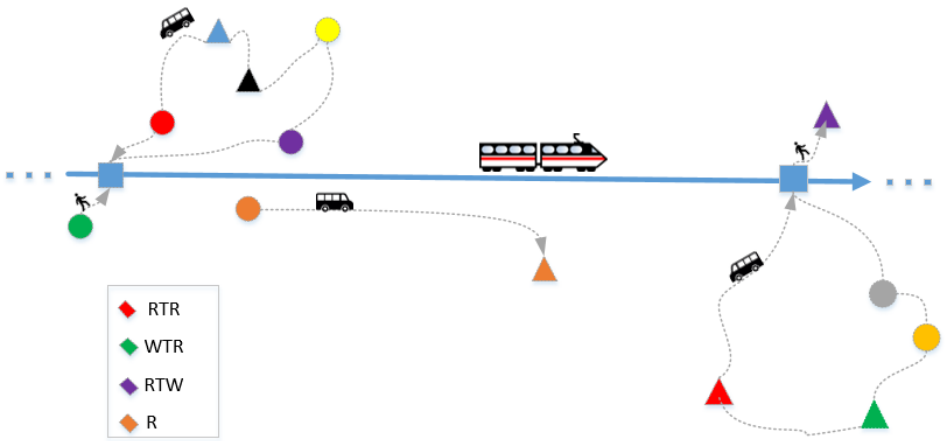


Figure 2.1: Integration of ridesharing and public transport, taken from Ma et al. (2019).

Hall, Palsson, and Price (2018) showed that public transport ridership has increased 2 years after the introduction of Uber in large US cities. However, the effects show heterogeneity over different cities. An important factor seems to be the size of the PT network. Positive effects are larger in geographically bigger networks. The effect of ride-hailing on public transport is more thoroughly explored in Cats et al. (2022). The interplay between ride-hailing (Uber-like service) and public transport largely depends on the quality of the PT network. A lack of suitable PT options leads to a large ride-hailing market share.

In research on how to combine public transport and ridesharing networks, the public transport network can either be a fixed existing network, or it can be flexible. The details for both types are discussed in the following sections.

2.4.1. Fixed public transport services

The high capacity of public transport means that busy routes (where PT is designed to be) require a limited amount of vehicles. The highest capacity lines are usually light rail networks that have fixed routes because of the required infrastructure. The networks have the highest efficiency in moving people for the vehicle miles travelled as they have a large capacity. In a combination with rideshare, the large capacity of public transport leads to more cars being available for first/last mile transport, as

cars have shorter trips on average. This is a positive effect from the combination of PT and rideshare (Ma et al. (2019)).

The situation where rideshare is used solely for first/last mile transport is referred to in the literature as a feeder model. A feeder model is used in Shen, Zhang, and Zhao (2018). A study is performed on first-mile transport to the large capacity light rail network of Singapore. The paper shows the possible advantages of replacing the least busy bus lines with shared vehicles. For areas with very high demand buses are still more effective than ridesharing in cars, because buses are in effect large rideshare vehicles if people are willing to walk to a limited set of meeting points.

A simple version of ridesharing in combination with public transport is performed by Chen and Nie (2017), where ridesharing is restricted to an area around a certain PT line. This uses ridesharing as a strict feeder model. In Maheo, Kilby, and Van Hentenryck (2019) ridesharing vehicles are used in off-peak hours to bring people from low demand areas to high demand bus lines where buses are still efficient.

As shown by Wardman, Hine, and Stradling (2001), people perceive transfers between modes of transport to take significantly longer than they actually do. This effect should also be considered when designing a system with transfers. Next to this, the perceived time in public transport is also influenced by the crowding in bus/metro/etc (A. Tirachini, Hensher, and Rose (2013)). If the information on crowding is available, this could be shared with the ridesharing planning if integrated mobility planning is executed as in Ma et al. (2019). In this method, a trip from the origin to the destination is planned based on available public transport, rideshare and walking. Total travel time and cost are considered and the best combination for a specific request is returned to the passenger. The different modes of transport can be used with a transfer between them, but this is not required. The method in the paper works on-demand, making it attractive for passengers. To keep the computational load limited, public transport only looks at the k nearest neighbours stations. If passengers transfer from public transport to rideshare, the vehicle is only contacted once the PT mode arrives at the station. This could be improved by having the vehicle arrive at the same time

2.4.2. Flexible public transport services

Another line of research on the topic of public transport and rideshare is the option to use flexible bus routes. This either means the bus routes and frequencies shift over the day or respond directly to current demand. This can be done with buses as these don't require special infrastructure, as opposed to the metro for example.

In Banerjee et al. (2021), historic demand is used to plan bus lines to maximize social welfare. Cars are used as feeders into the, to be designed, bus lines. The method has an available budget to open bus lines from a pre-made selection of feasible routes. Pinto et al. (2018) solves a similar problem. Bus routes are selected from a set and the required size of the shared vehicle fleet size is tuned. The optimization is done for a batch of requests spanning 30 minutes where demand is all known beforehand. A budget is allocated for a rideshare fleet based on demand. The method also contains a low-level optimization where passengers are assigned to shared vehicles. This part works on-demand to see if the allocated fleet size is sufficient for the demand. The method penalizes transfers as these are experienced worse by passengers, as described by Wardman, Hine, and Stradling (2001).

The second case, where buses plan their route based on demand, can also be seen as a ridesharing problem with multiple vehicle sizes and transfers between them. This problem is complex to solve as there are a lot of possibilities to consider.

Also, if PT lines are not predictable that would mean they become less accessible for people less into the newest technology. An example of this could be some elderly people that become unable to cope with the new ways of transport and thus lose part of their mobility. On the other hand, it could allow for the symbiotic relationship between PT and ridesharing as described above. With the current algorithms, it would depend on predicted demand as the methods are too slow to solve on demand. This means that the bus schedule is predictable, but has a lot of changes over the day/week/year.

To summarise, ridesharing could be used as an extension to public transit services. A commonly studied problem in literature is the scenario where ridesharing vehicles are used as a first or last-mile solution in a transit system. Another option is to optimize the assignment of buses and ridesharing simultaneously, however, the computational complexity of this problem is a drawback for this framework.

2.5. Additional aspects of ridesharing

In this section, aspects that broaden the scope of ridesharing will be discussed. First, methods that take into account future demand are reviewed. Second, systems that are not centrally controlled will be discussed. Last, the effect of stochastic travel on the ridesharing model is evaluated.

2.5.1. Non-myopic methods

Predictions of future demand can be used during matchmaking to improve the future performance of the ridesharing system. The improved performance can be reached by adding a rebalance step. In rebalancing, vehicles are re-positioned so that they are better able to serve future demand. Rebalancing can be implemented more efficiently with a non-myopic view since rebalancing can be performed during operation by routing them with future requests in mind. By routing occupied vehicles with future requests in mind, a trade-off between current and future service levels should be made.

The prediction of future demand is a complicated task that can be solved in different ways. In A. Fielbaum, Kronmueller, and J. Alonso-Mora (2021) past requests from the system are utilised to predict future requests. This has the benefit that the method can also be used without existing knowledge of demand and could adapt to new travellers using the system regularly. J. Alonso-Mora, Wallar, and Rus (2017) uses historical data to create a probability distribution for requests. Alternatively, X. Li et al. (2020) creates a probability distribution for future demand based on historic demand, processed with neural networks.

The first option to incorporate the predicted demand is by assigning a certain number of predicted requests in the current batch. This creates a bias in the matchmaking towards future demand, at the cost of optimizing for current demand. A similar approach is taken by J. Alonso-Mora, Wallar, and Rus (2017), where future demand is predicted based on a probability distribution function and added to the current demand to be assigned. A slightly different methodology is proposed by Tsao et al. (2019). Tsao et al. use model predictive control (MPC) with a demand forecast to consider future demand. The method proposed is restricted to only two passengers per vehicle to reduce the potential number of combinations that emerge from looking further ahead. Riley, Van Hentenryck, and Yuan (2020) apply MPC on a zone-to-zone version of ridesharing. The reason to reduce the problem to a more course zone-to-zone formulation is to save on computational complexity (smaller network graph). MPC is used to predict how many vehicles should be rebalanced towards each zone based on predictions for future demand. Lowalekar, Varakantham, and Jaillet (2021) proposes the use of zone-to-zone formulation as well. The proposed method makes use of future demand predictions between zones and current demand to be between exact locations. Furthermore, future demand is not allowed to be assigned to a vehicle that carries other passengers at that moment. Together with the zone-based demand predictions, this reduces computational complexity for assigning future demand. In this method predicting demand 15 minutes ahead seems to be optimal. This leads to enough response time without overflowing the assignment algorithm with potential matches.

In addition to introducing bias towards future demand by assigning future requests, another problem emerges. By adding future requests the shareability graph increases in size. Since adding demand causes the number of people that can be assigned together to grow quickly, more combinations are possible for the best assignment. With equal calculation time, this could lead to worse overall assignment for both current and future requests as less convergence might happen.

A second method to improve the future efficiency of the system is by rewarding certain assignments that are beneficial for the future. Different reward strategies are possible in this concept. In A. Fielbaum, Kronmueller, and J. Alonso-Mora (2021), this is incorporated by rewarding assignments towards high-demand areas (that usually have worse service rates). The new cost function equals the old cost function minus the reward. Shah, Lowalekar, and Varakantham (2020) propose to use a neural network to determine the value of a certain assignment concerning future requests. Approximate dynamic programming is used to assign passengers to vehicles in a way that takes the predicted future value into account. An advantage of using such data-driven methods to determine the value of a certain assignment is that they can take advantage of complex reward evaluations. On the other hand, a large dataset is required to determine the values for the state-action space. Next to this, the values assigned to actions in certain states are not necessarily flexible with respect to changes in for example fleet size,

demand forecast or travel times. Haliem et al. (2021) proposes to use a deep reinforcement learning model to assign passengers in a state-action space where future demand is taken into account. The assignment based on the predicted value is done in a greedy way to speed up the assignment procedure.

To recap, with the predictions on future demand the fleet can be allocated more efficiently for future demand. Two main methods of anticipating future demand are evaluated in the literature. In the first method, some future (artificial) demand is added to the current assignment leading to a bias towards future demand and increasing the combinatorial problem by adding requests. In the second method, a certain reward is added in the current assignment phase based on the predicted value for certain routing. The value of this reward can be determined based on strategic insights or data analysis.

2.5.2. Non centralised control

Most ridesharing research assumes a centralized system where all vehicles can be continuously controlled. In reality, this might not be the only available market setting as multiple companies could offer these services. Having competitive services might be a desirable setting as well since it could encourage competition and improvements in service quality.

In Pandey et al. (2019) three different market settings for ridesharing are discussed, these are centralized, cooperative and competitive. The centralized system is the most researched situation in which one controller has complete control of each vehicle in the fleet. This allows the operator to optimize for all demand and make decisions on the service quality versus operational profits trade-off. In the cooperative model, companies make an offer for each current customer in the batch. These bids are combined and the best matches in the batch are made, optimizing the entire batch. Each bid and request in the batch is seen as equal and the collectively best solution is looked for. To achieve this, a central bid processor is required that assigns passengers to bids based on a publicly available algorithm. An important assumption is that all cars make bids only based on their current time required for each bid they place. In the competitive model greedy assignment is performed for each passenger. This means that the different operators still make a bid for each passenger. The bids are solely based on current demand. The passenger is then assigned to the best bid for that passenger. This method can be extended to requested service by a particular company, for example a request to be transported in a vehicle from company X.

The results show that a cooperative system can reach almost comparable performance to a centralized system. The competitive system performs worse on average and performance deteriorates further if personal preference from users for a certain brand is taken into account.

For cooperative systems, matchmaking is performed with a central auction algorithm. An extension to this methodology is proposed in Xi et al. (2021). According to the proposed procedure, both operators and passengers make a bid for a certain trip. The assignment is made according to a two-sided auction. This also introduces the ability to differentiate passengers on their willingness to pay.

To sum up, most systems are controlled by a centralized operator that has influence over all vehicles in the system. Other methods of organizing a ridesharing market are possible, but not widely studied.

2.5.3. Stochasticity in ridesharing

Multiple parameters in ridesharing are often assumed to be fixed, but could prove to be stochastic in reality. An important parameter that is often considered static, but stochastic in reality, is travel time. Other stochastic processes could be the time required to enter/leave the vehicle, trip cancellations and no-shows. In this section, the focus will be on stochastic travel times, as the potential impact is significant.

An important aspect to take into account when planning trips, including ridesharing, is the travel time, as in reality, the duration of a trip is not certain. The travel time can be considered a stochastic variable that can vary based on traffic conditions and other factors. How to consider the stochastic travel times is relevant since it impacts all passengers in the vehicles and the people to be picked up. If transfers to other vehicles, such as public transport are considered, not being late might be even more important.

Stochastic travel times can be studied in multiple ways. Data can be used to determine the average speed in New York City for all moments in the day with a moving average for each hour as output (Simonetto, Monteil, and Gambella (2019); Hörl and Zwick (2022)). This data is used to predict travel time for each moment of the day. In effect, a shift in average speed is used to compensate for the average rush-hour effects to be expected. The resolution of this method is limited to hours and large regions are used to calculate speeds. In Van Woensel et al. (2008) the average speed is calculated based on road and vehicle density parameters. This can be used as an estimation with a higher resolution based on current information (if this is available).

Another option would be to look into the effects of unexpected, local events. This could be done by revealing the travel time of an edge once the edge is entered. This situation would better simulate stochastic effects due to accidents for example.

A third method to deal with travel times is to assume that not all information is present. A certain standard deviation and bias are taken on the expected travel time. The bias could shift over the day to represent the shift in average speed. A distribution could be used to add the effects of local disruptions.

The use of robust optimization with a one-sided uncertainty box is discussed in
Y. Li and Chung (2020). The proposition is that arriving earlier than planned is fine. The one-sided uncertainty box is meant to compensate for travel delays.

Besides travel time, there could also be other stochastic events that influence time requirements in ridesharing. A relevant parameter is the service time required for passengers to get in or out of the car. This is mainly important as people can be late for the pick-up time. The effects of people being late lead to accumulating delay (Kucharski, Fielbaum, et al. (2021)). Last-minute trip cancellations could also be a source of uncertainty. However, this leads to shorter travel times in general as less detour is needed to make the pick-up.

A method to allow for transfers under travel time uncertainty is described in
Kim and Schonfeld (2014). The main idea can be summarised by taking a buffer that is a function of the remaining travel time. So for stops close by a smaller buffer can be used as the risk of large disruptions is smaller. A second aspect researched is who should wait for whom. In the paper, high-demand bus lines and low-demand bus lines are used. The most logical solution is to let a bus with low occupation wait for a bus with high occupation.

As mentioned, stochastic travel times are especially important in case the connection to public transport is relevant for an itinerary. The public transport side of the transport can also be early/late. Different modes of transport can here have different sources of stochastic travel times as they can be of different modes. The travel time for a metro does not depend on congestion but is dependent on delays in the network itself. Buses on the other hand might be affected by traffic, but could also tend to depart early if the bus is ahead of schedule.

Connecting to these transfers could be accomplished by adding larger time buffers. This is not the optimal solution as additional constraints limit the solution. The other option is to accept the risk or accept the waiting cost as part of the collaboration with public transport.

Communication between public transport and ridesharing services might provide better overall service for the passenger. This communication could be straightforward if there is one mobility provider but can become more complex if multiple companies have to share their data. The amount of time gained by communication would depend on how many of the passengers would use a multi-modal route and the gains made by communication.

To sum up, multiple stochastic parameters in ridesharing influence the travel time. The most straightforward way to incorporate congestion effects is to shift the mean velocity in the network over the day. More complex measures are also possible but come at a computational cost. An important reason to study stochastic travel times is the inclusion of transfers in a rideshare model.

2.6. Research gaps

Based on the objective to improve the performance of ridesharing systems, multiple research gaps have been identified that can turn out to be relevant. Different research directions are discussed in this section and their potential upside is examined. The research gaps are identified to the best of my

knowledge.

Centralised system or competition

In research on non-centralized ridesharing systems, operators are currently assumed to only optimize for current demand.

A potential research gap would be to study incentives under different market settings. This corresponds with company strategies on where to position vehicles/how to bid on certain (less profitable) trips. Research can be performed on how to design incentives in the system to align operator objectives with the best outcome that maximizes social utility.

The value of waiting, transfers from public transport to rideshare and back

As indicated in sections 2.4 and 2.5.3, some research has been done on transfers between different modes of transport and stochastic travel times. Being late for a transfer due to congestion is unfavourable, as it means additional trip time. Vehicles could potentially wait for each other if the delay is small enough.

A research gap is identified in the communication between different vehicles (including PT) for transfers. Communication could make explicit how long and with what profit waiting can be considered. Insights could be created to show the potential gains that are available from communication.
An extension to this topic and the previous topic would be if the value of waiting changes in a different market setting (non-centralized). Since different parties could have diverse priorities, some incentives could need specific studying to optimize system performance.

Combination of pre-booked and on-demand requests

In the on-demand version of ridesharing, new demand is revealed over time as it emerges. A potentially interesting research area is to add some known future demand in an on-demand market setting. A reason why this is relevant is provided in Ma et al. (2019). In an integrated mobility service system, people want to move from their origin to their destination. This could thus also include rideshare after public transport. In this situation, some demand for rideshare is known in advance. This information could be used to improve the rideshare outcome and thus also benefit the passenger, for example with less waiting time.

Research can be done on the gains that can be achieved by the ridesharing system from knowing some demand upfront. Three important aspects can be studied on this topic. First of all, how much of the upcoming demand is known. This can be considered alongside how far upfront this information becomes available. Secondly, how system performance is impacted if there is any kind of immediate benefit for the user. Examples of immediate benefits that could be offered to people pre-booking could include a no-rejection rule or less waiting time. Lastly, the impact of the spatial and temporal distribution of known demand can be studied. If there is a certain benefit of booking a trip in advance, then areas that currently experience the worst service might be expected to use this option more. Another spatial effect to be studied is related to public transit stations as both first and last-mile travel from transit has a spatial aspect. Transit systems could also lead to temporal differences when these known requests are used.

Heterogeneous service

In the reviewed research there is a general assumption that all vehicles in ridesharing are identical. In real life, this does not have to be the case however.

It could be interesting to do research on fleets with different capacities. This is partly done in section 2.4.2, where buses have a flexible route but do assume vehicles as a feeder. This makes for a system with two-vehicle sizes, with additional constraints on the larger vehicles. In the current on-demand algorithms there is experimentation with different vehicle capacities, but not with mixed-capacity fleets. This might also require a heuristic to make the most efficient use of the larger vehicles or allow transfers.

An interesting research line could be to consider heuristics that utilize the benefits of different vehicle capacities in a single fleet. Fleet sizing for mixed-capacity fleets is another subject that is not yet studied extensively in the literature.

3

Methodology

A state-of-the-art on-demand ridesharing model is used as the starting point for this research. The model is based on the work in Alonso-Mora et al. (2017).

In this chapter, first the problem formulation is provided. This is followed by a section on the algorithm of Alonso-Mora et al. Thirdly, a section on the adjustments to the method to handle the challenges of reservations is included.

3.1. Model formulation

The goal of on-demand ridesharing is to assign a set of passenger requests to a set of vehicles with minimal overall cost and within determined constraints in each time step. Requests are collected in batches and assigned collectively. The problem takes place on a directional graph $G = (N, E)$, with N the nodes in the network and E edges connecting nodes. The edges are directional and weighed based on the average travel time needed to pass the encoded road section.

The simulation works with timesteps of 30 seconds, which is chosen because it offers a balance between the desirable on-demand characteristics for users and the possibility to combine requests for a better assignment. Requests are collected during a time step and assigned in batches. This means that requests have to wait for the next iteration of the model before they are considered. The maximum time a request has to wait before the assignment method starts is thus 30 seconds. Once the request is added to the list of requests to be assigned to a vehicle, the algorithm has another timestep available to solve this assignment. Vehicles that have movement instructions from the previous iteration do move during this time. After the assignment is made, the vehicle instructions are updated. For example, a request comes in 12 seconds after the simulation start. This request will be assigned in a batch with all other requests that came in before the 30 seconds mark. The assignment will be made during the time step between 30 and 60 seconds. At the 60 seconds mark, the final assignment will be communicated to all vehicles and their trajectories are updated accordingly.

Requests that will be picked up before the next time step (in this example at 90 seconds) will not be considered for re-assignment. All other requests, both assigned but not yet picked up and unassigned, will go back into the pool of requests. The pool of requests will be expanded with new requests coming in. The request pool will then be used for the next iteration of assignments. Requests that have passed their maximum waiting time are taken out of the request set and counted as unserved.

Requests are assigned to cars based on the objective to minimize overall waiting and detour time while staying within the constraints of the system. Also, the number of rejected requests is minimized, according to a weight function so that it can be compared with induced waiting/detour times.

Waiting time for request r w_r is the time between when a request emerges r_e and when the request is picked up r_p, $w_r = r_p - r_e$. Detour time is the delay a passenger on board a vehicle experiences compared to an unshared vehicle. The detour time for request r d_r is given by the drop off time r_d minus the pickup time minus the minimum travel time r_{min}, $d_r = r_d - r_p - r_{min}$. The waiting time and detour time combined give the total delay time for a request δ_r, $\delta_r = w_r + d_r$.

The goal of the assignment is to minimize equation 3.1. In this equation, the sum of the detour times for all passengers on board $\mathcal{P}_v$ a vehicle, the total delays for all assigned requests $\mathbb{R}_{ok}$ plus the rejection penalty Π_r for all requests that are not assigned $\mathbb{R}_{no}$, is minimized. This is done for each vehicle v in the set of all vehicles $\mathbb{V}$.

$$\sum_{v \epsilon V} \sum_{r \epsilon P_v} d_r + \sum_{r \epsilon \mathbb{R}_{ok}} \delta_r + \sum_{r \epsilon \mathbb{R}_{no}} \Pi_r \tag{3.1}$$

This equation is slightly different compared to the original paper from Alonso-Mora et al. (2017). The original paper considers the total delay for passengers already on board. In the proposed implementation, only the detour time for passengers already on board is considered. This small difference means that you only consider the cost that you can still influence. A downside is that passengers that had a long waiting time are now less compensated for that in the assignment procedure when their detour time is considered. The overall costs of trips will as a consequence be lower. This can have an impact on the rejection penalty that should be set. Also, assigning trips to vehicles with more passengers becomes cheaper compared to the original method. This decision is made as a consideration for the heuristics to be designed. This is discussed further in section 3.3.3.
The model's objective can be reached by providing the best assignment from the set of requests to the set of vehicles. This problem is solved with an integer linear problem (ILP). To formulate the problem as an ILP, the cost of assigning trips to vehicles has to be calculated. A trip $\mathcal{T}$ is a group of requests that can be assigned together to the same vehicle.

A function to determine the best route and corresponding cost for a trip assigned to a vehicle is designed. The cost of a trip consists of the detour time for the passengers on board, plus the total delay for the requests in the trip. The formula for the cost of assigning trip $\mathcal{T}$ to a vehicle v is given in equation 3.2. The function will return "unfeasible" in case the constraints for one of the individual requests is not respected. This means that the trip will not be added to the potential set of trips. If there are multiple feasible outcomes, the one with the lowest cost is selected.

Limits on individual constraints ensure the maximum waiting time Ω and maximum detour time Δ are not exceeded. Furthermore, requests have to be picked up before they can be dropped off. Lastly, the vehicle capacity can not be exceeded at any time during the trip. Since the waiting time for passengers on board can not be impacted anymore (and is guaranteed to be within the constraint already), this is not considered in determining the feasibility of a trip. The limits on waiting time are considered for all requests in the trip. The detour times are considered for all passengers on board and the requests in the trip. Furthermore, a check is done that there are at no time during the trajectory more people on board than the vehicle capacity.

$$\begin{aligned} & cost(\mathcal{T}, v) = \sum_{r \in \mathcal{P}_v} d_r + \sum_{r \in \mathcal{T}} (w_r + d_r) & \\ \text{subject to} \quad & d_r \leq \Omega & \forall r \in \mathcal{P}_v \cup \mathcal{T} \\ & w_r \leq \Delta & \forall r \in \mathcal{T} \\ & r_p < r_d & \forall r \in \mathcal{T} \end{aligned} \tag{3.2}$$

The trips are used to create a graph linking requests via trips to vehicles. The nodes in this graph consist of the requests, the trips and the vehicles. There are two kinds of edges in the graph. First, there are edges between requests and trips that contain these requests. Requests can be part of multiple trips and trips can consist of multiple requests. Secondly, there are edges between trips and vehicles that can serve that particular trip. Again, a trip can potentially be served by multiple vehicles and vehicles can be matched to multiple requests. The cost for each edge between trips and vehicles is given by equation 3.2.

The constraints in the ILP formulation will ensure that each request is assigned to 1 vehicle or ignored. Also, each vehicle will have at most 1 trip assigned. The ILP assignment has two sets of decision parameters. First, $\epsilon_{i,j} \quad \forall e(\mathcal{T}_i, v_j)$ is a parameter for all edges between trips and each vehicle that can serve that trip. This is a binary parameter, whether a trip is assigned to a vehicle or not. Second, $\mathcal{X}_r \quad \forall r \in \mathbb{R}$ is a parameter if a request is assigned to a vehicle. This is also a binary parameter since a request can only be assigned to 1 vehicle or not be assigned. $\mathcal{X}_r = 1$ means that a request is not assigned.

The ILP has the number of edges in the $e(\mathcal{T}, v)$ set, plus the number of requests as decision parameters. The number of constraints is equal to the number of vehicles plus the number of requests.

Three subsets are used in this formulation, namely $\mathbb{T}_{v=j}$ is the set of trips that can be served by vehicle j. $\mathbb{T}_{\mathcal{R}=r}$ is the set of trips that contain request r. $\mathbb{V}_{\mathcal{T}=i}$ is the set of vehicles that can serve trip i

The ILP formulation is given in equation 3.3.

$$\begin{aligned} \text{minimize} \quad & \sum_{\mathcal{T}_i \in \mathbb{T}} \sum_{v_j \in \mathbb{V}} c_{i,j} * \epsilon_{i,j} + \sum_{r \in \mathbb{R}} \Pi_r * \mathcal{X}_r \\ \text{subject to} \quad & \sum_{i \in \mathbb{T}_{v=j}} \epsilon_{i,j} \leq 1 && \forall v_j \in \mathbb{V} \\ & \sum_{i \in \mathbb{T}_{\mathcal{R}=r}} \sum_{j \in \mathbb{V}_{\mathcal{T}=i}} \epsilon_{i,j} + \mathcal{X}_r = 1 && \forall r \in \mathbb{R} \end{aligned} \tag{3.3}$$

The ILP uses a greedy solution as an initial guess. The solution will be improved over time. The ILP problem will be solved using the commercial Gurobi solver under an academic license.

Incorporation of reservations

Reservations can be included in the existing formulation. Reservations can be added to the set of requests to be assigned as all other requests. The difference with on-demand requests is that reservations can not yet be picked up once they are revealed to the system. An additional parameter is added for the reveal time of a request r_r. For normal requests, $r_r = r_e$, for reservations r_r is the time the request is added to the set of requests to be assigned. To ensure the correct implementation of reservations, an additional constraint is needed to determine the feasibility of a trip. This extra constraint states that a trip is only feasible if all requests are picked up as soon as, or after they emerge. Equation 3.2 should thus be updated to include this extra constraint. The updated equation is given in 3.4.

$$\begin{aligned} & cost(\mathcal{T}, v) = \sum_{r \in \mathcal{P}_v} d_r + \sum_{r \in \mathcal{T}} (w_r + d_r) \\ \text{subject to} \quad & d_r \leq \Omega && \forall r \in \mathcal{P}_v \cup \mathcal{T} \\ & w_r \leq \Delta && \forall r \in \mathcal{T} \\ & r_p < r_d && \forall r \in \mathcal{T} \\ & r_p \geq r_e && \forall r \in \mathcal{T} \end{aligned} \tag{3.4}$$

3.2. Existing algorithm

An important challenge in making the system work is in determining the cost of trips. Since there is a large computational cost to determine trip costs, this has to be executed efficiently and in an effective order. To reach this goal, multiple steps are taken. First, the requests are grouped in the RV graph, which is then extended to the RTV graph. This graph is used as input for the ILP formulation to assign requests to vehicles. Afterwards, idle vehicles are rebalanced for improved performance.

The steps taken are visualized in figure 3.1. Requests are linked with vehicles and if relevant pairwise couples. Then requests are combined into trips that are in turn linked to vehicles. Finally, trips are assigned to cars and idle vehicles are used to rebalance. The details of each step are explained in the following sections.

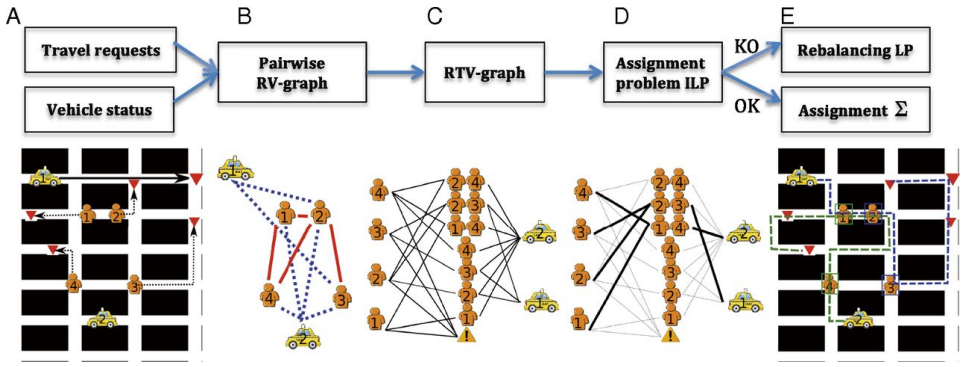


Figure 3.1: Visualization of the steps taken in Alonso-Mora et al. (2017).

3.2.1. Request Vehicle (RV) Graph

The main step taken in creating the RV graph (initial combinations of requests and vehicles) is to connect requests to vehicles that could serve these individual requests within the constraints. These connections are built upon further in the next step. Secondly, in the RV graph pairs of requests that could be serviced together are connected. This is done by computing the cost for an empty virtual vehicle starting at one of the two nodes. If the result is a feasible trip, the pair of requests is accepted. Pairs of requests that can not be serviced by an empty vehicle starting at one of their start locations can in no case be serviced by a real vehicle. This check is a method to reduce the number of combinations that have to be studied for feasibility in later steps. This is implemented since checking these trips for multiple vehicles is a waste, since the outcome will remain unfeasible. This method is only implemented for trips of size 2 (a trip consisting of 2 requests). The same method could also be applied to larger trip sizes but might be less beneficial there. The potential number of trips for larger sizes increases exponentially, while constraints on the vehicles will make most of these trips irrelevant to consider anyway.

Another consideration is that for a trip to be feasible, all sub-trips have to be feasible individually. In this context, it means that if a request can not be serviced by any vehicle it will not be considered for the pairwise part of the RV graph. This is because the pair would never be able to be assigned to a real vehicle, even though the pairwise cost might be feasible. This is because the pairwise cost is calculated for an empty vehicle starting at the start node. In the RTV stage of the process, this property means that there is a computationally cheap way to determine if it is reasonable to compute the cost of a trip.

3.2.2. Request Trip Vehicle (RTV) graph

In this step, requests are combined into larger trip sizes and matched to available vehicles. This is implemented on a per-vehicle basis. The implementation is multi-threaded to speed up the calculation time (as are the other steps), this does not affect the steps taken. Single requests that can be added to vehicles are already calculated in the RV graph. In the RTV graph, requests are bundled into trips and then connected to vehicles that can serve these bundled trips. For each vehicle, trips are calculated in increasing size, as long as the calculation time limit is not met and there are more options to consider.

The process of making the RTV graph is the same for each vehicle. Initially, the RV graph is checked for single requests that can be added to the vehicle. Secondly, all possible combinations of two requests are created, based on the individual requests that can be served by the car. If the combination of the two requests is feasible according to the check on the pairwise feasibility in the RV graph, the cost of serving that trip with this vehicle is calculated.

Trips of larger sizes are created if the available computation time has not been yet been exceeded. This is done by taking two trips of size k-1 with k the size of the trip to be made. So for a trip of 3, two trips of size 2 are considered. If there is one request of overlap between the trips (so 3 different requests), it is checked if all possible subsets of this trip exist. For a trip of size 3, there are 3 possible subsets of size 2. More generally, for a trip of size k, there are k subsets of size k-1. If all these subsets can be served by the vehicle, the cost for serving the trip is calculated. This operation to check the feasibility of the subsets is relatively cheap as it is a lookup operation for edges that can be served by

this vehicle. If not all subsets can be served, there is no need to calculate the cost since the trip will be infeasible anyway.

The same process is used to determine which combinations of trips of size 3 are used to create trips of size 4. Trips of size larger than 4 are not considered, as the cost for calculation increases exponentially, while the usefulness of the calculation is limited in the experimental settings applied. The possibilities to assign larger trip sizes are influenced by the overlap in demanded trajectories for passengers and vehicle capacity.

It is important to consider the order in which trips are calculated since there is a limited time to compute trips. The method to determine which requests to combine is based on the idea of mixing the trips with the lowest cost first. This is achieved by ordering the cost for the trips of size k-1. The ordered trips can be represented as a matrix as in figure 3.2. The red squares do not have to be considered since they are a combination of the same request. This can not lead to the correct trip size. The purple squares are not considered since they are the same as the squares opposite the diagonal. The values on the x and y axis are the trip number, in order of cost to assign to the current vehicle. All these trips are of size k-1. The trip numbers are for illustration purposes only.

The combination of trips to be checked can be implemented horizontally through the matrix, as in figure 3.3a, or vertically as in figure 3.3b. The arrows in the figure indicate which trips are first tried as a combination. The advantage of running through the matrix vertically over horizontally is that for the same amount of tries, more low-cost trip combinations are tested as a combination. A trip with a low cost indicates that there is more available delay before individual constraints on waiting/detour time are violated. Therefore, it is more likely that these trips are able to be combined into larger trips.

In a 100x100 matrix (meaning 100 feasible trips of size k-1 for this vehicle) after considering 80 combinations, the best trip has been combined with 80 trips of increasing cost with the horizontal setting. For the vertical setting, the 12 lowest-cost trips have all been cross-examined. This is no guarantee for finding better options, but a heuristic to find better options earlier on in the process since trips with a high cost indicate a lot of delays. The high delay reduces flexibility as it is more likely you would reach the constraint for waiting or detour time for one of the requests.

If a trip is viable and not yet in the RTV graph, the trip node is added. The edges linking the requests that make up the trip are added as well. If the trip node was already present in the graph (because it was added when calculating the edges for another vehicle), only the edge is added to the graph.

Once the time limit for the calculation of the RTV edges for a specific vehicle is reached, the edges for the next vehicle are calculated.

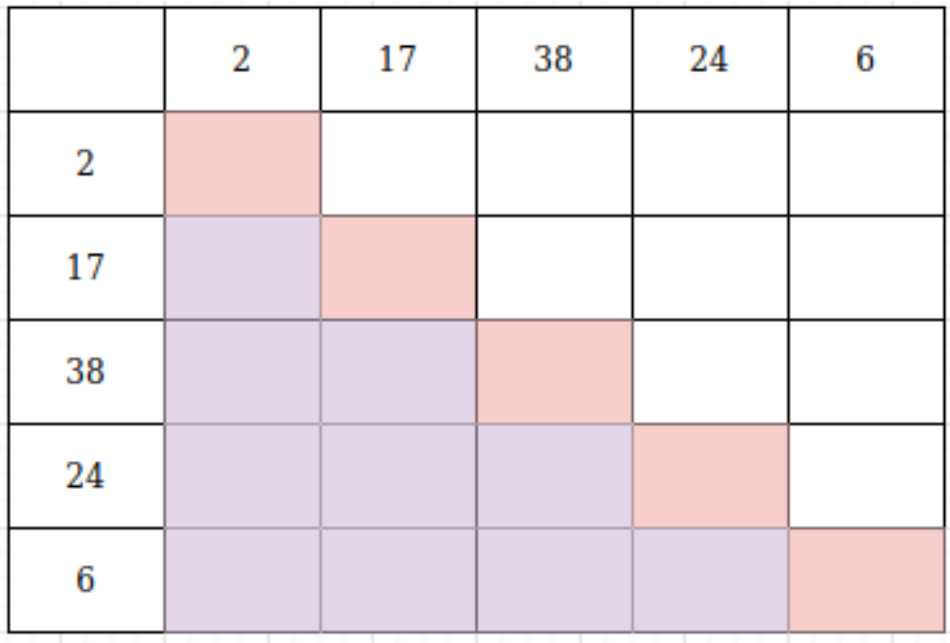
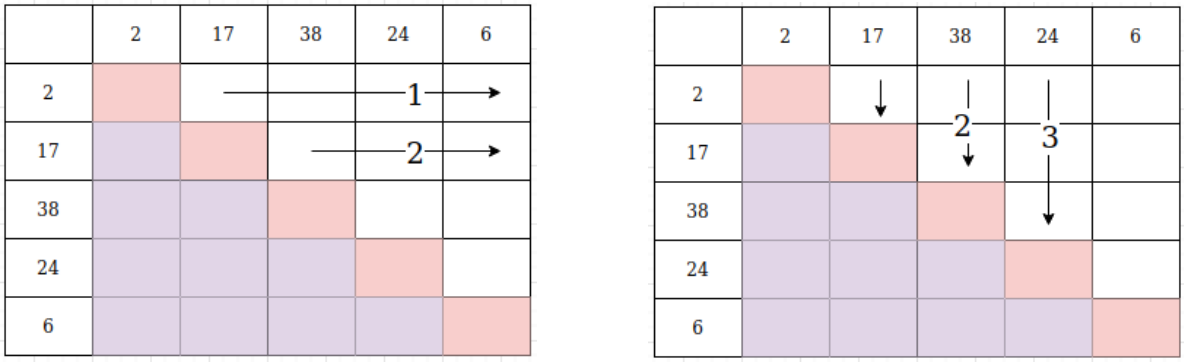


Figure 3.2: The different trips that can be combined (numbers are for illustration purposes only).



(a) Combinations of trips horizontal through the matrix.

(b) Combinations of trips vertical through the matrix.

Figure 3.3: Different orders to make combinations from trips.

3.2.3. Solve

In the solve step, trips are assigned to vehicles according to the objective function and within the constraints. The problem is implemented as an integer linear programming problem (ILP). In this formulation, each trip is assigned to a vehicle or not. This is a binary process.

This model can be visualized as a matrix with each row a vehicle and each column a trip. Based on the constraints trips can, or can not, be assigned to a vehicle. This means that the value can be 1 or 0 if there is a potential match and 0 if there is no potential match between the vehicle and the trip. The constraints on trips assigned to vehicles mean that the sum of each row is 1. The constraint on each request being assigned to at most 1 vehicle means that the sum of all columns (trips) where that request is part of, is at most 1.

Since most trips can not be serviced by a lot of cars, the matrix will be very sparse and thus not efficient to model as a matrix. If it would be modelled as a matrix, a lot of variables are added that have to be 0 by constraint. Therefore, a better implementation only adds the variables that are potentially non-zeros.

The initial guess for a solution is based on a greedy initialization. Trips are assigned in decreasing size and with increasing cost over the vehicles. The principle is to serve as many requests for the lowest cost. Having an initial guess can improve the solving time for the ILP solver. Furthermore, it guarantees a feasible solution which is incrementally improved upon.

3.2.4. Rebalance

In the rebalancing step vehicles that are not assigned to requests and don't have passengers on board are considered available for rebalancing. The goal of rebalancing is to move vehicles from areas where they are not useful to areas where they are potentially more useful. This is done by re-directing those vehicles to areas where there currently is unserved demand. These vehicles are not able to serve that particular demand (otherwise they could have been assigned to that demand), but the idea is that it will allow more requests to be served in the future. This is because areas with unserved demand are more likely to also contain demand in the future that can then be serviced by the vehicle.

This problem is implemented as a separate ILP to solve. The objective is to minimize the total travel time for all idle vehicles and still assign them to nodes with unserved demand. Constraints are that unserved demand nodes can be assigned to at most 1 vehicle and that each vehicle can be assigned to 1 unserved demand origin node.

3.3. Additions to algorithm

The inclusion of reservations in an on-demand ridesharing system brings both computational challenges and potential upside. Adjustments are made compared to the existing algorithm to accommodate reservations computationally and increase the potential benefit. This part of the methodology contains the core of the practical research and findings of this thesis.

3.3.1. Challenges from reservations

First of all, reservations that are available significant time in advance could in the worst case be available to be added to each trip of a certain vehicle. In this case, the number of potential trips would be multiplied by 2, plus 1 (only serving that reservation) for each reservation available to the vehicle. This phenomenon is shown for 1 reservation in figure 3.4, where a blue reservation is added.

A second effect is that reservations can be assigned to more vehicles than on-demand requests. Reservations are revealed to the system longer in advance, thus allowing more vehicles to serve them. Therefore, extra trips have to be computed for more vehicles compared to normal requests.

Not only are there more trips, but the trips also become longer and thus significantly more expensive to compute. As shown in table 3.1a and 3.1b, larger trip sizes are especially expensive as you compute more new passengers. Longer trips are more expensive since there are more possible permutations of actions in the trip.

Since reservations can be added to more trips than on-demand requests, the inclusion of reservations has a significant impact on the amount of larger trip sizes that are available.

Lastly, reservations stay in the request pool longer than on-demand requests. This means that the potential trips are computed more often as well. An on-demand request with a maximum waiting time of 5 minutes would at most be considered in 10 iterations (with time steps of 30 seconds). A reservation revealed 10 minutes in advance would be considered between 20 and 30 iterations.

On the other hand, the impact of a reservation that is placed far in advance could be limited with logic checks to keep the number of permutations somewhat limited. For example, a reservation that is revealed far in advance can be limited to being picked up and dropped off after all other passengers are served. In this case, the number of permutations would not grow. However, most situations will be somewhere in between. They can be added to a lot of trips and have multiple possible permutations. The growth in practice will be discussed in the results section 5.

It should be noted that additional normal requests have a scaling effect on the computational load as well. However, reservations do scale much worse in practice.

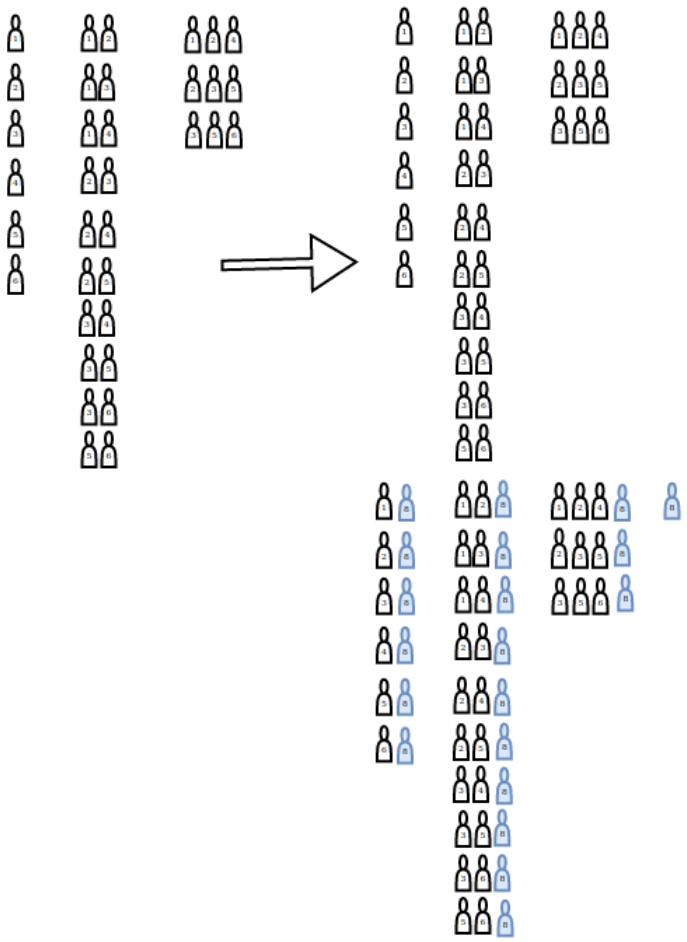


Figure 3.4: Worst case growth in RTV size if a blue reservation is added.

Methods to deal with the increased computational load from the inclusion of reservations have to be implemented. In this section, multiple suitable methods are explained. The biggest gains can be made in how to calculate the cost of a trip and which trips to calculate. Furthermore, additions to the model are made to increase the potential benefit of reservations and to reward reservations made. Lastly, reservations are only revealed to the system a set time in advance. In case reservations are revealed to the system 10 minutes in advance, a reservation placed a day in advance will be treated the same as a reservation placed 12 minutes in advance. Both reservations will be revealed to the system 10 minutes in advance and are not differentiated in any way.

3.3.2. Travel function

In the travel function, the cost of adding a certain set of requests to a vehicle is calculated. The cost in this case is the sum of the waiting times for all passengers that are not yet in the car and the sum of the detour time for all passengers, including those in the car. Trips are made by checking all the feasible permutations in which pick-up and drop-off can happen. Only the nodes where pick-ups/drop-offs happen are modelled. The time to move from node to node is read from a look-up table. The induced delays are added up and compared to the constraints at each step of the route. The route determination has the option to wait at nodes in case this is beneficial for the overall delay and does not violate any of the constraints.

The most expensive part of determining the cost of a trip is to compute the cost of all feasible permutations. A permutation in this case means the sequence of pick-ups and drop-offs for all passengers. The goal is thus to limit the permutations to be checked. Two methods are considered in this section.

The first decision is to only compute trips based on an insertion heuristic. The second step is to use

logic checks to determine if a certain permutation of a trip is sensible to compute.

The order in which the passengers can be dropped off is limited. Passengers that are already in the car do not shift in the drop-off order. This is an insertion heuristic, where new requests are inserted in the existing order, without shuffling the existing order. This heuristic limits the number of combinations that have to be checked for cost. On the other hand, potentially better options are not considered because of this heuristic. Since the drop off order is set, the number of feasible permutations is reduced by the factorial of “Passengers_on_board”.

A reason to use the insertion heuristic is to speed up the calculation of the travel cost. In table 3.1a and 3.1b the potential number of feasible, unique permutations are shown for the insertion heuristic and a complete shuffle respectively. In this number of permutations, the constraint that requests have to be picked up before they can be dropped off is implemented. The other logic checks are not used, as they depend on the specific scenario. The results are thus a worst-case scenario. The difference between the numbers in table 3.1a and table 3.1b is the factorial of “On_Board_passengers”.

An important conclusion is that more new passengers explode the number of options because 2 parameters are added, the origin and the destination locations. Passengers already on board only add 1 parameter, their drop off moment.

		New Passengers			
		1	2	3	4
On board passengers	0	1	6	90	2520
	1	3	30	630	22680
	2	6	90	252	113400
	3	10	210	7560	415800
	4	15	420	18900	1247400

(a) Number of potentially feasible permutations with the insertion heuristic.

		New Passengers			
		1	2	3	4
On board passengers	0	1	6	90	2520
	1	3	30	630	22680
	2	12	180	5040	226800
	3	60	1260	45360	2494800
	4	360	10800	453600	29937600

(b) Number of potentially feasible permutations without insertion heuristic.

Table 3.1: Number of unique feasible permutations with and with the insertion heuristic with a vehicle capacity of 4.

Multiple logic checks can be done to determine if it is sensible to compute the cost for a certain permutation. Permutations are different sequences in which pick-up/drop-off actions are planned. The initially performed checks are listed in order below:

1. The permutation does not contain an infeasible subset
2. No drop-off can happen before the pick-up of that request (constraint)
3. The order of drop-offs for passengers already in the car remains the same (insertion heuristic)
4. Reservation sequence limitations
5. The vehicle capacity should not be violated at any time during a trajectory (constraint)

In case a subset of a permutation is determined to be infeasible, all other permutations starting with this subset will also be infeasible. For example, if the permutation 1-2-3-6-5-8-7 is infeasible because action 6 happens before action 5 (say, 5 is the pick up of a passenger and action 6 is the drop off for that passenger), then you know already that the permutations 1-2-3-6-8-5-7, 1-2-3-6-8-7-5, 1-2-3-6-5-7-8, etc are also infeasible. This is because action 6 still happens before action 5. It is easy to check if a permutation starts with an infeasible subset because of the encoding. This check is performed first, as it makes other checks unneeded if it is a limiting factor.

No drop-off can happen before the request’s pick-up has happened means that only passengers that were already on board can be dropped-off. This excludes permutations where a passenger would be dropped off before being picked up.

The order of drop-offs for passengers already on board is set, because of the insertion heuristic. If new passengers are picked-up during an iteration, the order will be updated based on the planned moment to drop off the new passengers.

For reservation sequence limitations, there is a limit on where reservations can be in the sequence of pick-up/drop-off orders. The time a reservation emerges (can be picked up) has to be before the last potential drop-off time for a request still on board that is later in the sequence. For example, a request that emerges in 10 minutes can not be placed in the sequence before a request that has to be dropped off within the next 8 minutes. The last potential drop-off time is the time a passenger got on board plus the minimal travel time for that passenger plus the maximum detour time. This check can affect reservations and thus limits the computational cost of including reservations.

Furthermore, the vehicle capacity should not be violated at any time during a trajectory to be considered. Since pick-ups/drop-off permutations are considered, it is straightforward to count the occupancy beforehand.

Once a permutation passes the logic tests, the actual feasibility is checked based on waiting and detour time constraints. If the permutation turns out to be infeasible, the subset leading up to that point is added to the list of infeasible subsets. So if a certain sequence leads to exceeding the detour constraint of request 2, the part of the sequence leading up to this point is added to the infeasible subset. The permutations that include this part (but a different sequence after this part) will also be infeasible as the same constraint will be violated anyway.

The same idea is applied to the other logic checks. If a permutation is not feasible according to one of the checks, the subset leading to that point is added to the infeasible set. Sn efficient stopping criterion is available because of the encoding used.

The road network is pre-processed such that the shortest path between every node is calculated in advance. Therefore, only the start and end nodes of the requests in the trip are considered. Since the route between these nodes can be filled in later, it is not relevant to model this at every permutation. The distance between each pair of nodes is given in travel time and read from a look-up table.

For completeness, a speed factor is included in determining the used time. This would make it possible to simulate with different average speeds. For example, a scenario where all vehicles can travel 20% faster can be simply simulated.

If there is no feasible route for a certain trip, the function will return that the trip is unfeasible. The edge will thus not be added to the RTV graph. This is a guarantee that all trips that are eventually assigned will be feasible.

3.3.3. Heuristics

Multiple heuristics have been designed to reduce the computational load while maintaining as much as possible of the advantages from reservations. The goal of the heuristics is to reduce the number of trips to be computed. The different heuristics will be discussed in this section. In the section on results, the effect of the different heuristics is compared to see where the most gains are possible. Heuristics are designed to be simple to be efficient in the implementation and to ensure the results can be more generalized and trends are valid over multiple data sets.

The designed heuristics can be split into two categories. First, some heuristics can apply to all requests/vehicles. Secondly, there are heuristics that specifically apply to reservations.

Heuristics applied to all requests

Maximum requests per vehicle A straightforward heuristic to apply would be to sort the cost of assigning trips of size 1 to each vehicle and only select the X best requests for combinations in larger trip sizes for this vehicle. X is here a variable for how many requests can be assigned to a vehicle.
This heuristic can also be applied to larger trip sizes. The method is also implemented to prune the RTV graph at trips of size 2.

The edge between a vehicle and the not selected trips is still added in the RTV graph, as the cost for combinations that are not used for larger trip sizes is already calculated. The working principle of this method is that fewer requests become available for larger trip sizes, thus limiting the number of large trip sizes that have to be calculated. An underlying assumption is that cheap trips can be combined into longer trips, whereas expensive trips come closer to the waiting/detour time constraints and thus

are less likely to be part of successful longer trips.

Maximum vehicles per request Another option to limit the number of large trips is to constrain the number of vehicles a request can be assigned to in each iteration. In this method, the cost to assign a trip of size 1 to all available vehicles is calculated. Only the Y vehicles with the lowest cost can use this particular request to create trips larger than size 1. Y is here a variable to be tuned. The trip stays available for assignment to the other vehicles since the cost is calculated already. The main idea behind this heuristic is that requests will usually be serviced by vehicles with a low cost. Requests that have a low cost for a specific vehicle are a better match with the location of the vehicle and the destinations of current onboard passengers for this vehicle. Requests with a low cost impose less constraint on trips that they are matched with and thus could on average be successfully matched with more other trips.

Limit the trip size Based on the results from table 3.1a, large trip sizes (new passengers) cause longer computation times. To reduce this problem, a limit on trip sizes can be imposed. A strict cap on the trip size to be assigned is the most straightforward way of decreasing the computational cost and also the least refined. A more refined approach where the maximum trip size is calculated could be based on the current number of passengers. It is not unreasonable to assume that vehicles with more onboard passengers are on average a worse match for large trips because more constraints have to be considered. An example of this method would be a limit on the new passengers plus current passengers on board.

Heuristics applied to reservations only

Maximum vehicles per reservation Similar to the “Maximum vehicles per request” method, only applied solely to reservations. The idea is that reservations cause the increase in computational time compared to the base scenario. Therefore, restricting the growth in computation time can be limited by restraining reservations to be assigned only to a set of vehicles and thus having fewer vehicles with more large trip sizes to compute. This heuristic is applied again at each iteration.

Restrict future requests to a set of vehicles In this heuristic, reservations are restricted to a set of vehicles that can serve the request. The set of vehicles is based on the vehicles with the lowest cost of serving this vehicle once it is revealed to the system. Contrary to the heuristic before, the set of vehicles is not updated over time. This does impose a stricter constraint on RTV growth, as not the best X vehicles are used to construct large trips, but a pre-determined set that was best at the moment the requests were revealed.

This method does not consider other reservations that have been assigned to this vehicle, but only passengers that will be on board at the start of the next iteration are included. This decision is made since requests that are not yet picked up can still be assigned to other vehicles and then be irrelevant to the cost determination. An additional parameter to this heuristic is whether or not a request is put back in the general pool so that it can be assigned to all vehicles a certain time before, or once, it emerges. This allows for a middle way where some vehicles will consider this request once it is revealed, but if the system finds a better alternative shortly before it emerges that can be allowed.

3.3.4. Pledged service constraint

An additional constraint that can make ridesharing more user-friendly is implemented. The effects will be discussed in the results section.

The constraint states that all passengers that have been assigned to a vehicle before, and thus expect to be served, have to be served by the system. This is implemented by formulating the constraint that all requests that were assigned before, have to be assigned to a vehicle in this iteration as well. Since requests that are picked up are removed from the list of requests to be assigned, this phrasing is equivalent.
Two new subsets are defined for this constraint. $\mathbb{R}_{db}$ for demand that was assigned before and $\mathbb{R}_{rb}$ specifically for reservations that have been assigned before.

In mathematical form this can be written as:

$$\sum_{\mathcal{T}_i \epsilon \mathbb{T}_{r=k}} \sum_{j \epsilon \mathbb{V}_{\mathcal{T}=i}} \epsilon_{i,j} = 1 \quad \forall r \in \mathbb{R}_{db} \tag{3.5}$$

A slightly more specific version of this constraint can be formulated to only apply this constraint to reservations.

$$\sum_{\mathcal{T}_i \epsilon \mathbb{T}_{r=k}} \sum_{j \epsilon \mathbb{V}_{\mathcal{T}=i}} \epsilon_{i,j} = 1 \quad \forall r \in \mathbb{R}_{rb} \tag{3.6}$$

To ensure these constraints can be satisfied, it is required to add the edge of the previous assignment to the RTV graph for all vehicles. A reason the edge is not yet in the RTV graph could be that the computation limit for a certain vehicle was reached before the specific trip was calculated. The trip to compute can be slightly different compared to the previous assignment as some requests can be picked-up already. The previous assigned trip should always the feasible as the vehicle has taken that route as the trajectory to follow. To guarantee a feasible solution, the previous solution is used is as the initialization of the ILP in case the pledged service constraint is applied. This guarantees a viable solution to be found as the previous assignment will be feasible.

The subsets of the previous assignment are not separately added to the RTV graph. This means that there is a reduced chance of re-assigning these requests to different vehicles since the other requests can not be assigned as a subset. However, as subsets are smaller trips that are computed earlier in the process, most likely they have been added already.

3.3.5. Vehicle options

Vehicles have to finish an edge they have started on. For route planning, this means that the vehicle will start at the end node of the current edge. The remaining time for the current edge is added to the travel cost for moving towards the first node. In case an arc will not be finished in the next time step, then it is not necessary to assign new requests as the vehicle can not re-plan its route before the next time step anyway.

This is a heuristic where some slightly less optimal assignments are made in return for less computational time. The worse assignment can be caused by assigning a request to a different vehicle as the rejection penalty can be higher than the total delay caused to this vehicle. A straightforward method to maintain the benefit and lose the downside is to only calculate the last assigned trip for this vehicle. This does still allow for re-assignment of requests if it is more beneficial system-wise but does not negatively impact the solution space too much. A small downside to the implementation is that subsets of the trip are not calculated. Therefore the flexibility is reduced as it became harder to re-assign requests from this trip to other vehicles.

Another aspect of a vehicle update is the waiting at node function. Based on the inclusion of reservations, it might be relevant to wait a limited amount of time at a node since you know a request is going to emerge there at a certain time. The ability to wait is included in the calculation of the best route in a trip. In case waiting is employed in a route, an additional node is added in the route, and an additional time and node pair is added.

The node is the same as the previous node (you can only wait at nodes), but the arrival time is later. The reason not to increase the time of arrival at that node is that it would make the system think the vehicle would be very slow to reach that node. This would imply that the vehicle is not available for re-assigned in the meantime, while in reality the vehicle is just waiting and can be re-assigned at any time if that would be better for the system.

4

Experimental setup

In this section, the experimental setup is discussed in detail. This includes the requests and road network used, the vehicle initialization and the warm-up procedure. The KPIs to be evaluated are introduced as well. Furthermore, the default parameters for the experiments are discussed and a second data set to experiment with the sensitivity is elaborated on. Last, the hardware used is listed for reproducibility purposes.

4.1. Network layout

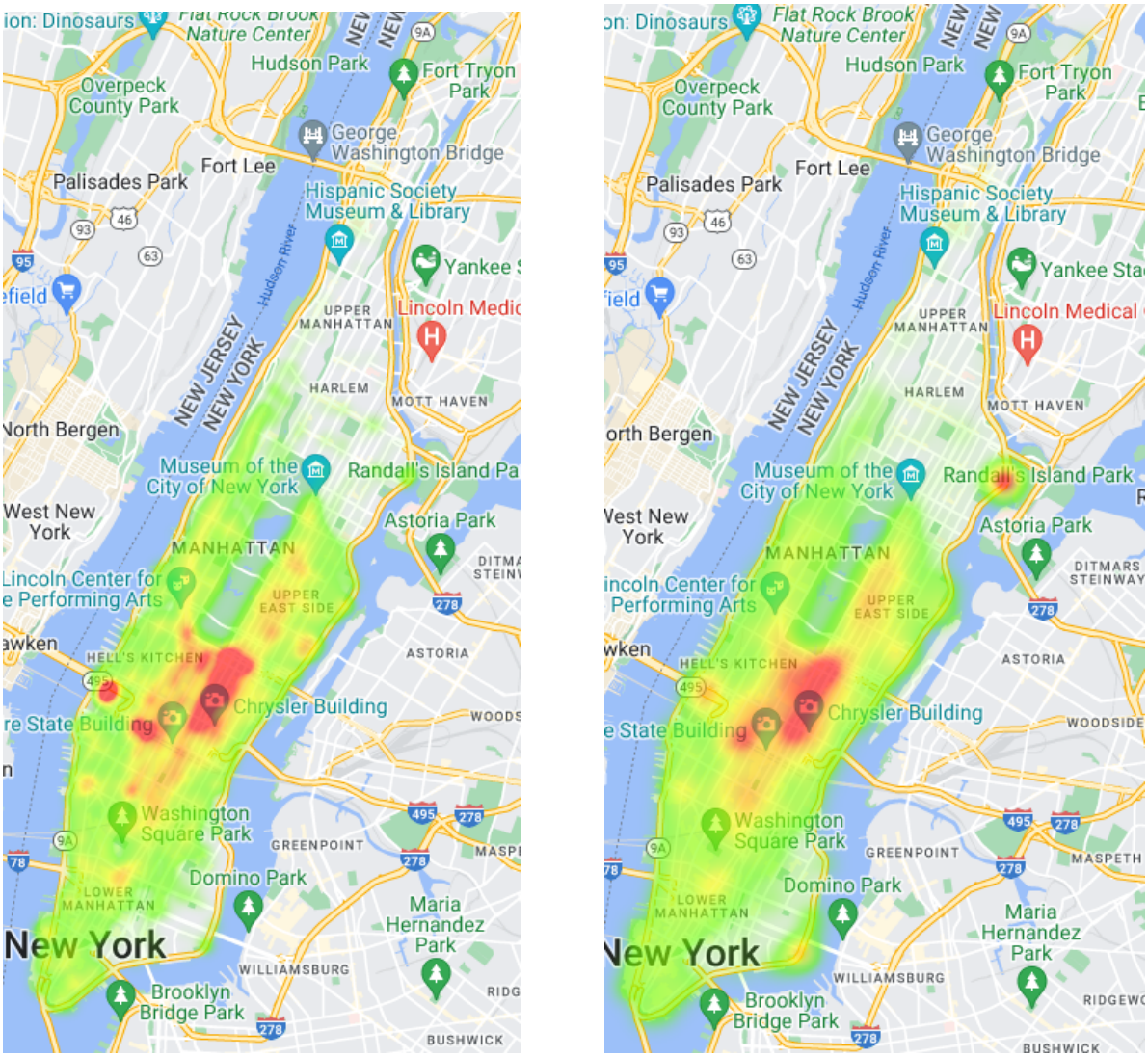
The network for the experiments is from Manhattan, New York City. The graph describing the network consists of 4091 nodes and 9454 directional edges. The cost for each edge in the graph is given in seconds and based on the average real-world travel time for that specific edge. The shortest route and distance between each pair of points in the graph is calculated beforehand using the Dijkstra algorithm and stored in a lookup table. The cost to go from a node to itself is always set to 0. The distance look-up table is a matrix of 4091 by 4091 with the cost to go from each node everywhere. The matrix has to be filled, as the edges are directional. To compute routes, the best next node to go to is determined for each node pair in the network. This means that if you want to go from A to B, you look at row A, column B. It will state what node you need to go to next, say F. You then iterate to row A, column F. You continue this process until you reach node B. With this method, you have to store less data, as only nodes are stored instead of complete routes.

An option to vary the speed of vehicles is implemented. This is done by creating the option that vehicles travel more/less distance per second of simulation time.

4.2. Request creation

Requests are taken from actual data for taxi demand in Manhattan from 12 AM till 1 PM, on the 15th of January 2013. The set of requests consists of 10784 requests, of which 226 have the same start and end point and thus are neglected. This results in 10548 valid requests to be considered. Each request has starting and end coordinates. These have been matched to the closest node in the road network. The requests consist of pick-up and drop-off nodes, request time and group size. For the numerical simulations run, all group sizes are set to 1. The request times are given in minutes. The requests for each minute are split in half and assigned to start times for each half minute. This is done to have the requests enter the system spread more evenly over the time steps in the model.

In figure 4.1, the locations where the requests originate and want to go to are shown. Uncoloured areas have a very low origin/destination density. The images do not show from where to where demand flows.



(a) Spatial distribution of the origin nodes for requests.

(b) Spatial distribution of the destination nodes for requests.

Figure 4.1: Request origin/destination distribution.

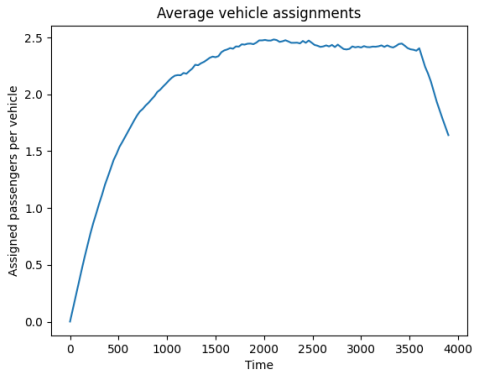
4.3. Vehicle initialization

The vehicles in the system have to be assigned a starting location before the simulation starts. The decision is made to have a uniform distribution over the network to start with. This is implemented as every Xth node has a vehicle with that start location. X can be determined by rounding the number of nodes in the network (4091) by the number of vehicles. In the case of 1000 vehicles, every 4.091th node will have a vehicle. Since vehicles are initialized at whole nodes, a rounding method is used. This means vehicles will be initialized at nodes 1, 5, 9, etc in this example.

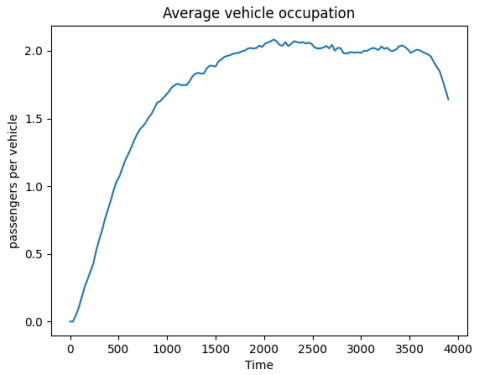
Results have been generated for 750 and 1000 vehicles, in some relevant cases, the scenarios have also been simulated with 1250 vehicles. In the result section, the results for 750 will be highlighted while the other results are placed in the appendix. The reason to discuss the results with 750 vehicles in more detail is that the differences are slightly larger and trends are slightly clearer.

4.4. Test scenario

The standard test scenario used consists of a warm-up phase and a “real” part. The goal of the warm-up phase is to approximate what the initial conditions would be with service in the previous hour. This includes the location and occupation (with the accompanying constraints on waiting/detour time) for each vehicle.



(a) Assignment to vehicles without a warm-up phase.



(b) Occupation of vehicles without a warm-up phase.

Figure 4.2: Average vehicle assignment and occupation without a warm-up phase.

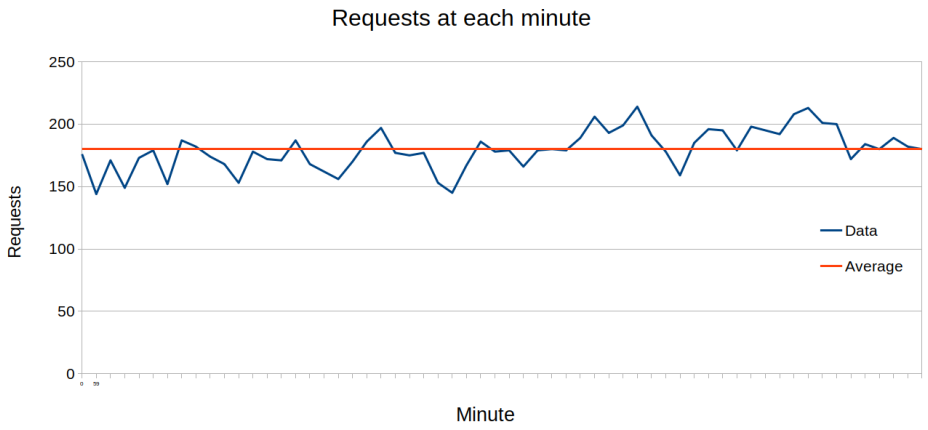


Figure 4.3: Request distribution over time.

4.4.1. Warm-up phase

In the warm-up phase, 5250 requests are added to the system over half an hour. This is approximately half of the requests that will enter the system in the hour of “real” simulation. This means that the workload will be equivalent and thus that vehicle occupation and positions are representative. The set of 5250 requests is randomly drawn without redraw from all possible requests and equally spaced over the available time for warm-up. Spacing uniformly over time is approximately similar to the request emergence rate during the real experiment, as is shown in figure 4.3. The implicit assumption is made that the number of requests and the origin/destination ratios are similar to the current data set.

The need for a warm-up phase is shown in figure 4.2. Time is needed before steady-state behaviour is reached. A steady state is desirable as it shows the impact on prolonged performance.

To ensure that the conditions at the end of the warm-up are as similar as possible to the “real” experiment, the same percentage of requests are considered reservations. This is necessary since reservations do increase the average vehicle occupation. Furthermore, the same parameters for heuristics are used.

After the warm-up time, the KPIs are reset to monitor the behaviour of the system from the “real” experiment. The distance travelled by the vehicles is reset to 0. The state data from the solve time, solve state, RTV building time, rebalance time, vehicle occupation and assignment and total computational time for each iteration are reset. The warm-up requests are still available for pick-up and assignment as long as their maximum waiting time is not violated. This is because removing them would cause a decrease in assignments and vehicle occupation.

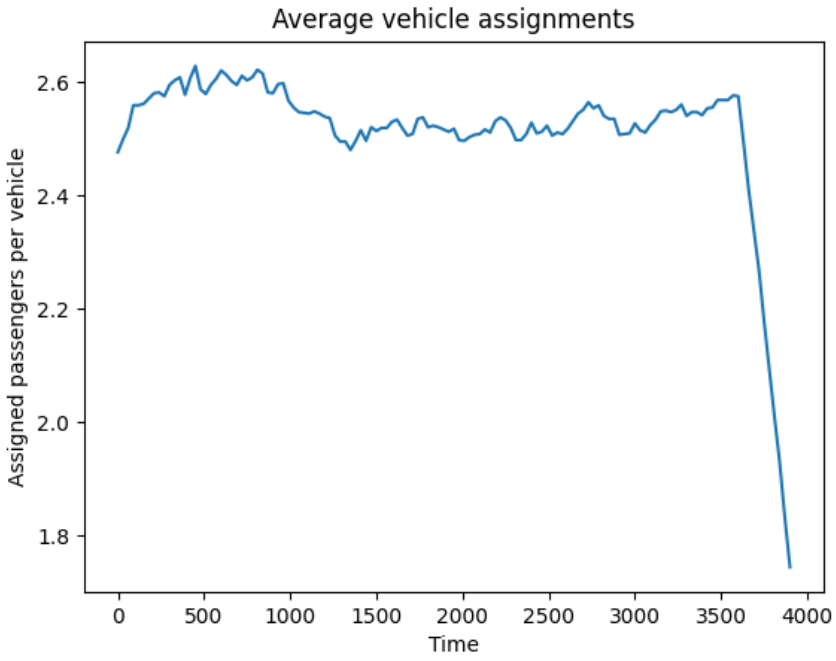


Figure 4.4: Assignment to vehicles with a warm-up phase.

4.4.2. Test phase

In the real test phase, the performance of the system is monitored and stored. The actual test consists of 10548 requests.

The inclusion of a warm-up phase as described above leads to average vehicle assignment as in figure 4.4. As can be observed, the starting level of vehicle assignments is much more in line with the rest of the experiment. Therefore, the experiment is considered to have reached a steady state before the KPIs are recorded. The dip in assignments at the end of the simulation is because there are no reservations and requests included from the next hour of operation. The simulation ends when there are no more requests, which happens once the last request is served or retracted because the waiting time constraint has been violated.

4.5. Reservations

Requests are called reservations if they are revealed to the system before they emerge. This means that the system is aware that a request with those specifications will emerge at a certain time from now on. The system can thus already assign the request. For a request to emerge means that the first time it can be picked up has happened. For “normal” requests, this is the same time as when that request is revealed to the system.

Reservations can be placed any time in advance. In the experiments, all requests will be revealed to the system X minutes before they emerge, with X a variable to be examined. The motivation to show reservations only a set time in advance is the growth in computation time as described in section 3.3.1. The focus is on reservations that are revealed shortly in advance.

Two different cases of requests as reservations will be used during testing. In the case of “random reservations”, a sample of all requests is selected and the reveal time for these requests is reduced with the time in advance the requests are to be revealed to the system. In the case of “rejections as reservations”, the rejected (unserved) requests from the base case (without reservations) can be selected to become reservations for a new experiment. This is a scenario where more difficult requests make reservations to improve their service.

4.6. Evaluation parameters

Multiple key performance indicators (KPIs) will be used to evaluate the performance of the system. The different KPIs used are listed below. Not all KPIs are equally relevant to evaluate different aspects of the model. The relevant KPIs will be discussed for each result.

- Service rate
- Service rate for reservations
- Waiting time, detour time and total delay time
- Distance ratio
- RTV build time
- Computation time
- Number of RTV edges

Service rate is a number that indicates how many of the requests are serviced by the system. This means requests that are picked up, transported and dropped off within the time constraints. The service rate is given as a fraction of all requests placed, both normal requests and reservations. The service rate is also determined for reservations. This provides insight into the differences between normal requests and reservations.

The average waiting and detour time for each served request is monitored. The sum of these two values provides the average total delay experienced by people in the system.

The distance ratio is a parameter to show the efficiency of the vehicle's distance travelled. The distance ratio is the sum of all effective distance travelled divided by all vehicle distance passed. Effective distance is the minimum distance between the origin and destination for a request, this is the route an unshared vehicle would take. By using this ratio, the KPI tells something about how efficient the sharing that the system provides is. It is influenced by how many people are on board and how much of a detour they experience. It also includes rebalancing distance and automatically adjusts for the number of requests served.

RTV build time is the time spent on creating the RTV graph. This part should be a heavy part of the simulation. It is only the time for the "real" requests, the warm-up phase is not considered here.

Computation time is the total time used for the computational steps. This means creating the RV and RTV graphs and solving the ILP problem.

The number of RTV edges indicates how many trips can be assigned to vehicles. This gives an impression of the solution space for the assignment algorithm to use for the best combinations of requests and vehicles.

4.6.1. Default parameters

The experiments will test different parameters for the system. In order to keep results comparable, the other parameters will be set as the default parameters described in this section unless mentioned otherwise.

Heuristics are by default not applied unless stated that they do. The pledged service constraint on requests that have been assigned before is also not used in the default settings.

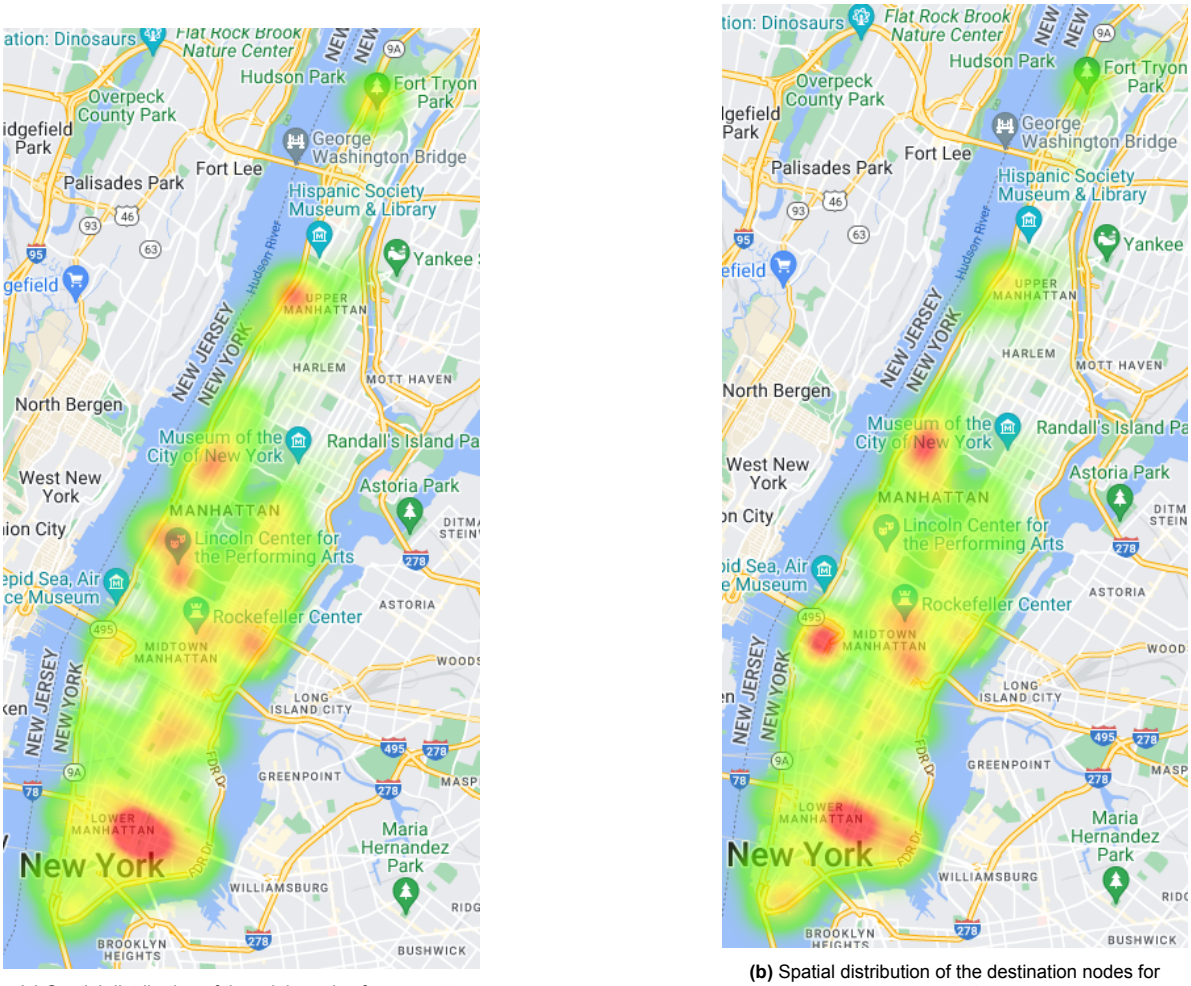
Parameter	Default value
Maximum waiting time	300 [s]
Maximum detour time	300 [s]
Vehicle capacity	4
Rejection penalty	10
Reservation penalty factor	2
RTV build time	75 [s]
ILP solve time	15 [s]
Maximum trip size	4

A rejection penalty of 10 means that the rejection penalty is 10 times larger than the maximum waiting time. A reservations penalty factor of 2 means that the rejection penalty for reservations is twice as high as for normal requests.

RTV build time is implemented as the total time available to generate the RTV graph of all vehicles. The time is divided equally among the vehicles. As not all vehicles reach the time limit, computation is significantly faster in most cases.

4.7. Sensitivity analysis

The sensitivity towards the underlying request data will be examined. A second set of requests from Manhattan is used for this purpose. The control set is taken from the morning peak. The morning request set consists of 28030 requests. To be comparable with the noon data set, 10548 unique samples will be drawn from this morning hour. The distribution of origin/destination nodes is shown in figure 4.5. Compared with the original distribution from figure 4.1, significant differences are observed, therefore this data set can be used for validation of the sensitivity to the demand data.



(a) Spatial distribution of the origin nodes for requests.

(b) Spatial distribution of the destination nodes for requests.

Figure 4.5: Request origin/destination distribution used in the sensitivity test.

4.8. Hardware used for simulations

For the results to be as comparable as possible, all simulations are performed on the same hardware with minimal background programs running. For reproducibility, the relevant computer specifications are provided below.

- CPU: Intel i7-4770 quad core
- RAM: 8GB

5

Results

In this section, the results from different experiments are discussed. First, a scenario without benefits for reservations is tested. Second, a test on available computational time is performed. Third, the effects of benefits for reservations are evaluated. This is done for the pledged service constraint and increased rejection penalties. Fourth, different heuristics are evaluated to speed up the process. Fifth, the heuristics and pledged service constraint are applied to test performance with more available information. Sixth, the sensitivity with respect to the request data is evaluated. Last, the performance of the model is tested during morning rush hour to evaluate performance at a large scale.

5.1. Reservations without benefits

Initially, the system will be tested without benefits for reservations nor heuristics in order to see the potential system-wide effects. Compared with the default settings, the reservation penalty factor is set to 1. It can be expected that by having additional information on future demand without cost to that information (optimization-wise, computationally you still have to pay the price), a better solution will be found. Both more reservations and reservations that are placed longer in advance will lead to better results.

The results for this simulation are shown in table 5.1. The rows indicate the percentage of reservations. A number as a percentage indicates that random requests from the set are taken as reservations. Rejections in this column mean that all requests that are not served in the scenario without reservations will be considered as reservations for this experiment. Each column indicates the time in advance the reservations are revealed to the system. A time of 0 means that there are no reservations revealed earlier to the system.

Each cell contains the outcome for a simulation run with the settings as in the row/column. There are five numbers in each cell. The top number shows the overall service rate. The second number is the service rate for the reservations in the system. The third numbers indicate the average total delay served passengers experience. The fourth number is the ratio of effective distance travelled. This is the the effective passenger distance travelled divided by total distance travelled from all cars. The effective distance a passenger travels is the minimum distance that trip would be if the trip would be unshared. The fifth number is the simulation time needed for that particular run in seconds.

		Time revealed in advance [minutes]	
		5	10
Random reservations	0%	0.692 - 274 1.06 82	0.692 - 274 1.06 82
	10%	0.71 0.842 270 1.09 231	0.726 0.841 267 1.11 901
Service rate (SR) Reservation service rate Average total delay [s] Distance ratio Simulation time	20%	0.737 0.849 267 1.09 430	0.749 0.866 263 1.14 3064
	30%	0.746 0.844 265 1.13 738	0.76 0.866 262 1.15 6034
Rejections as reservations	30.8%	0.749 0.728 266 1.13 705	0.77 0.777 273 1.15 5714

Table 5.1: Performance with reservations for 750 vehicles in case of no benefits for reservations and no heuristics applied.

Service rate improves with more information, both with more reservations and reservations that are revealed to the system longer in advance. The maximum increase in service rate with this setup is from 69.2% up to 77% in case the rejections are reused as reservations, this is an increase of 11.3%. Based on the trends in the table, the maximum benefits could be larger in case of more reservations are placed or reservations are revealed longer in advance to the model. However, one would expect diminishing returns with more information, as some vehicles are already assigned optimally at least some iterations with the current settings. The best overall service rate is reached in case the rejections from the scenario without bookings are used as reservations in a scenario. In the table above, this is also the scenario with the highest percentage of reservations (slightly). However, the difference in reservation rates is not enough to explain all of the differences in service rates. In appendix B.1, where the results from the same scenario for 1000 vehicles are shown, the rejections as reservations scenario performs relatively better than random reservations as well. The higher service rate achieved is paid for by increased waiting and detour time compared to random reservations. This is because the requests that are reserved are harder to service. This shows from the fact that they would be rejected without reservations. However, if you know in advance you can consider them a bit better and the value of this information seems to be more relevant than in the case of random reservations.

The service rate for reservations is higher than for on-demand requests in the case of random requests. This can be explained by the fact that the assignment to vehicles takes routing towards these reservations into account and thus reduces the cost in the next iteration compared to requests that are not yet considered. This effect is less obvious in the rejections as reservations scenario. The requests that are not served without reservations are those requests that are not a good match in time and origin/destination compared to the other requests/vehicle occupation at that moment.

As opposed to a trade-off between service rate and total delay time, a general decrease in total delay can be observed with increasing service rates. This is less contradictory than it seems at first glance,

as a decrease in total delay time is also strongly correlated with more reservations or reservations that are revealed to the system longer in advance. Since total delay time is in the objective function, better optimization, allowed by having more information, will also decrease the total delay time. Another aspect to consider in evaluating the service rate and the total delay time is the rejection penalty, as this value determines the weight each aspect has in the objective function. An in-depth review of the rejection penalty is done in section 5.3.2.

An explanation for the decreased delay time and higher service rate with more reservations can be found in the increased distance ratio. This indicates vehicles travel more relevant distance per distance travelled. Since the service rate is not close to 1, it can be assumed that (as good as) all vehicles are continuously busy, either picking-up/dropping-off passengers or rebalancing. If vehicles travel more relevant distance during the same time they can serve more passengers.

With more reservations or reservations revealed longer in advance, vehicles will spend less time rebalancing. With 10% random reservations, revealed 10 minutes in advance, with 750 vehicles, less than 2% of vehicles are assigned to rebalance at any iteration. Without reservations, up to 5% of vehicles are rebalancing. The optimal assignment of 1 reservation for a vehicle that otherwise would rebalance becomes more and more possible. This means that the vehicle is in effect still rebalancing, but with the certainty of there being demand in the future where the vehicle is going.

Computation time clearly increases with more information. Scaling towards the right side is more expensive than going downwards in the table. This is because requests that are available longer in advance can be combined with more vehicles, and more importantly become part of larger trips. Based on the results from table 3.1a, these longer trips are significantly more expensive to calculate as there are more possible permutations to be considered on pick-up and drop-off actions. The required computation time grows with orders of magnitude, highlighting the problem. The large increase in computation time moving right in the table is the reason results are for now only considered up to 10 minutes in advance. Once relevant heuristics to limit the computational time have been established, longer in-advance reveal times can be studied effectively. Using rejections as reservations does decrease the computational time slightly as these requests are harder to service.

The trends in the results described do hold for larger vehicle fleets as well. The results are shown in similar tables in appendix B.1 and B.2 for 1000 and 1250 vehicles respectively. As can be expected, the inclusion of reservations shows slightly diminishing returns with more vehicles, as there are fewer requests not serviced yet that can be serviced additionally.

5.2. Available computational time

The main time-consuming step in the simulation is creating the RTV graph. The second largest time consumer is solving the ILP assignment problem with the Gurobi solver. However, this step takes a lot less time compared to creating the RTV graph.

The time available for creating the RTV graph is determined beforehand and split equally over the available vehicles. Since most vehicles don't reach this time limit, the simulation can still run in real-time (meaning iterations take less than the time step to compute) if the total time available is set larger than the time step.

In table 5.2 the simulation results with different combinations of computational time constraints are shown for a scenario with 750 vehicles where 10% random reservations are placed 10 minutes in advance. Furthermore, the reservation penalty factor is set at 2.0, these effects will be thoroughly discussed in section 5.3.2.

		ILP time [s]		
		5	15	25
RTV build time	30 s 0.04s per vehicle	0.726	0.729	0.729
Service rate		0.991	0.992	0.991
Reservation service rate		2140	2092	2104
Computational time	75 s 0.1s per vehicle	0.731	0.727	0.729
		0.986	0.986	0.991
		3243	3042	3042
	150 s 0.2s per vehicle	0.732	0.73	0.733
		0.988	0.983	0.991
		4087	4209	4043

Table 5.2: Performance with different limits on computational time for 750 vehicles. The scenario is with 10% random reservations, revealed 10 minutes in advance.

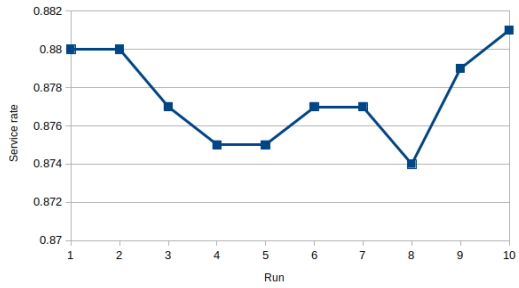
Based on the result in table 5.2, more available time for solving the ILP does have a very small effect on the results of the simulation. This is because most often, the optimal solution for the assignment is found within 5 seconds for the tested scenario. The general trend with increased computational time per vehicle is a very slight improvement in performance in return for significant increases in computational time. The large increase in computational time is explained by the order in which trips are created. First, small trips are created and once a trip size is complete, the next trip size is calculated. This process continues until a timeout is reached. With more time available, all this extra time is used to calculate larger trip sizes. Based on the marginal increase in performance, it appears that these larger trip sizes are barely assigned to vehicles.

In a scenario with more requests, where more vehicles can be shared with extra passengers, larger trip sizes can show more benefits. Additional time for finding better solutions to the ILP problem can also improve the solution in these scenarios. Experiments where more reservations are made or if reservations are revealed longer in advance benefit from additional computation time as well.

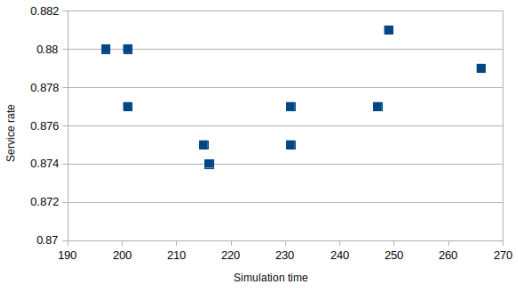
The results for the same scenarios, but with 1000 vehicles, are shown in appendix B.3. The results are consistent.

When looking at very small differences in the results, it is important to keep in mind that the results are impacted by computational performance. The system created is deterministic in computing the cost of trips. Based on small variances in computational time, a small difference in which trips are computed can happen over different runs. The ILP solver is deterministic as well, but only for the same input. Because of the multi-threaded implementation, small differences in variable order can happen. For example, a trip can be number 6 or 7 in different runs based on priority in threads. Based on these small differences, the ILP solver will take slightly different paths in optimizing the solution. In case two solutions have the same objective value, the optimum solution can be different over different runs. An easy example how this can happen is if only 1 reservation needs to be assigned to 5 vehicles. If the reservation is revealed far enough in advance, all vehicles can serve this request without any delay, thus all having a cost of 0. The optimum solution will thus have an objective value of 0 but can assign the request to different vehicles based on the initialization of the ILP. Small differences lead to different vehicle routing and thus to different trip availability in consecutive time steps. A second explanation can be in the implementation of optimality tolerance. This phenomenon is not experimented with and the default value from the ILP solver is used.

The results for 10 runs with the exact same settings and requests are visualized in fig 5.1. The service rate is plotted over the different runs in figure 5.1a and versus the total simulation time in figure 5.1b.



(a) Service rate over 10 runs with the same settings on the same request data.



(b) Scatterplot of service rate vs simulation time for 10 runs on the same request data.

Figure 5.1: Results for 10 consecutive runs with the same settings.

Based on the results from figure 5.1, it becomes clear that swings in service rate can occur over different runs, in an unpredictable pattern. Based on figure (b), the runtime for the simulation can fluctuate up to 5% and the service rate 0.6% under these settings.

Because of these fluctuations, all results shown in this chapter (except for these results) are the average of 3 runs with the same settings. This solution mitigates, but not excludes, the variance in solution quality.

5.3. Benefits for reservations

Benefits to reservations are assigned in two different ways. First, the effects of the pledged service constraint are evaluated. Second, the impact of the rejection penalty is discussed. Last, the two methods to assign benefits to reservations are compared against each other in different scenarios.

5.3.1. Pledged service constraint

As observed from the raw data with respect to the rejection of reservations, as good as all reservations have been assigned to a car at one moment or another. As mentioned in section 3.3.4, an additional constraint, the pledged service constraint, can be added to ensure that requests that have been assigned before should be assigned in each iteration afterwards. This constraint can be applied to reservations or to all requests. The results of a scenario with 20% random reservations revealed 10 minutes in advance, with the pledged service constraint active for either reservations or all requests, are shown in table 5.3. The results shown are for an overall rejection penalty of 10, with a reservation penalty factor of 1. Based on experiments, these parameter settings do not impact the solution significantly. The results for different rejection penalty parameters are shown in appendix B.4. Results in case rejections from the base case (without reservations) are used as reservations can be found in appendix B.5. The trends in results are the same for different rejection penalties, reservation penalty factors and the different reservation set.

The results shown are for 750 vehicles, the results for 1000 vehicles are in the appendix for completeness.

	Pledged service constraint			
20% random reservations Revealed 10 minutes in advance	Not active	Applied to reservations	Applied to all requests	No reservations
Service rate	0.749	0.731	0.708	0.692
Reservation service rate	0.866	1	1	-
Average total delay [s]	263	268	278	274

Table 5.3: Impact of pledged service constraint on the average service rate and the service rate for reservations. Experiment with 20% random reservation revealed 10 minutes in advance.

Based on the results in table 5.3, it can be seen that the service rate for reservations increases to 1 in both cases where the pledged service constraint is activated. This means that all reservations have been assigned to a vehicle at least once and then again each iteration afterwards. The additional

constraints on assignment do lead to a lower overall service rate and increased total delay. This is explained by the reduced optimization space available to the ILP solver.

5.3.2. Impact of the rejection penalty

The effect of different rejection penalties in the system is tested on a scenario with 750 vehicles, with 20% random reservations revealed 10 minutes in advance. Rejections as reservations will be discussed later in this section. Since a higher rejection penalty will guide the ILP assignment algorithm to assign more passengers, the optimization will consider total delay for passengers (waiting time plus detour time) less. Furthermore, in this simulation, the penalties for reservations are increased as a factor of the rejection penalty for normal requests. The rejection penalty in general is a multiplication of the maximum waiting time. The results for the simulation with 750 vehicles are shown in table 5.4

Rejection penalty = 5 750 vehicles	Reservation penalty factor						
	1	**1.2**	**1.5**	**1.8**	**2**	**2.5**	**5**
service rate	0.744	0.739	0.74	0.74	0.741	0.74	0.731
reservation service rate	0.872	0.931	0.971	0.986	0.996	0.998	1
average total delay [s]	254	258	258	259	259	259	260

(a) Rejection penalty 5

Rejection penalty = 10 750 vehicles	Reservation penalty factor						
	1	**1.2**	**1.5**	**1.8**	**2**	**2.5**	**5**
service rate	0.735	0.737	0.741	0.735	0.732	0.722	0.727
reservation service rate	0.86	0.928	0.953	0.977	0.90	0.998	1
average total delay [s]	267	269	267	267	265	267	270

(b) Rejection penalty 10

Table 5.4: Effect of different reservation penalty factors over two rejection penalties with random reservations.

Based on the results in table 5.4, a clear increase in service rate for reservations can be observed with increases in the reservation penalty factor compared to normal requests. This is as expected, since rejecting reservations is now more expensive compared to rejecting regular requests and/or increased delay for passengers assigned to/in the car. Based on the KPIs in the table, there seems to be a substitution effect between serving reservations and serving on-demand requests at increased reservation penalties. The differences are generally small in the scenario with random reservations. Since on average, requests are similar in how costly they are to service compared to reservations, substitution on which to service can be done for very limited costs.

The overall service rate is barely affected by the increase in reservation penalty, except for the largest reservation penalty. This is similar for the average total delay passengers experience.

Compared to the pledged service constraint being applied to reservations, an almost similar service rate for reservations can be accomplished with an increased penalty for reservations. This does lead to a slightly higher service rate for all requests overall and slightly shorter average total delays. However, there is no guarantee for reservations that they are serviced in the end.

In case the rejections from the scenario without bookings are used as reservations, the effect of tuning the rejection penalty and reservation penalty factor increases. This is because the reservations are actually harder to service compared to the normal requests. In this scenario, there are 30.8% reservations (compared to 20% in the scenario above), based on results from table 5.1, the overall service level is expected to be higher in this scenario for the same condition. The results are shown in table 5.5

Rejection penalty = 5 750 vehicles	Reservation penalty factor						
	1	**1.2**	**1.5**	**1.8**	**2**	**2.5**	**5**
service rate	0.779	0.768	0.762	0.756	0.752	0.746	0.747
reservation service rate	0.791	0.864	0.934	0.955	0.976	0.991	0.999
total delay time [s]	253	256	260	262	265	262	266
Distance ratio	1.17	1.16	1.16	1.15	1.14	1.13	1.12
RTV edges *1e6	7.16	7	7.23	7.37	7.31	7.36	7.6

(a) Rejection penalty 5

Rejection penalty = 10 750 vehicles	Reservation penalty factor						
	1	**1.2**	**1.5**	**1.8**	**2**	**2.5**	**5**
service rate	0.762	0.759	0.752	0.744	0.745	0.741	0.725
reservation service rate	0.756	0.857	0.913	0.951	0.959	0.981	0.997
total delay time [s]	270	271	274	272	275	276	278
Distance ratio	1.15	1.14	1.14	1.12	1.11	1.11	1.08
RTV edges *1e6	6.54	6.82	6.88	6.91	7.06	7.03	7.46

(b) Rejection penalty 10

Table 5.5: Effect of different reservation penalty factors over two rejection penalties with rejections as reservations.

Based on 5.5a, it becomes clear that the expected trade-off between servicing reservations or normal requests is present in this scenario. Furthermore, not all reservations are served with the highest set reservation penalty factor. This could be “solved” by increasing the reservation penalty factor further. Another trend is that the average total delay time does increase with a higher service rate for reservations. This is because the reservations are the requests that are more costly to service. This also leads to a decreasing distance ratio, indicating that more vehicle distance is travelled per passenger distance. The total number of feasible RTV edges does increase with a higher rejection reservation penalty factor. This is because, at a lower service rate, there are more requests that are not in vehicles and thus can become part of trips.

For the situation with a higher rejection penalty, the trends are similar. A remarkable result is that a higher rejection penalty in general does lead to a slightly lower service rate overall. Reservation service rates are closer between the different rejection penalties though. This is unexpected, as a higher rejection penalty does incentivize the system to assign more passengers at the cost of more delays. On the other hand, this also makes the system a bit more greedy as passengers are assigned to vehicles with worse matches (just to not reject). Rejecting requests that were assigned before does not happen significantly more often compared to the scenario with a rejection penalty of 5.

The assignment to worse matches at a higher rejection penalty does in later iterations lead to fewer options for additional passengers since the vehicles will be less flexible for assignment (worse matches mean more total delay and thus being closer to waiting and detour time constraints). This reduction in flexibility can be observed as an increase in average total delay and also in a reduction in the amount of RTV edges found over the course of the simulation. This last parameter provides some insight into the optimization space available to find the best solution. Furthermore, the distance ratio is decreased at a higher rejection penalty. Since served passenger distance is divided by the total distance, the number of served passengers is already included. A decrease in this parameter indicates that the system spends more time transporting fewer passengers or taking longer detours. The used time to construct the RV and RTV graphs is very comparable between the settings. This indicates that approximately the same amount of trips are calculated, but fewer trips are returned as feasible options.

5.3.3. Comparison between benefits for reservations

The two methods to assign benefits to reservations are compared in this section. The average service rate is plotted in case of random reservations revealed 5 minutes in advance. The results are plotted in case of “no benefits” in green, “pledged service” in blue and “increased rejection penalty” in orange. The pledged service constraint is only applied to reservations. The dotted lines represent the service rates for reservations in the corresponding colour. The results are plotted in Figure 5.2. The experiments

are done with 1000 vehicles.

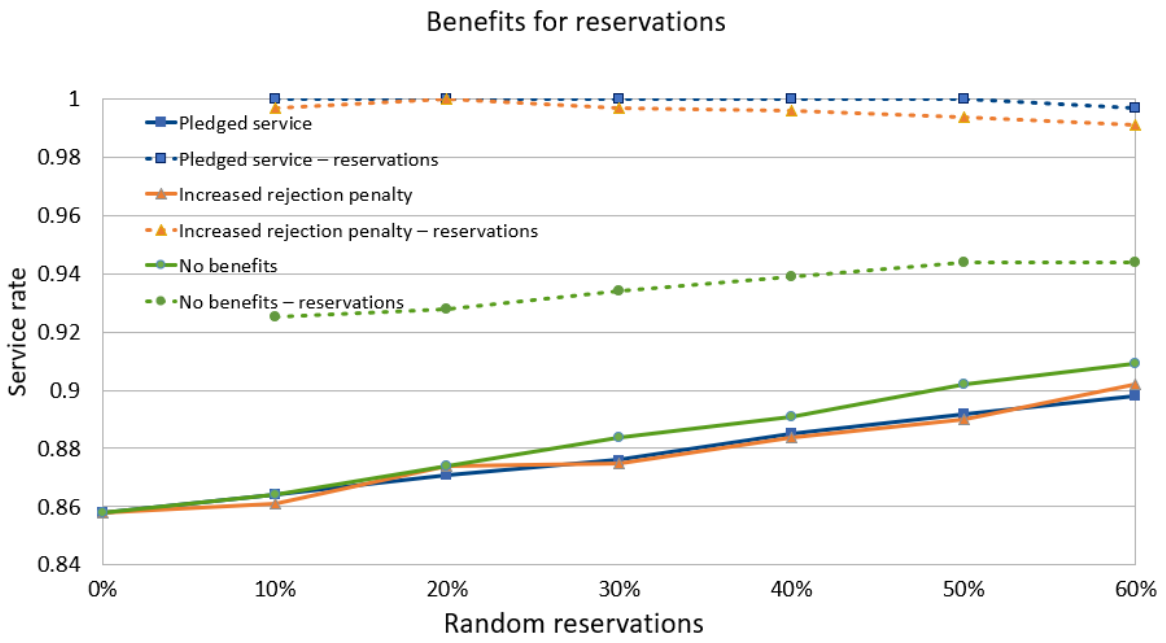


Figure 5.2: A comparison between the methods to assign benefits for reservations in case of random reservations revealed 5 minutes in advance. The results are for a scenario with 1000 vehicles.

Based on the results, the service rate is slightly lower on average in case benefits are awarded to reservations. The service rate for reservations (dotted lines) is close to 1 with the increased rejection penalty. Applying the pledged service constraint has the benefit that passengers know they will be served as soon as their reservation is revealed to the system. The difference in average service rate between the pledged service constraint and an increased rejection penalty is marginal.

In case the pledged service constraint is applied in further results, this is in combination with a small increase in the rejection penalty. In this way, reservations have a higher likelihood to be assigned initially in scenarios with high numbers of reservations.

5.4. Heuristic parameters

The result of using the heuristics described in section 3.3.3 are discussed in this section. The selected scenario for the simulations is with 10% random reservations, revealed 10 minutes in advance. The rejection penalty is set at 10, with a reservation penalty factor of 2. The rejection penalty is set at 10 since this is the same as for the earlier experiments.

The experiments have been run with 750, 100, and 1250 vehicles. This is to examine the results over different stress levels for the system. Trends over the different vehicle levels are similar, therefore only the results for 750 vehicles are discussed here. The results for 1000 and 1250 vehicles are shown in tables in appendix B.6 till B.12. For reference, the scenario is also run without heuristics. The results as shown in table 5.6. The RTV build time is reported as the heuristics aim to reduce RTV construction time.

Vehicles	**750**
service rate	0.719
reservation service rate	0.987
RTV build time	747

Table 5.6: Reference results: KPIs for 750 vehicles without heuristics used. The scenario is with 10% random reservations revealed 10 minutes in advance.

Maximum vehicles per request

The first experiment to run is based on limiting the number of vehicles a request can be assigned to. This is done for all requests, or just for reservations. Based on the results in table 5.7, an increase in performance is observed with more available vehicles. This comes at the cost of increased computational time used. A second observation is that the increase in service rate flattens out with more vehicles available. This happens since a vehicle with a mediocre match does not often lead to the cheapest larger trip sizes or best assignments.

The results in case the heuristic is only applied for reservations show similarity with the case it is applied to all requests. The similarity of results can be explained by the idea that the increase in service rate is caused by reservations. The reasons for this are both more options and longer trips. By limiting the reservations, the benefits are maintained while the costs are kept in check better quite well as unlikely options are not calculated.

Overall, gains in computational time can be achieved at very limited to no apparent reductions in service rate.

max vehicles per request	**1**	**10**	**20**	**40**	**60**
service rate	0.594	0.711	0.72	0.72	0.717
reservation service rate	0.684	0.983	0.993	0.987	0.996
RTV build time [s]	38	232	333	510	608

(a) Heuristic applied to all requests.

max vehicles per reservation	**1**	**10**	**20**	**40**	**60**
service rate	0.675	0.716	0.719	0.719	0.721
reservation service rate	0.777	0.991	0.988	0.991	0.991
RTV build time [s]	63	223	379	443	681

(b) Heuristic applied to reservations only.

Table 5.7: Limit on the number of vehicles available for each request to create trip sizes larger than 1. Results for 750 vehicles.

Prune the RTV graph

In table 5.8, the results of pruning per layer of the RTV graph are shown. The first number of “prune per layer” indicates the limit on the first layer of the RTV graph (trips of size 1), and the second number is the limit on trips of size 2 to make larger trips. To emphasize the difference with the previous heuristic, the limit on trips of size 1 means that there is a limit on requests per vehicle.

Based on the results, it becomes clear that limiting the first layer holds back the potential for longer trips and thus system performance. A large limit on the first layer does not limit the growth effectively. This is because only a few vehicles can at any point serve more than 80 different requests. The prerequisite to achieve this would be an almost empty vehicle at a high-demand origin location. A larger limit on the second RTV layer does allow for more options at larger trip sizes. Based on these results, there is a small benefit in those extra trips at the cost of some more computational time. The overall gains in computational time are small for the less restricted scenarios.

prune per layer	**20, 20**	**20, 40**	**20, 80**	**40, 40**	**40,80**	**40, 120**	**80, 80**	**80, 120**	**120, 120**
service rate	0.678	0.675	0.682	0.707	0.709	0.71	0.711	0.716	0.716
reservation SR	0.946	0.923	0.947	0.952	0.952	0.954	0.995	0.991	0.994
RTV build time [s]	322	366	416	537	522	553	639	759	769

Table 5.8: Heuristic to limit growth by pruning per trip size. Pruning is done on trip sizes 1 and 2.

Restrict reservations to X best vehicles

In case reservations are assigned to a set of cars once they are revealed, there still is the option to make those reservations available to all vehicles some time in advance. The result of revealing reservations to all vehicles 5 minutes in advance, once emerged, or never are shown in table 5.9.

Compared to the results from table 5.7, where requests the best set of vehicles is determined each iteration, both service rate and computation times are lower for the same vehicle set size.

This is because requests are assigned a set of vehicles in the iteration they are revealed. This initial assignment takes only the passengers into account that are currently on board the vehicle. It can

therefore happen that two requests that can not be serviced together get assigned to the same vehicle. Furthermore, the service for reservations is restricted to only a set of vehicles and the reservation penalty factor is 2. The system will thus focus on servicing reservations. Since there is limited flexibility in servicing reservations, this will come at the price of serving normal requests. In case requests can be serviced by more vehicles, the system will be more flexible and thus be able to serve more normal requests.

The performance differs in whether or not the reservations should be exposed to all vehicles sometime in advance are not conclusive. The effect of revealing the demand to all vehicles has a small positive impact in the case that only 1 or 10 vehicles are available in the set. An explanation would be that reservations are in general served by vehicles from the subset regardless of the availability of different vehicles since it offers the best or only match.

With a larger vehicle set, the results vary. A possible explanation is that if the cost of assigning a request to different vehicles is the same for more vehicles than there are places in the set, a random subset from the vehicles with the best scores will be used. This can impact the results if it happens often, for example because the reservations are placed a long time in advance. This variance is implemented to prevent a skew towards the vehicles that are computed first (those with the lowest vehicle ID). If all requests would be only assigned to the lowest scores, they would be more inclined to be selecting the same set. Another explanation could be that most of the requests can not/barely be serviced by vehicles outside the set anyway if the set is sufficiently large.

Assigned to X best vehicles	**1**	**10**	**20**	**40**	**60**
service rate	0.685	0.701	0.706	0.717	0.718
reservation service rate	0.901	0.975	0.988	0.994	0.987
RTV build time [s]	31	81	198	329	553

(a) Available for all vehicles 5 minutes in advance.

Assigned to X best vehicles	**1**	**10**	**20**	**40**	**60**
service rate	0.694	0.698	0.704	0.721	0.717
reservation service rate	0.889	0.967	0.989	0.992	0.994
RTV build time [s]	31	102	188	335	523

(b) Available for all vehicles once emerged.

Assigned to X best vehicles	**1**	**10**	**20**	**40**	**60**
service rate	0.696	0.701	0.708	0.711	0.718
reservation service rate	0.913	0.97	0.99	0.992	0.994
RTV build time [s]	34	87	190	354	524

(c) Only available to selected vehicles.

Table 5.9: Restriction on the available vehicles to assign reservations to.

Maximum trip size

The last studied heuristic to limit the computational time works by limiting the size that trips can take. The results for this experiment are shown in table 5.10. From the results, it becomes clear that larger trip sizes are the reason for longer RTV calculation times. The service rate for all requests and for reservations is remarkably similar over the different settings. This can be attributed to the request set used. The set of requests is taken around noon, not the busiest moment for taxi transport in Manhattan. This modest amount of requests also translates into a modest number of trips at each moment that can be efficiently shared. Based on the results, it can be concluded that for the shareable trips in this scenario, a limit on a trip size of 2 or 3 would be sufficient to handle the demand. With trips of size 2 or 3, the computation time required to construct the RTV graphs is reduced enormously compared to the reference scenario in table B.6, while maintaining a similar level of service.

It should be noted that determining a heuristic based on the trip size is sensitive to the used request set and reservation scenario used. A set with more options to efficiently share trips will show benefits from larger trip sizes. Vehicle capacity is also an important factor in the effect of maximum trip size. Vehicles with a smaller capacity will benefit less from larger trip sizes for example.

max RTV layers	**2**	**3**	**4**
service rate	0.713	0.717	0.718
reservation service rate	0.996	0.99	0.993
RTV build time [s]	32	129	708

Table 5.10: Limit on trip sizes that can be made.

To summarize

The results from designed heuristics show that significant reductions in RTV build time can be reached at a limited loss in service rate.

Limiting the number of vehicles a request can be assigned to shows comparable results with the reference case, while the RTV build time is reduced +- 55%, see table 5.7.

Pruning the RTV size in the first and seconds layer can reach similar performance, but only at a small reduction in computational time required, see table 5.8.

Assigning reservations to a set of vehicles that can serve them and only reveal them to all vehicles at a later stage does show good results, but does require a longer time than if the set of vehicles is recomputed each iteration. This is because a larger set of vehicles is needed if the best vehicles are only determined beforehand. Similar performance as in the reference case can be achieved with +- 30% less RTV build time, see table 5.9.

Limiting the trip size (RTV layers) shows very good results at a great reduction in computational load, see table 5.10. The method could scale worse with larger request sets though.

Combined heuristics

Different heuristics could also be combined. Settings for heuristics are determined so that each individual setting does not yet deteriorate performance.

The maximum trip size is combined with the other effective heuristics to study the combined impact. The results are in table 5.11.

	Reference values without heuristics		**Maximum trip size 3** 10% random reservations				
	Random reservations:		Restrict to X best vehicles		Max vehicle per request	Max vehicle per reservation	
	0%	10%	40	60	20	20	40
Service rate	0.692	0.719	0.713	0.714	0.717	0.715	0.715
Reservation service rate	-	0.987	0.995	0.993	0.992	0.994	0.992
Computation time [s]	82	747	73	96	81	83	103

Table 5.11: Combination of a maximum trip size 3 with other heuristics.

The most impact can be had by combining a limit on the trip size with a maximum number of vehicles per request. However, the difference is small indicating that by far most computation requirements come from reservations. The number of on-demand requests that can be assigned to more than 20 vehicles and be part of larger trip sizes seems to be very small.

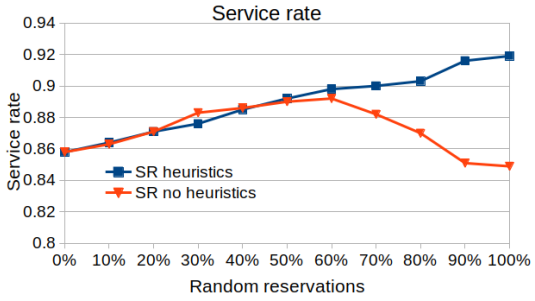
The heuristics are tuned on one scenario with 10% random reservations revealed 10 minutes in advance. Reservations receive the benefit of an increased rejection penalty. Results prove similar with the pledged service constraint applied as a benefit for the reservations.

The combination of a maximum trip size of 3 and a maximum number of vehicles per request of 20 will be used as the heuristic for further results.

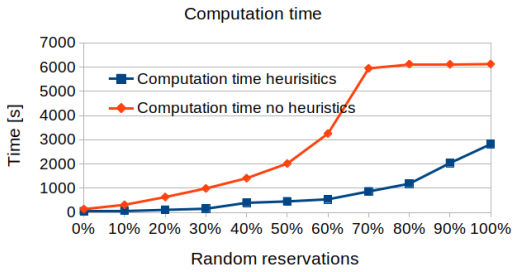
5.5. Applied heuristics

The reason to apply heuristics is to be able to process more reservations revealed longer in advance. The goal is to use this additional information for better overall service. In this section, the heuristic described above and the pledged service constraint are both applied. Results are shown for 1000 vehicles.

First, the added value of applying the heuristic is shown in Figure 5.3a. The blue line represents the average service rate with the heuristic and the orange line without a heuristic. All reservations are revealed to the system 5 minutes in advance.



(a) Average service level with and without the heuristic for different percentages of random reservations. Results are for 1000 vehicles with the pledged service constraint applied to reservations.



(b) Computational time required to run the simulations with and without the heuristic.

Figure 5.3: The reachable service quality with the use of the heuristic.

Initially, a steady increase in service rate is reached with more reservations. At more than 60% reservations, the average service rate starts to decrease without the heuristic. The reason for the drop in performance of the orange line is given in Figure 5.3b. The computation time for each scenario is shown with and without the heuristics. It is clear that for more reservations the experiment without heuristics reaches the computational time limits and deteriorates in performance as a consequence. The computational time limit will be elaborated on in the discussion.

The reason to use heuristics is to allow for more information to be used effectively by the model. This means that more reservations can be revealed to the system longer in advance. The heuristics will still reach a point where more information decreases system performance. However, this point is reached later and therefore a better solution becomes computationally feasible. A second reason to apply the heuristic is to simulate rush hour demand.

The heuristic will be used in the following results in this paper.

With the effect of applying the heuristic established, experiments with reservations revealed longer in advance can be performed. Both random reservations and the scenario with rejections as reservations are evaluated five to twenty minutes in advance. The resulting service rate, service rate for reservations and required computational time are listed in Table 5.12.

		Time revealed in advance [minutes]			
		5	10	15	20
Random reservations	0%	0.858 - 39	0.858 - 39	0.858 - 39	0.858 - 39
SR Reservation SR computational time	10%	0.864 1 61	0.868 1 107	0.874 1 240	0.874 1 921
	20%	0.871 1 103	0.883 1 382	0.885 1 2136	0.834 1 2367
	30%	0.874 1 151	0.888 1 2677	0.8 1 2730	0.804 0.993 2775
	40%	0.885 1 395	0.836 1 2786	0.806 0.985 2890	0.808 0.976 3101
	50%	0.892 1 639	0.819 0.995 2932	0.816 0.965 3036	0.805 0.959 3435
Rejections as reservations	14.2%	0.864 1 75	0.871 1 325	0.877 1 1209	0.862 1 3242

Table 5.12: Result from the incorporation of reservations in an on-demand ridesharing system for varying random reservations and times in advance the reservations are revealed. The heuristic and the pledged service constraint are applied. The results are for 1000 vehicles.

The results from Table 5.12 are plotted in Figure 5.4. Each row in the table is visualized as a colour in the figure.

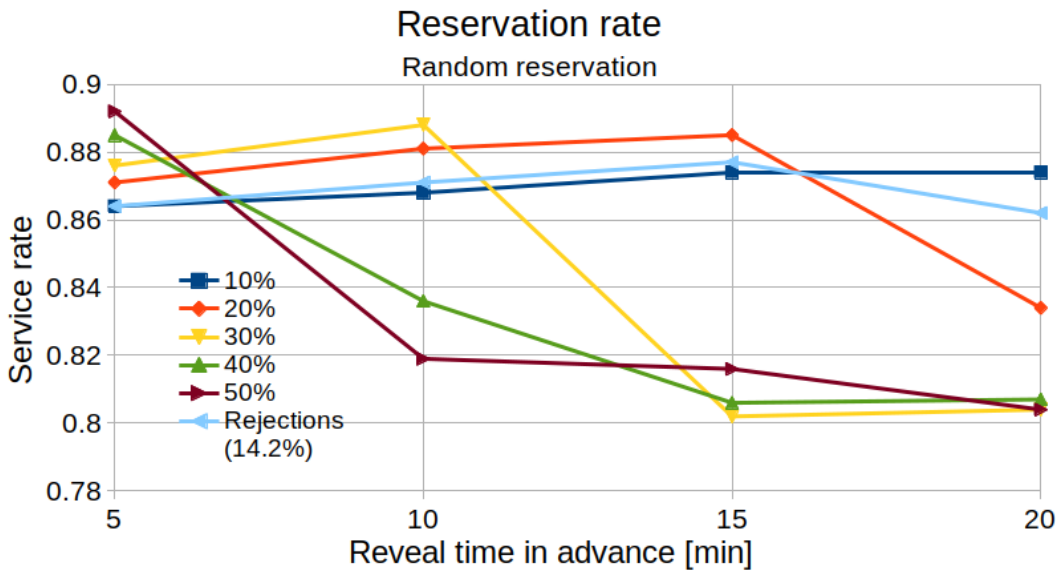


Figure 5.4: Service rate results from Table 5.12 visualized.

Based on these results it can be concluded that additional reservations improve performance more strongly than reservations that are revealed longer in advance, especially if there is a large fraction of all requests placed as reservations. Reservations that were placed are still being served significantly

better compared to on-demand requests because of the pledged service constraint.

The average service rate decreases partly because the heuristic parameters are tuned on just one scenario with limited reservations revealed shortly in advance. Since the tuning is not effective for all scenarios, the model can not both adhere to the pledged service constraint and pick effective options as these are not sufficiently created. Furthermore, the heuristic is too strict in limiting computations for reservations that are revealed longer in advance as the computation time limit is not always reached. The goal of heuristics is to compute the most likely trips in the available time. If not all time is used, the heuristics limit too many options that are potentially relevant. This problem was not present in the scenario where the parameters are tuned as most of the relevant options were calculated within the options allowed by the heuristic.

By tuning heuristic parameters to the selected scenario, a better result can probably be achieved with more reservations revealed longer in advance. A better tuned heuristic should be able to perform at least as well with more information as you can ignore the additional information. The added information can then be used only if it is beneficial. A downside of revealing more information longer in advance is that the pledged service constraint is applied to more requests in the set to be assigned and thus constrains the solution space more.

5.6. Sensitivity analysis

Experiments are run with the morning peak request set are run to assess the impact of the underlying request set. Since the heuristic parameters are optimized on reservations being revealed shortly in advance, these settings will be used for the comparison in Figure 5.5. All reservations are revealed 5 minutes in advance. The orange line indicates the service rate in the morning hour. The dotted lines indicate the service rate for the passengers that placed a reservation. Results are for a simulation with 1000 vehicles

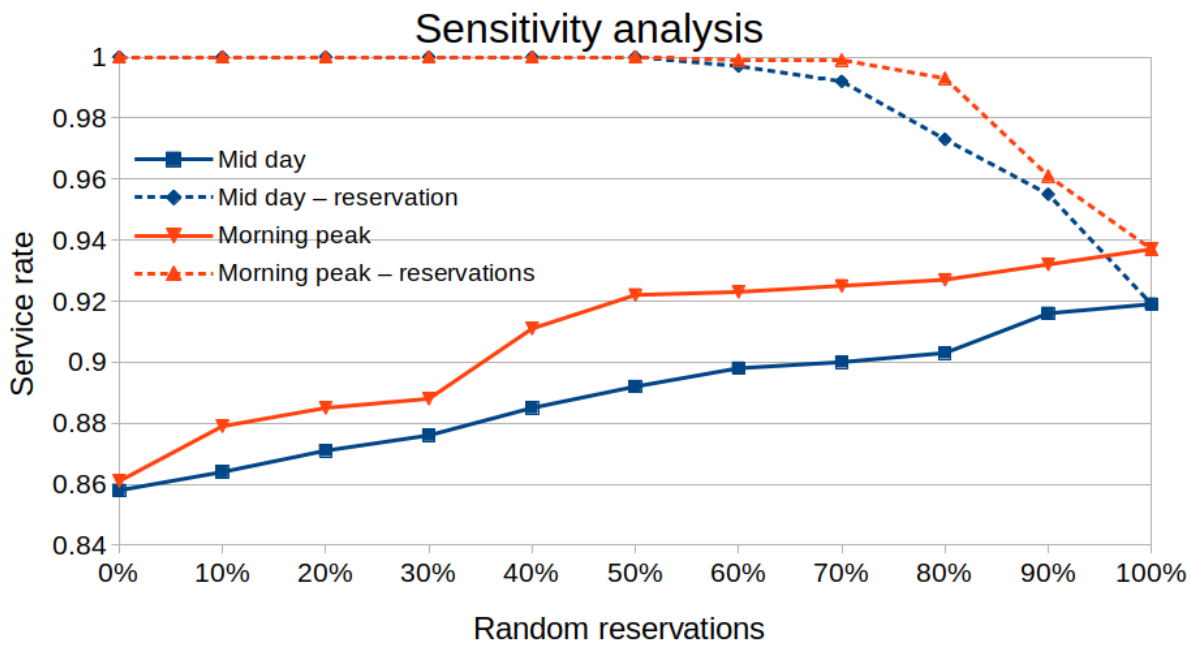


Figure 5.5: Simulations for sensitivity data set compared to the test set. Requests are all revealed 5 minutes in advance, experiment with 1000 vehicles.

The most important conclusion for the results is that the trend service rate is similar for the different request set used. The overall service rate is slightly higher, indicating that is possible to combine more requests into trips. This is because there is slightly more overlap in the origin/destination combinations in the set.

5.7. Rush hour performance

The request set from the morning peak will be used to test if the methodology works at a large scale. For this, all 28030 requests from the morning data set are used. The results will be run for random reservations ranging from 0-50%, with all reservations revealed 5 minutes in advance. As the request set consists of 28030/10548 = 2.66 times more requests, additional vehicles will be needed to serve the demand. As more requests allow for more sharing, increased performance can be expected with the same fraction of vehicles. The resulting service rates are shown for 2000 and 2500 vehicles in Figure 5.6. Extensive results are in Table 5.13.

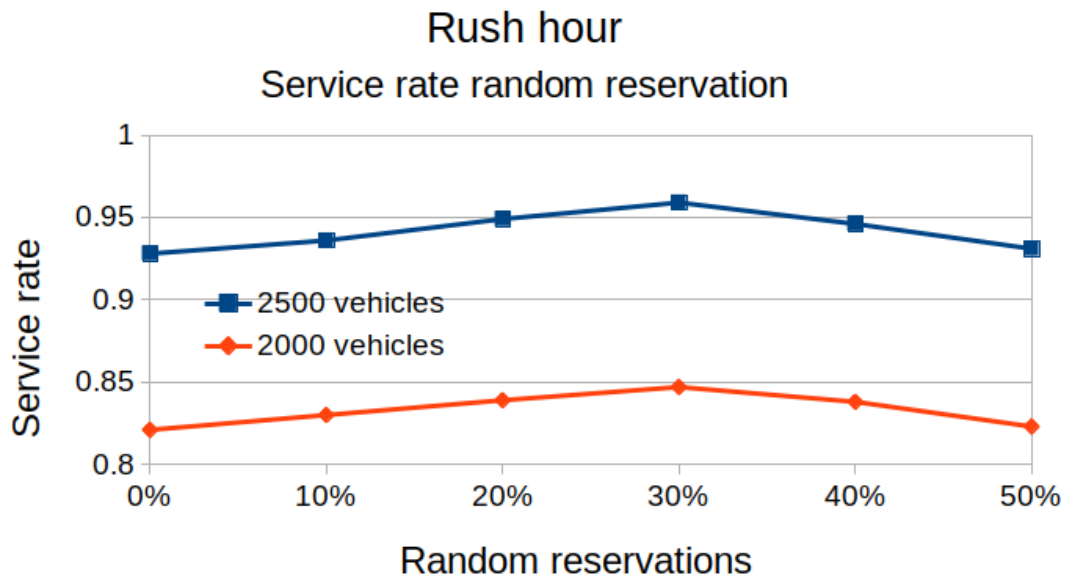


Figure 5.6: Service rate during morning rush hour with 2000 and 2500 vehicles. Random reservations are revealed 5 minutes in advance.

		Random reservations						
		0	10%	20%	30%	40%	50%	
Vehicles	2000	0.821	0.83	0.839	0.848	0.838	0.823	Service rate
		-	1	0.999	0.999	0.998	0.989	Reservation service rate
		304	302	308	308	303	298	Average total delay
		1.22	1.24	1.25	1.25	1.23	1.2	Distance ratio
		402	761	1586	3973	6265	6537	Computational time
	2500	0.928	0.936	0.949	0.959	0.946	0.931	
		-	1	1	1	1	0.999	
		288	286	281	284	285	277	
		1.16	1.17	1.19	1.2	1.18	1.14	
		474	902	2042	5299	6243	6558	

Table 5.13: Detailed results for the performance of 2000 and 2500 vehicles of capacity 4 in the morning rush hour with random reservations revealed 5 minutes in advance.

Based on these results, it can be concluded that the method can be used to increase the service rate during the morning rush hour at a limited computational cost. Having more vehicles allows for serving more passengers at a slightly reduced efficiency from the operator's point of view, as can be seen in the reduced distance ratio. More vehicles reduce the average total delay passengers experience. The service rate for reservations is higher as well with more available vehicles. This is because the system is more flexible and can actually assign the reservations once they are revealed.

6

Conclusions

This chapter summarises the conclusions with regard to the incorporation of reservations into an on-demand ridesharing system. This is followed by a discussion of the research, methods and assumptions. Lastly, future work related to the topic is discussed.

6.1. Conclusion

The objective of this work has been to assess the effectiveness and feasibility for the inclusion of reservations in a large-scale on-demand ridesharing system. Reservations are included in an on-demand ridesharing model by revealing reservations at a certain time in advance to the system. The existing problem formulation is extended with a constraint stating that requests for a ride can only be handled once passengers actually need to be picked up.

The upside from having information on future demand is demonstrated when there are no additional benefits for reservations. Both the service rate and average delay times are improved. From the research, it follows that the service provided to reservations is better on average compared to on-demand requests. The relevance and effectiveness of employing the designed heuristics is demonstrated for scenarios where more information about future demand is given to the system.

If reservations are awarded a distinct benefit, a trade-off emerges with the service level for on-demand requests. Two different benefits are implemented and tested. As a first option, a pledged service constraint is implemented. Reservations revealed in advance have a small cost initially and are assigned all consecutive timesteps, leading to a (far) higher service rate with this constraint applied. The results show a trade-off between servicing on-demand requests and reservations. The second option to let reservations receive a benefit in service is by assigning a higher rejection penalty to these requests. The results show that the service rate for on-demand requests is slightly better compared to the “no benefits” scenario and slightly worse than the pledged service scenario. Overall, an increased rejection penalty serves more on-demand requests if almost all reservations are serviced, it does however not provide any guarantees to reservations that they are not rejected last minute.

The inclusion of reservations is facilitated by developing methods to reduce the number of trips to compute. Different heuristics are developed and tested to determine which trips to compute in the (limited) available time. The results show that large reductions in computation time can be achieved with a limited effect on the solution quality. The tested heuristic is less effective in scenarios where reservations are revealed to the system longer in advance. Scenarios with reservations revealed longer in advance pose different challenges compared to reservations revealed shortly in advance. Therefore, the heuristics should be tuned specifically for the intended time in advance reservations are to be revealed.

In conclusion, the incorporation of reservations into an on-demand ridesharing system can provide significant advantages for system-wide performance and the experienced service of passengers that

place the reservations. A trade-off between providing better service to reservations and on-demand requests exists. However, even with benefits for reservations, the service level for the on-demand passengers is higher on average than without reservations in the system. System-wise, the computational load from reservations is large but can be managed with appropriate heuristics. The heuristics prove to be effective at problem sizes for which they are tuned but not particularly at scenarios with reservations revealed longer in advance. The trends in the results prove not sensitive to the underlying request data set.

Based on the results, this methodology seems most suited to increase the service level for certain on-demand ridesharing systems. The largest gains can be made in a scenario where passengers have the option to leave as soon as possible, or make a reservation to leave in for example 10 minutes from now. The designed methodology is capable of using the available information for better service overall and can provide service guarantees for reservations. The results show that the method can be applied effectively to large scale problems and can be run on standard hardware.

6.2. Discussion

An important consideration would be if passengers see sufficient benefits to using ridesharing more often if the possibility of reservations would be included. Based on the results it is not effective to guarantee service for reservations placed long in advance. Reservations placed a short time in advance can receive a service guarantee, while at the same time increasing system performance. The improved system performance includes better service overall and more effective use of vehicles. Since vehicles drive less distance per passenger serviced, the operators' cost decreases. Depending on market conditions this would thus benefit passengers, operators or both.

A second consideration would be to look at the price of computational power. Results in this study focused on limiting the computational load because of the available hardware. The additional computation power required to handle more information has a certain computational price and represents a certain value. With current trends in increasing computational power, the trade-off in how much additional information is useful and can be applied economically will shift over time towards handling more information. A future extension could also include options for distributed computing, for example on the phone of each driver.

With respect to the designed heuristics, this is just a subset of all possible heuristics. The focus of this research was to identify the trade-offs, benefits and costs of the inclusion of reservations in an on-demand ridesharing model. The designed heuristics are kept simple to provide general usability and provide enough improvement to perform the numerical analysis for the described scenarios. More detailed and dedicated heuristics can be designed for specific situations and improve performance further and/or reduce the computational load.

An important consideration for evaluating the heuristic is that the underlying parameters are tuned on just one scenario. Since this was a scenario with some reservations revealed shortly in advance, the heuristic parameters prevent the model from taking advantage of information that is available longer in advance. Based on the limited computational load used by reservations that are revealed a short time in advance, sufficient computation time would be available to consider reservations revealed longer in advance with appropriate heuristics. Better results can likely be obtained with heuristics tailored to the available information.

Last, the simulations in this research are done with a certain limit on computation time. The time limit is used to mimic the requirements of the system to find solutions fast enough to allow for on-demand ridesharing. The exact value the time limit should have is debatable. This is because it is closely related to computational power. With faster hardware, you make more computations in the same time span. To take this into account, slower hardware can be used with more available time as the answer is not needed in real time for a simulation. The trade-offs in handling more information and the need for heuristics will exist even on the best hardware as the combinatorial problem grows exponentially. By implementing the time limit as in this work, the relevant trade-offs show in the results.

6.3. Future research

Multiple avenues for future research related to the incorporation of reservations in an on-demand ridesharing system are discussed in this section. Ideas for future research on ridesharing in general are provided at the end of the literature chapter.

First, an option would be to improve performance in the proposed model by improving the heuristics. This can consist of three approaches. The first option is to tune the heuristic parameters or make better combinations of the existing heuristics for other reservation scenarios. Moreover, other working principles/information used by the heuristics can be applied to design more sophisticated heuristics that would scale well with large problem sizes and take advantage of different types of information available. Another option would be to use machine learning to predict which trips to compute. This research angle could employ available data to have modern machine learning algorithms design the heuristics. This can identify more/different underlying patterns than humans ever could and therefore allow for the inclusion of more knowledge in the ridesharing formulation.

Second, on-demand requests can be incorporated in a scenario where most of the passengers make day-ahead reservations. This is a different problem and would have its own methods to solve. A reason to solve this problem is to be able to provide service guarantees for day-ahead reservations while increasing system performance. An option could be to investigate static solutions with a slack variable or a method to quantify flexibility in the objective function. These solutions can be computed in advance and don't have to be recomputed all timesteps. Therefore, different types of algorithms could be applied to utilize all available information.

A third option is to design heuristics based on the expected use case. For example, consider a scenario where passengers use ridesharing as last-mile transport. Requests could come from passengers who use the train for long distances and ridesharing only for the last part of their trip. This will creates large spatial and temporal differences in demand around train stations. Heuristics to take advantage of this information can specifically be created.

Bibliography

Aarts, Henk, Bas Verplanken, and Ad van Knippenberg (1998). "Predicting Behavior From Actions in the Past: Repeated Decision Making or a Matter of Habit?" In: *Journal of Applied Social Psychology* 28.15. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1559-1816.1998.tb01681.x, pp. 1355–1374. ISSN: 1559-1816. DOI: 10.1111/j.1559-1816.1998.tb01681.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1559-1816.1998.tb01681.x (visited on 12/15/2021).

Agatz, Niels et al. (Dec. 1, 2012). "Optimization for dynamic ride-sharing: A review". In: *European Journal of Operational Research* 223.2, pp. 295–303. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2012.05.028. URL: https://www.sciencedirect.com/science/article/pii/S0377221712003864 (visited on 12/20/2021).

Alonso-Mora, Javier, Alex Wallar, and Daniela Rus (Sept. 2017). "Predictive routing for autonomous mobility-on-demand systems with ride-sharing". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866, pp. 3583–3590. DOI: 10.1109/IROS.2017.8206203.

Alonso-Mora et al. (2017). "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment". In: *Proceedings of the National Academy of Sciences of the United States of America*. MAG ID: 2565919573. DOI: 10.1073/pnas.1611675114.

Banerjee, Siddhartha et al. (Mar. 10, 2021). "Real-Time Approximate Routing for Smart Transit Systems". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5.2. version: 1. arXiv: 2103.06212. URL: http://arxiv.org/abs/2103.06212 (visited on 11/11/2021).

Beraldi, Patrizia et al. (June 2010). "Efficient Neighborhood Search for the Probabilistic Multi-Vehicle Pickup and Delivery Problem". In: *Asia - Pacific Journal of Operational Research* 27.3. Num Pages: 14 Place: Singapore, Singapore Publisher: World Scientific Publishing Co. Pte., Ltd., pp. 301–314. ISSN: 02175959. URL: https://www.proquest.com/docview/746814920/abstract/1066ECF0AE8C41E0PQ/1 (visited on 11/22/2021).

Cats, Oded et al. (Jan. 14, 2022). "Beyond the dichotomy: How ride-hailing competes with and complements public transport". In: *PLOS ONE* 17.1. Ed. by Wenjia Zhang, e0262496. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0262496. URL: https://dx.plos.org/10.1371/journal.pone.0262496 (visited on 01/20/2022).

Chen and Nie (Apr. 1, 2017). "Connecting e-hailing to mass transit platform: Analysis of relative spatial position". In: *Transportation Research Part C: Emerging Technologies* 77, pp. 444–461. ISSN: 0968-090X. DOI: 10.1016/j.trc.2017.02.013. URL: https://www.sciencedirect.com/science/article/pii/S0968090X17300530 (visited on 11/11/2021).

Fagnant, Daniel J. and Kara M. Kockelman (Jan. 1, 2018). "Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas". In: *Transportation* 45.1, pp. 143–158. ISSN: 1572-9435. DOI: 10.1007/s11116-016-9729-z. URL: https://doi.org/10.1007/s11116-016-9729-z (visited on 11/18/2021).

Fielbaum and Alonso-Mora (Dec. 1, 2020). "Unreliability in ridesharing systems: Measuring changes in users' times due to new requests". In: *Transportation Research Part C: Emerging Technologies* 121, p. 102831. ISSN: 0968-090X. DOI: 10.1016/j.trc.2020.102831. URL: https://www.sciencedirect.com/science/article/pii/S0968090X2030735X (visited on 11/18/2021).

Fielbaum, Andres, Maximilian Kronmueller, and Javier Alonso-Mora (2021). "Anticipatory routing methods for an on-demand ridepooling mobility system". In: *Transportation*, pp. 1–42.

Fielbaum, Bai, and Alonso-Mora (May 1, 2021). "On-demand ridesharing with optimized pick-up and drop-off walking locations". In: *Transportation Research Part C: Emerging Technologies* 126, p. 103061. ISSN: 0968-090X. DOI: 10.1016/j.trc.2021.103061. URL: https://www.sciencedirect.com/science/article/pii/S0968090X21000887 (visited on 11/16/2021).

Fielbaum, Tirachini, and Alonso Mora (2021). "New sources of economies and diseconomies of scale in on-demand ridepooling systems and comparison with public transport". In.

Fu, Zhexi and Joseph Chow (Jan. 1, 2022). “The pickup and delivery problem with synchronized en-route transfers for microtransit planning”. In: *Transportation Research Part E: Logistics and Transportation Review* 157. Publisher: Pergamon, p. 102562. ISSN: 1366-5545. DOI: 10.1016/j.tre.2021.102562. URL: https://www-sciencedirect-com.tudelft.idm.oclc.org/science/article/pii/S1366554521003203 (visited on 12/17/2021).

Haliem, Marina et al. (Dec. 2021). “A Distributed Model-Free Ride-Sharing Approach for Joint Matching, Pricing, and Dispatching Using Deep Reinforcement Learning”. In: *IEEE Transactions on Intelligent Transportation Systems* 22.12. Conference Name: IEEE Transactions on Intelligent Transportation Systems, pp. 7931–7942. ISSN: 1558-0016. DOI: 10.1109/TITS.2021.3096537.

Hall, Jonathan D., Craig Palsson, and Joseph Price (Nov. 1, 2018). “Is Uber a substitute or complement for public transit?” In: *Journal of Urban Economics* 108, pp. 36–50. ISSN: 0094-1190. DOI: 10.1016/j.jue.2018.09.003. URL: https://www.sciencedirect.com/science/article/pii/S0094119018300731 (visited on 12/14/2021).

Hörl, Sebastian and Felix Zwick (Jan. 2022). “Traffic Uncertainty in On-Demand High-Capacity Ride-Pooling”. In: *101st Annual Meeting of the Transportation Research Board (TRB)*. Washington D.C., United States. URL: https://hal.archives-ouvertes.fr/hal-03405574 (visited on 12/13/2021).

Kaddoura, Ihab and Tilmann Schlenther (Jan. 1, 2021). “The impact of trip density on the fleet size and pooling rate of ride-hailing services: A simulation study”. In: *Procedia Computer Science*. The 12th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 4th International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops 184, pp. 674–679. ISSN: 1877-0509. DOI: 10.1016/j.procs.2021.03.084. URL: https://www.sciencedirect.com/science/article/pii/S1877050921007213 (visited on 11/11/2021).

Kim, Myungseob and P. Schonfeld (2014). “Integration of conventional and flexible bus services with timed transfers”. In: *Transportation Research Part B: Methodological*. DOI: 10.1016/J.TRB.2014.05.017.

Kucharski and Oded Cats (Sept. 1, 2020). “Exact matching of attractive shared rides (ExMAS) for system-wide strategic evaluations”. In: *Transportation Research Part B: Methodological* 139, pp. 285–310. ISSN: 0191-2615. DOI: 10.1016/j.trb.2020.06.006. URL: https://www.sciencedirect.com/science/article/pii/S0191261520303465 (visited on 01/03/2022).

Kucharski, Fielbaum, et al. (Dec. 10, 2021). “If you are late, everyone is late: late passenger arrival and ride-pooling systems’ performance”. In: *Transportmetrica A: Transport Science* 17.4, pp. 1077–1100. ISSN: 2324-9935, 2324-9943. DOI: 10.1080/23249935.2020.1829170. URL: https://www.tandfonline.com/doi/full/10.1080/23249935.2020.1829170 (visited on 01/03/2022).

Li, Xiaoming et al. (Dec. 2020). “A Data-Driven Dynamic Stochastic Programming Framework for Ride-Sharing Rebalancing Problem under Demand Uncertainty”. In: *2020 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*. 2020 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom), pp. 1120–1125. DOI: 10.1109/ISPA-BDCloud-SocialCom-SustainCom51426.2020.00167.

Li, Yinglei and Sung Hoon Chung (Nov. 1, 2020). “Ride-sharing under travel time uncertainty: Robust optimization and clustering approaches”. In: *Computers & Industrial Engineering* 149, p. 106601. ISSN: 0360-8352. DOI: 10.1016/j.cie.2020.106601. URL: https://www.sciencedirect.com/science/article/pii/S0360835220303351 (visited on 12/08/2021).

Lowalekar, Meghna, Pradeep Varakantham, and Patrick Jaillet (Jan. 11, 2021). “Zone pAth Construction (ZAC) based Approaches for Effective Real-Time Ridesharing”. In: *Journal of Artificial Intelligence Research* 70, pp. 119–167. ISSN: 1076-9757. DOI: 10.1613/jair.1.11998. URL: https://jair.org/index.php/jair/article/view/11998 (visited on 02/08/2022).

Luo, Qi et al. (Aug. 19, 2021). “Efficient Algorithms for Stochastic Ridepooling Assignment with Mixed Fleets”. In: *arXiv:2108.08651 [cs, math]*. arXiv: 2108.08651. URL: http://arxiv.org/abs/2108.08651 (visited on 12/14/2021).

Ma, Tai Yu et al. (Aug. 2019). “A dynamic ridesharing dispatch and idle vehicle repositioning strategy with integrated transit transfers”. In: *Transportation Research, Part E: Logistics and Transportation Review* 128, pp. 417–442. ISSN: 1366-5545. DOI: 10.1016/j.tre.2019.07.002. URL: http:

//www.scopus.com/inward/record.url?scp=85068868692&partnerID=8YFLogxK (visited on 11/11/2021).

Maheo, Arthur, Philip Kilby, and Pascal Van Hentenryck (Feb. 2019). "Benders Decomposition for the Design of a Hub and Shuttle Public Transit System". In: *Transportation Science* 53.1, pp. 77–88. ISSN: 0041-1655, 1526-5447. DOI: 10.1287/trsc.2017.0756. arXiv: 1601.00367. URL: http://arxiv.org/abs/1601.00367 (visited on 11/11/2021).

OECD (2015). "Urban Mobility System Upgrade How shared self-driving cars could change city traffic". In.

Pandey, Venktesh et al. (Nov. 2019). "On the needs for MaaS platforms to handle competition in ridesharing mobility". In: *Transportation Research Part C: Emerging Technologies* 108, pp. 269–288. ISSN: 0968090X. DOI: 10.1016/j.trc.2019.09.021. arXiv: 1906.07567. URL: http://arxiv.org/abs/1906.07567 (visited on 12/15/2021).

Pelzer, Dominik et al. (Oct. 2015). "A Partition-Based Match Making Algorithm for Dynamic Ridesharing". In: *IEEE Transactions on Intelligent Transportation Systems* 16.5. Conference Name: IEEE Transactions on Intelligent Transportation Systems, pp. 2587–2598. ISSN: 1558-0016. DOI: 10.1109/TITS.2015.2413453.

Pinto, Helen K.R.F. et al. (2018). "Joint design of multimodal transit networks and shared autonomous mobility fleets". In: *Transportation Research Procedia* 38, pp. 98–118. ISSN: 2352-1457. DOI: 10.1016/j.trpro.2019.05.007. URL: http://www.scopus.com/inward/record.url?scp=85074937060&partnerID=8YFLogxK (visited on 11/11/2021).

Riley, Connor, Pascal Van Hentenryck, and Enpeng Yuan (Mar. 24, 2020). "Real-Time Dispatching of Large-Scale Ride-Sharing Systems: Integrating Optimization, Machine Learning, and Model Predictive Control". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*. arXiv: 2003.10942. URL: http://arxiv.org/abs/2003.10942 (visited on 02/11/2022).

Santi, Paolo et al. (Sept. 16, 2014). "Quantifying the benefits of vehicle pooling with shareability networks". In: *Proceedings of the National Academy of Sciences* 111.37, pp. 13290–13294. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1403657111. URL: http://www.pnas.org/lookup/doi/10.1073/pnas.1403657111 (visited on 11/18/2021).

Shah, Sanket, Meghna Lowalekar, and Pradeep Varakantham (Apr. 3, 2020). "Neural Approximate Dynamic Programming for On-Demand Ride-Pooling". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.1. Number: 01, pp. 507–515. ISSN: 2374-3468. DOI: 10.1609/aaai.v34i01.5388. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5388 (visited on 02/07/2022).

Shen, Yu, Hongmou Zhang, and Jinhua Zhao (July 1, 2018). "Integrating shared autonomous vehicle in public transportation system: A supply-side simulation of the first-mile service in Singapore". In: *Transportation Research Part A: Policy and Practice* 113, pp. 125–136. ISSN: 0965-8564. DOI: 10.1016/j.tra.2018.04.004. URL: https://www.sciencedirect.com/science/article/pii/S096585641730681X (visited on 11/11/2021).

Simonetto, Andrea, Julien Monteil, and Claudio Gambella (Apr. 1, 2019). "Real-time city-scale ridesharing via linear assignment problems". In: *Transportation Research Part C: Emerging Technologies* 101, pp. 208–232. ISSN: 0968-090X. DOI: 10.1016/j.trc.2019.01.019. URL: https://www.sciencedirect.com/science/article/pii/S0968090X18302882 (visited on 12/13/2021).

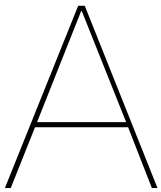
Tirachini, Alejandro, David A. Hensher, and John M. Rose (July 1, 2013). "Crowding in public transport systems: Effects on users, operation and implications for the estimation of demand". In: *Transportation Research Part A: Policy and Practice* 53, pp. 36–52. ISSN: 0965-8564. DOI: 10.1016/j.tra.2013.06.005. URL: https://www.sciencedirect.com/science/article/pii/S0965856413001146 (visited on 11/16/2021).

Tsao, Matthew et al. (May 2019). "Model Predictive Control of Ride-sharing Autonomous Mobility-on-Demand Systems". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019 International Conference on Robotics and Automation (ICRA). ISSN: 2577-087X, pp. 6665–6671. DOI: 10.1109/ICRA.2019.8794194.

Van Woensel, T. et al. (May 1, 2008). "Vehicle routing with dynamic travel times: A queueing approach". In: *European Journal of Operational Research* 186.3, pp. 990–1007. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2007.03.012. URL: https://www.sciencedirect.com/science/article/pii/S0377221707003116 (visited on 11/30/2021).

Wardman, M., J. Hine, and S. Stradling (2001). “INTERCHANGE AND TRAVEL CHOICE - VOLUMES 1 AND 2”. In: *TRANSPORT RESEARCH SERIES*. ISBN: 9780755930302. ISSN: 0950-2254. URL: https://trid.trb.org/view/735076 (visited on 12/17/2021).

Xi, Haoning et al. (Sept. 17, 2021). “Single-leader multi-follower games for the regulation of two-sided Mobility-as-a-Service markets”. In: *arXiv:2106.02151 [cs, math]*. arXiv: 2106.02151. URL: http://arxiv.org/abs/2106.02151 (visited on 01/03/2022).



Research paper

The research paper version of this thesis is on the following pages of this appendix.

Incorporation of reservations in an on-demand ridesharing system

Author: Nander Theodoridis
Supervisors:
Andres Fielbaum, Bilge Atasoy, Javier Alonso Mora

With increasing urbanization and a need to reduce greenhouse gas emissions, new forms of urban mobility are studied worldwide. An interesting transportation method is ridesharing, where people can share parts of rides with other passengers travelling in a similar direction in the same car.
In this work, the inclusion of reservations (pre-bookings) into the on-demand ridesharing problem is studied. The potential benefit from the inclusion of reservations is in the better assignment that can be made with knowledge about a part of the future demand. The main challenge from reservations is the increased computational load required. This paper proposes multiple adjustments and different heuristics for a state-of-the-art on-demand ridesharing system to deal with the requirements of reservations. An additional consideration is a trade-off between providing benefits for reservations and serving on-demand requests.
Numeric experiments are performed on the Manhattan street network with NYC taxicab request data. The results indicate that reservations can be included in on-demand ridesharing at a tolerable computational cost with the proposed framework. Passengers that make reservations shortly in advance can be guaranteed service once initially accepted by the system. Having passengers make reservations in advance does increase the overall service the system can provide.

1 Introduction

With increasing urbanization and a need to reduce greenhouse gas emissions, new forms of urban mobility are studied worldwide. An interesting transportation method is ridesharing, where people can share parts of rides with other passengers travelling in a similar direction in the same car. The type of ridesharing reviewed in this paper is where a dedicated fleet of vehicles is centrally controlled to provide transportation through ridesharing.

With current technology, large numbers of passengers can be efficiently allocated for ridesharing [9]. 98% of taxi demand in Manhattan can be served with just 30% of the number of taxis, while staying within reasonable constraints on delays [3]. In general, ridesharing shows a potential upside for users, operators and society as a whole from the widespread adoption of ridesharing mobility [2]; [8]. The advantages of ridesharing could potentially be amplified by the adoption of autonomous vehicles (AVs). However, ridesharing is still not employed on a large scale in reality. For potential users (passengers) and/or operators, the perceived disadvantages apparently outweigh the offered advantages.

To improve the attraction of ridesharing systems, research into extensions to the existing ridesharing model formulation is required. The goal is to increase the advantages and/or reduce the perceived disadvantages. If successful, the real-world adoption and the corresponding benefits might become more widespread. One potentially relevant extension to the rideshare formulation is the inclusion of reservations. A reservation (pre-booking) is a request that is known to the system a certain time before its desired pick-up time.

Reservations would provide the system with additional knowledge about how part of the demand in the future will look like and thus allow for better assignment in the current timestep. Increased performance with better anticipation of future demand is shown in literature [6].

From a model's perspective, incorporating reservations into the ridesharing problem formulation brings challenges. Firstly, a higher computational load can be expected. Secondly, it is reasonable to assume passengers will only make reservations if placing a reservation is associated with benefits.

For this research, reservations that are placed shortly in advance are considered. This is applicable in an on-demand ridesharing system where passengers can indicate if they want to be served as soon as possible or intend to leave in for example 10 minutes.

Research objective

The objective of this work is to assess the effectiveness and feasibility of the incorporation of reservations in an on-demand ridesharing system.

- Include reservations efficiently into a state-of-the-art on-demand ridesharing method, allowing for implementation on a large scale.
- Quantify the effects of having reservations available on multiple KPIs and review the imposed computational burden.
- Study the trade-offs in providing benefits to reservations over normal requests.

Contribution statement

A state-of-the-art on-demand ridesharing system is adjusted to allow for reservations to be assigned together with on-demand requests. The adjustments focus on decreasing the imposed computational load by limiting the computation cost for passengers and using heuristics to restrict the combinations of passengers and reservations. The result is a lightweight method able to incorporate reservations made shortly in advance and benefit from the additional information.

More specifically, the contributions are listed below:

- Detailed experiments are performed to gain numeric insights into the potential benefits from the inclusion of reservations and the imposed computational cost.
- A numerical study about the trade-off between providing improved service for reservations over on-demand requests.

- The development, application and testing of multiple heuristics to limit the computational burden imposed by the inclusion of reservations is performed.

Structure of this paper

The work in the research is presented as follows. In section 2, relevant background literature is discussed. The methodology of the research is described in section 3. The section consists of two parts. The first part describes the problem formulation. The second part provides insights into a state-of-the-art method and contains the methodological extensions used to include reservations. Section 4 contains the experimental setup for the research. The results are discussed in section 5. Here the potential benefits and trade-offs concerning reservations are analyzed. Finally, section 6 consists of conclusions, discussions and recommendations for further work.

2 Related literature

The main challenge in ridesharing is to assign a set of passengers to a set of vehicles that service the demand for mobility. Previous work shows that most demand in Manhattan can be serviced by shared vehicles [9]. The assignment of passengers to vehicles can be performed statically and dynamically.

In a static assignment, all decisions are made beforehand. This allows for additional computation time available and predictable service. The predictability of service is an important aspect of the perceived level of service [1].

The option to dynamically assign passengers during operation allows for attractive on-demand service to be implemented. The dynamic nature of the problem demands efficient algorithms as large problems have to be computed in reasonable time to allow for re-assignment as new information becomes available. The assignment is usually performed in batches to allow for better matching between requests and vehicles. The method proposed by Alonso Mora et al.[3] (used in the methodology of this paper) combines requests into groups that are collectively assigned to vehicles. The method allows for anytime optimal assignment in large-scale ridesharing systems. A modification is made by Simonetto et al.[10], where requests are assigned individually to vehicles in shorter batch times. This heuristic allows for a speedup in computation time at the cost of the service level provided.

Significant research to improve ridesharing by estimating future demand is performed. The use of historic demand distribution to predict future requests is done in [4]. More historic data is processed with machine learning to more accurately predict demand in [7]. Alternatively, current system parameters can be used to anticipate future demand and take it into account when assigning passengers to vehicles [6]. A different approach is taken in [11], which incentives some vehicles to remain idle to handle future demand. These different methods show that (estimated) information on future demand improves the service level that can be offered.

The incorporation of reservations in an on-demand ridesharing system is to the best of my knowledge only done by [5]. In this method reservations (pre-bookings) are placed a day ahead and confirmed or rejected directly. Acceptance of a reservation is considered binding to the operator.

Opposed to [5], this paper focuses on reservations that are made shortly in advance.

3 Methodology

The methodology section is structured as follows: First, the problem description is provided. Second, modifications to a state-of-the-art model to allow for reservations are discussed.

3.1 Problem description

The goal of on-demand ridesharing is to assign a set of travel requests to a set of vehicles with minimal overall costs and within predetermined constraints. Requests are collected in batches during 30 seconds and assigned collectively. The problem takes place on a directed graph $G = (N, E)$, with N the nodes and E the edge in the graph. The edges are directional and weighted based on the required travel time of the road segment it represents.

Waiting time for request r w_r is the time between when a request emerges r_e and when the request is picked up r_p, $w_r = r_p - r_e$. Detour time is the delay a passenger on board a vehicle experiences compared to an unshared vehicle. The detour time for request r d_r is given by the drop off time r_d minus the pickup time minus the minimum travel time r_{min}, $d_r = r_d - r_p - r_{min}$. The waiting time and detour time combined give the total delay time for a request δ_r, $\delta_r = w_r + d_r$.

The goal of the assignment is to minimize Equation 1. In this equation, the sum of the detour times for all passengers $\mathcal{P}_v$ on board vehicle v, the total delays δ_r for all assigned requests $\mathbb{R}_{ok}$ plus the rejection penalty Π_r for all requests that are not assigned $\mathbb{R}_{no}$, is minimized. This is done for each vehicle in the set of all vehicles $\mathbb{V}$.

$$\sum_{v \epsilon V} \sum_{r \epsilon P_v} d_r + \sum_{r \epsilon \mathbb{R}_{ok}} \delta_r + \sum_{r \epsilon \mathbb{R}_{no}} \Pi_r \tag{1}$$

Assignment of requests to vehicles is done via an integer linear program (ILP) method. To use this method, requests are combined into trips. Trips are assigned to vehicles based on the cost of serving that trip. The method is similar to [3]. A trip $\mathcal{T}$ is a group of requests that can be assigned together to the same vehicle.

Trip computation A trip is feasible if all requests in that trip and all passengers on board the vehicle to serve that trip can be serviced within the waiting and detour time constraints. To include reservations in this framework, they can simply be added to the set of requests to be assigned.

Individual constraints ensure the maximum waiting time Ω and maximum detour time Δ are not exceeded. Furthermore, the vehicle capacity can not be exceeded at any time during the trip.

The cost of adding a trip to a vehicle is given in Equation 3.1. If multiple feasible routes (sequence of pick up and drop off actions) are found for a trip, the one with the lowest cost is selected.

$$cost(\mathcal{T}, v) = \sum_{r \epsilon \mathcal{P}_v} d_r + \sum_{r \epsilon \mathcal{T}} \delta_r \tag{2}$$

The goal of the ILP is to assign requests, via trips, to vehicles and minimize cost overall.

Feasible trip-vehicle combinations are computed for increasing size with the property that for trip (T, V) to be feasible, all (T', V) with $T' \in T$ have to be viable. Last, idle vehicles are rebalanced to nodes with unserved demand. The ILP formulation is given in Equation 3. The ILP is initialized with an updated version of the previous assignment.

ILP assignment The ILP assignment has two sets of decision parameters. First, $\epsilon_{i,j} \quad \forall e(\mathcal{T}_i, v_j)$ is a parameter to describe a feasible combination between trips and each vehicle that can serve that trip. This is a binary parameter, whether a trip is assigned to a vehicle or not. Second, $\mathcal{X}_r \quad \forall r \in \mathbb{R}$ is a parameter if a request is assigned to a vehicle. This is also a binary parameter since a request can only be assigned to one vehicle or not be assigned. $\mathcal{X}_r = 1$ means that request r is not assigned.

The constraints in the ILP formulation will ensure that each request is assigned to one vehicle or ignored. Also, each vehicle will have at most one trip assigned.

The ILP has the number of edges in the $e(\mathcal{T}, v)$ set, plus the number of requests as decision parameters. The number of constraints is equal to the number of vehicles plus the number of requests. Three subsets are used in the ILP formulation, namely $\mathbb{T}_{v=j}$ is the set of trips that can be served by vehicle j. $\mathbb{T}_{\mathcal{R}=r}$ is the set of trips that contain request r. $\mathbb{V}_{\mathcal{T}=i}$ is the set of vehicles that can serve trip i

$$\begin{aligned} \text{minimize} \quad & \sum_{\mathcal{T}_i \in \mathbb{T}} \sum_{v_j \in \mathbb{V}} c_{i,j} * \epsilon_{i,j} + \sum_{r \in \mathbb{R}} \Pi_r * \mathcal{X}_r \\ \text{subject to} \quad & \sum_{i \in \mathbb{T}_{v=j}} \epsilon_{i,j} \leq 1 \qquad \forall v_j \in \mathbb{V} \\ & \sum_{i \in \mathbb{T}_{\mathcal{R}=r}} \sum_{j \in \mathbb{V}_{\mathcal{T}=i}} \epsilon_{i,j} + \mathcal{X}_r = 1 \qquad \forall r \in \mathbb{R} \end{aligned} \tag{3}$$

Once the assignment procedure is done, it is communicated to the vehicles. If a request is not picked up before the next time step, it is put back in the set of requests to be assigned.

An additional constraint that can make ridesharing more user-friendly is implemented. This constraint states that all passengers that have been assigned to a vehicle before, and thus expect to be served, have to be served by the system. This constraint will only be applied to reservations.

For this purpose, an additional subset $\mathbb{R}_{rb}$ is defined for all reservations that have been assigned before.

In mathematical form this can be written as:

$$\sum_{\mathcal{T}_i \epsilon \mathbb{T}_{\mathcal{R}=r}} \sum_{j \epsilon \mathbb{V}_{\mathcal{T}=i}} \epsilon_{i,j} = 1 \quad \forall r \in \mathbb{R}_{rb} \tag{4}$$

3.2 Modifications

The computational load from the inclusion of reservations into the set of requests to be assigned will grow exponentially if no provisions are taken.

First of all, reservations could in the worst case be available for each trip of a certain vehicle. This phenomenon is shown for one reservation in Figure 1, where a blue reservation is added.

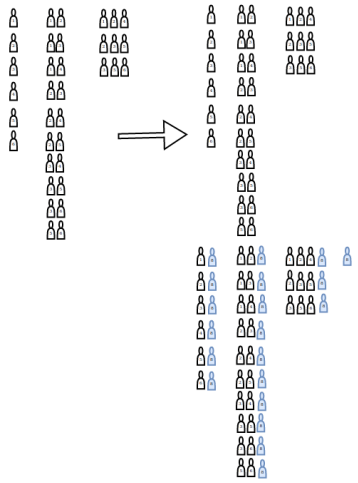


Figure 1: Worst case growth in the number of feasible trips if a blue reservation is added

A second effect is that reservations can be assigned to more vehicles than on-demand requests. Therefore, extra trips have to be computed for more vehicles compared to on-demand requests.

Third, reservations stay in the request pool longer than on-demand requests. This means that the potential trips are computed more often as well.

Not only are there more trips, but the trips also become longer and thus significantly more expensive to compute. As shown in Table 1 larger trip sizes are especially expensive as you compute more new passengers. Since reservations can be added to more trips than on-demand requests, the inclusion of reservations has a significant impact on the amount of larger trip sizes that are available.

		New Passengers			
		1	2	3	4
On board passengers	0	1	6	90	2520
	1	3	30	630	22680
	2	6	90	252	113400
	3	10	210	7560	415800
	4	15	420	18900	1247400

Table 1: Number of potentially feasible permutations with insertion heuristic

It should be noted that additional on-demand requests have a scaling effect on the computational load as well. However, reservations do scale much worse in practice.

Because of the scaling effects from reservations, countermeasures are required. First off, reservations are revealed to the system a limited time in advance, even when placed longer in advance. Second, an insertion heuristic is applied to trip computation cost. The insertion heuristic locks the order in which passengers already on board are dropped off. This reduces the number of possible permutations by a factor of "passengers on board" factorial compared to not using this heuristic.

Further steps to limit the impact on computation time are required.

The additional methodology proposed in this research consists of 2 parts. The first part is to limit the cost of computing a trip by applying logic checks on which permutations to consider. For the second part, heuristics are developed to determine which trips should be considered. The goal is to minimize the number of trips to compute the costs while maintaining the performance of the system.

Logic checks

Multiple logic checks are used to determine if it is sensible to compute the cost for a certain permutation. Permutations are different sequences in which pick-up/drop-off actions are planned. The idea is that considering a logic check is computationally far cheaper than computing the cost for a permutation. The checks done initially are as follows: 1) The permutation does not start with a subset previously determined to be infeasible to start with. 2) Reservations are limited in their position in the sequence to respect the time constraints for other passengers. 3) The vehicle capacity is not violated (constraint). Furthermore, only permutations that could provide a valid solution are considered. These considerations can be computed beforehand once and tested against the permutation to be considered.

Once a permutation passes the logic tests, the actual feasibility and cost is checked based on waiting and detour time constraints. If the permutation turns out to be infeasible, the subset leading up to the constraint is added to the list of infeasible subsets. The same principle is applied to the other logic checks. If a permutation is not feasible according to one of the checks, the subset leading to that point is added to the infeasible set.

Heuristics

Heuristics have been designed to reduce the computational load while maintaining as much as possible of the advantages from reservations. The goal of the heuristics is to reduce the number of trips to be computed by only computing relevant trips. An assumption made in the design of heuristics is that trips with a lower cost are easier to combine. The cost of a trip is correlated with the margin to the constraints for waiting and detour time. A lower cost thus indicates more flexibility while staying within the set constraints.

Based on testing a heuristic is designed and tuned on an average scenario. The heuristic consists of two parts. First, the heuristic limits the trip size to be considered to three. Second, all requests (including reservations) are only considered by the X cheapest vehicles in each iteration (we use X = 20).

4 Experimental setup

The network for the experiments is from Manhattan, New York City. The graph describing the network consists of 4091 nodes and 9454 directional edges. 1000 vehicles with a capacity of four passengers are used for the simulation. The shortest route and distance between each pair of points in the graph are calculated beforehand using the Dijkstra algorithm and stored in a lookup table.

Requests are taken from actual data for taxi demand in Manhattan from 12 AM till 1 PM, on the 15th of January 2013. The set of requests consists of 10548 valid requests to be considered. Each request has starting and end coordinates. These have been matched to the closest node in the road network. The requests consist of pick-up and drop-off nodes, request time and group size. For the numerical simulations, all requests are for 1 person.

The standard test scenario used consists of a warm-up phase and a “real" part. The goal of the warm-up phase is to approximate what the initial conditions would be with service in the previous hour. This includes the location and occupation (with the accompanying constraints on waiting/detour time) for each vehicle.

In the warm-up phase, 5250 requests are added to the system over half an hour. The set of warm-up requests is randomly drawn without redraw from all possible requests and equally spaced over the available time for warm-up. Service for the warm-up set is not considered in the KPIs evaluated.

The results in this paper will be discussed for random reservations and for “rejections as reservations". In the case of random reservations, a sample of all requests is taken and the reveal time for these requests is reduced with the time in advance the requests are revealed to the system. For rejections as reservations, the requests that are not served without reservations are assumed to make a reservation. This represents a set of requests that are harder to service in the case without reservations.

To examine the sensitivity towards the underlying request data, a second set of requests from Manhattan is used for this purpose. The control set is taken from the morning peak. There is a significant difference in origin/destination distribution, therefore the set can be used for sensitivity analysis. The distributions are shown in Figure 2. The morning request set consists of 28030 requests. To be comparable with the noon data set, 10548 unique samples will be randomly drawn from this morning hour. The warm-up data for the test set is drawn from the same request set.

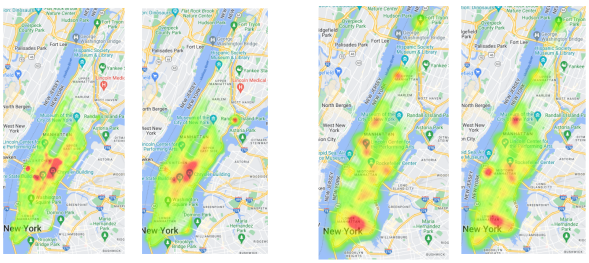


Figure 2: Origin and destination distribution for the test case and the sensitivity analysis.

5 Results

In this section the results are discussed. First, the impact of benefits for reservations are discussed. Second, the effects from the designed heuristic are shown. Third, the results

for scenarios with both pledged service and the heuristics are evaluated. Last, the sensitivity of the results to the request data is analyzed.

Pledged service

Passengers that place a reservation would expect some to receive a benefit for making that reservation. This benefit can be provided using the pledged service constraint or by increasing the rejection penalty for reservations.

A comparison between the average service rate and the service rate for reservations (dotted lines) is shown in Figure 3. The experiments are performed with random reservations revealed 5 minutes in advance.

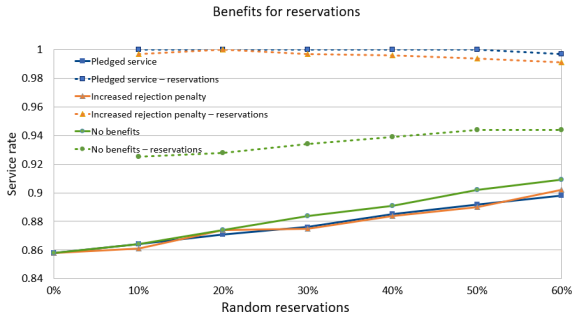


Figure 3: Effects on service rate if benefits are assigned to reservations.

From the figure it shows that providing benefits to reservations does lead to a small reduction in average service rates compared to a "no benefits" scenario. The service rate is on average still higher compared to a scenario without reservations.

Reservations do receive slightly better service in case of pledged service over the increased rejection penalty at higher reservation fractions. This is because the pledged service constraint does differentiate between reservations and the increased rejection penalty does not.

Based on the objective to make ridesharing more attractive, the further sections of the results will be discussed with the pledged service constraint applied for all reservations.

Heuristics

More available information does lead to better possible assignments and thus increased system performance. However, a limit on the available computational time is applied to allow for on-demand service. The impact of more information on the service rate is shown in Figure 4. The orange line is without heuristics and the blue line a simulation with heuristics. In this experiment all reservations are revealed to the system 5 minutes in advance.

Initially a steady increase in service rate is reached with more reservations. The reason for the drop in performance of the orange line is given in Figure 5. The computation time for each scenario is shown with and without the heuristics. It is clear that for more reservations the experiment without heuristics reaches the computational time limits and deteriorates in performance as a consequence.

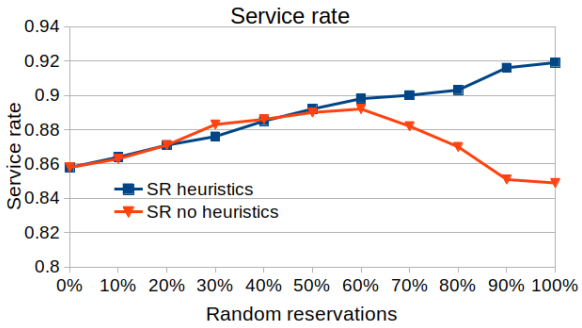


Figure 4: Service rate with random reservations revealed 5 minutes in advance.

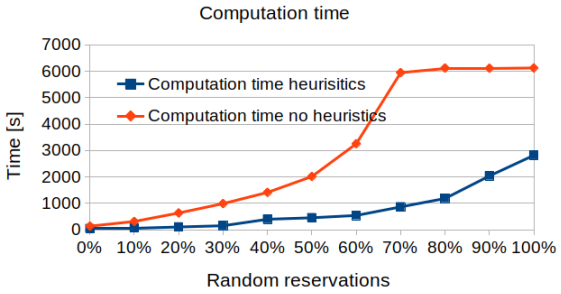


Figure 5: Computational time with random reservations revealed 5 minutes in advance.

The reason to use heuristics is to allow for more information to be used effectively by the model. This means that more reservations can be revealed to the system longer in advance. The heuristics will still reach a point where more information decreases system performance. However, this point is reached later and therefore a better solution becomes computationally feasible. A second reason to apply the heuristic is to be able to simulate rush hour demand with more requests per hour.

The heuristic will be used in the following results in this paper.

5.1 Combined results

The results from including reservations in an on-demand ridesharing system are summarized in table 2. The results are with the described heuristic and the pledged service constraint. Each cell in the table shows the service rate, service rate for reservations and the computational time used for the simulation.

Based on these results it can be concluded that additional reservations improve performance more than reservations that are revealed longer in advance. In case rejections are used as reservations, the average service rate is lower compared to random requests. The costly requests that made a reservation are all served, this shows the trade-off in providing benefits for reservations.

If too much information becomes available, the service rate can deteriorate significantly to levels lower than without reservations. Reservations that were placed are still being served significantly better because of the pledged service constraint.

The average service rate decreases partly because the heuris-

		Time revealed in advance [minutes]			
		5	10	15	20
Random reservations	0%	0.858 - 39	0.858 - 39	0.858 - 39	0.858 - 39
SR Reservation SR sim time	10%	0.864 1 61	0.868 1 107	0.874 1 240	0.874 1 921
	20%	0.871 1 103	0.883 1 382	0.885 1 2136	0.834 1 2367
	30%	0.874 1 151	0.888 1 2677	0.8 1 2730	0.804 0.993 2775
	40%	0.885 1 395	0.836 1 2786	0.806 0.985 2890	0.808 0.976 3101
	50%	0.892 1 639	0.819 0.995 2932	0.816 0.965 3036	0.805 0.959 3435
Rejections as reservations	14.2%	0.864 1 75	0.871 1 325	0.877 1 1209	0.862 1 3242

Table 2: Result from the incorporation of reservations in an on-demand ridesharing system for different percentages of random reservations revealed a certain time in advance.

tic parameters are tuned on just one scenario with reservations revealed shortly in advance. Since the tuning is not effective for all scenarios, the model can not both adhere to the constraint and pick effective options as these are not sufficiently created.

Reservations revealed shortly in advance do show significant gains to be achieved and good service for reservations.

The service rate for each result in the table is plotted in Figure 6.

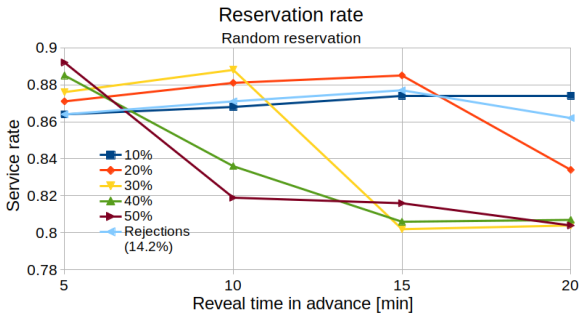


Figure 6: Visualization of the service rates as in Table 2.

5.2 Sensitivity

Experiments are run with the morning peak request set to assess the impact of the underlying request set. Since the heuristic parameters are optimized on reservations being revealed shortly in advance, these settings will be used for the comparison in Figure 7. All reservations are revealed 5 minutes in advance. The orange line indicates the service rate in the morning hour. The dotted lines indicate the service rate for the passengers that placed a reservation.

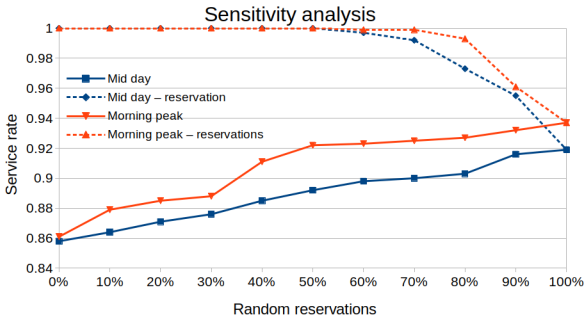


Figure 7: Simulations for sensitivity data set compared to the test set. Requests are all revealed 5 minutes in advance.

Service rate is slightly higher in the morning peak as there is a bit more overlap in desired trajectories for passengers. Trends in performance are similar and more reservations revealed shortly in advance increase the service rate significantly in both scenarios.

From these results it can be concluded that the method is not sensitive to the underlying spatial distribution of origin and demand for the requests.

6 Conclusion

The objective of this work has been to assess the effectiveness and feasibility of the inclusion of reservations in a large-scale on-demand ridesharing system.

In conclusion, the incorporation of reservations into an on-demand ridesharing system can provide significant advantages for system-wide performance and the experienced service level for passengers that make a reservation. A trade-off between providing better service to reservations and on-demand requests exists. System-wise, the computational load from reservations is large but can be managed with appropriate heuristics if reservations are not revealed far in advance. The trends in the results prove not sensitive to the underlying request data set.

Based on the results, this methodology seems most suited to increase service of on-demand ridesharing systems where passengers have the option to leave as soon as possible or make a reservation to leave in for example 10 minutes from now. The methodology is capable of using the information for better service overall and can provide service guarantees for these kinds of reservations.

Discussion

An important consideration would be if passengers see sufficient benefits to use ridesharing more often if the possibility of reservations would be included. Based on the results, it is not effective to guarantee service for reservations placed long in advance in this methodology. Reservations placed a short time in advance can receive a service guarantee, while at the

same time increasing system performance.

A second consideration would be to look at the price of computational power. Results in this study focused on limiting the computational load because of the available hardware. The additional computation power required to handle more information has a certain computational price and provides a certain value. With current trends in increasing computational power, the trade-off in how much additional information is useful and can be applied economically will shift over time towards handling more information. A future extension could also include options for distributed computing, for example on the phone of each driver.

Concerning the designed heuristic, the components represent just a subset of all possible heuristics. The focus of this research was to identify the trade-offs, benefits and costs of the inclusion of reservations in an on-demand ridesharing model. The designed heuristics are kept simple to provide general usability and provide enough improvement to perform the numerical analysis for the described scenarios. More detailed and dedicated heuristics can be designed for specific situations and improve performance further and/or reduce the computational load.

An important consideration for evaluating the heuristic is that the underlying parameters are tuned on just one scenario. Since this was a scenario with some reservations revealed shortly in advance, the heuristic parameters prevent the model from taking advantage of information that is available longer in advance. Based on the limited computational load used by reservations that are revealed a short time in advance, sufficient computation time would be available to consider reservations revealed longer in advance with appropriate heuristics. Better results can likely be obtained with heuristics tailored to the current status and number of reservations in the network.

Future research

In this research, the goal was to include reservations in an on-demand ridesharing problem. Relevant future work could focus on two aspects of this problem.

First, an option would be to improve performance in the proposed model by improving the heuristics. Other working principles/information used by the heuristics can be applied to design more sophisticated heuristics that would scale well with large problem sizes and take advantage off different types of information available.

Another option would be to use machine learning to predict which trips to compute. This research angle could employ available data to have modern machine learning algorithms design the heuristics. This can identify more/different underlying patterns than humans ever could and therefore allow for the inclusion of more knowledge in the ridesharing formulation.

Second, on-demand requests can be incorporated in a scenario where most of the passengers make reservations sufficiently long in advance. This is a different problem and would have its own methods to solve. A reason to solve this problem is to be able to provide service guarantees for day-ahead reservations.

References

[1] Henk Aarts, Bas Verplanken, and Ad van Knippenberg. Predicting behavior from actions in the past: Repeated decision making or a matter of habit? 28(15):1355–1374. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1559-1816.1998.tb01681.x.

[2] Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Optimization for dynamic ride-sharing: A review. 223(2):295–303.

[3] Alonso-Mora, Samaranayake, Wallar, Frazzoli Emilio, and Rus Daniela. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. MAG ID: 2565919573.

[4] Javier Alonso-Mora, Alex Wallar, and Daniela Rus. Predictive routing for autonomous mobility-on-demand systems with ride-sharing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3583–3590. ISSN: 2153-0866.

[5] Engelhardt, R., Dandl, F., and Bogenberger, K. Simulating ride-pooling services with pre-booking and on-demand customers.

[6] Andres Fielbaum, Maximilian Kronmueller, and Javier Alonso-Mora. Anticipatory routing methods for an on-demand ridepooling mobility system. *Transportation*, pages 1–42, 2021.

[7] Xiaoming Li, Chun Wang, Xiao Huang, and Yimin Nie. A data-driven dynamic stochastic programming framework for ride-sharing rebalancing problem under demand uncertainty. In *2020 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 1120–1125.

[8] OECD. Urban mobility system upgrade how shared self-driving cars could change city traffic.

[9] Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H. Strogatz, and Carlo Ratti. Quantifying the benefits of vehicle pooling with shareability networks. 111(37):13290–13294.

[10] Andrea Simonetto, Julien Monteil, and Claudio Gambella. Real-time city-scale ridesharing via linear assignment problems. 101:208–232.

[11] Matthew Tsao, Dejan Milojevic, Claudio Ruch, Mauro Salazar, Emilio Frazzoli, and Marco Pavone. Model predictive control of ride-sharing autonomous mobility-on-demand systems. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6665–6671. ISSN: 2577-087X.

B

Additional results

In this appendix, additional results are shown. The results are for the experiments without benefits for reservations nor heuristics; the value of computational time; for the effects of an increased rejection penalty; the tuning of the heuristic parameters. The experiments are done with the same settings as the corresponding ones in the thesis. The variable to review is the number of vehicles to use.

The first results are for the experiments without benefits for reservations and without the use of heuristics. The results below are for experiments with 1000 or 1250 vehicles.

		Time revealed in advance [minutes]	
		5	10
Random reservations	0%	0.863 - 261 1.02 98	0.863 - 261 1.02 98
	10%	0.877 0.95 252 1.03 207	0.886 0.952 268 1.04 899
Service rate (SR) Reservation service rate Average total delay [s] Distance ratio Simulation time	20%	0.882 0.941 149 1.05 424	.899 0.96 145 1.05 3258
	30%	0.9 0.948 240 1.07 752	0.912 0.963 236 1.07 6643
Rejections as reservations	13.7%	0.888 0.845 262 .04 223	0.902 0.882 262 1.05 1260

Table B.1: Performance with reservations for 1000 vehicles in case of no benefits for reservations and no heuristics applied.

		Time revealed in advance [minutes]	
		5	10
Random reservations	0%	0.928 - 250 0.98 109	0.928 - 250 0.98 109
	10%	0.939 0.975 241 0.97 184	 0.953 0.979 235
Service rate (SR) Reservation service rate Average total delay [s] Distance ratio Simulation time	20%	0.945 0.973 228 0.98 388	0.963 0.981 225 0.99 3473
	30%	0.951 0.981 226 1.01 755	0.978 0.993 206 1.01 7177
Rejections as reservations	7.2%	0.941 0.885 251 0.99 128	0.945 0.912 255 1 485

Table B.2: Performance with reservations for 1250 vehicles in case of no benefits for reservations and no heuristics applied.

The next result is for the test on available computational time, in this case with 1000 vehicles. The results are for 10% reservations, revealed 10 minutes in advance.

1000 vehicles	ILP time			
	5	**15**	**25**	
30 s 0.03s per vehicle	0.89 0.998 2302	0.894 0.999 2232	0.893 0.998 2285	service rate future service rate simulation time
75 s 0.075s per vehicle	0.89 0.998 3464	0.891 0.999 3450	0.892 0.999 3410	
150 s 0.15s per vehicle	0.892 0.997 4602	0.888 0.997 4486	0.89 0.999 4506	

Table B.3: Performance with different limits on computational time for 1000 vehicles.

Third, the results on the combined effects of an increased rejection penalty for reservations and the pledged service constraint applied to reservations or to all requests. The results are for 750 vehicles.

Rejection penalty = 5 750 vehicles	pledged service for reservations				pledged service for all requests			
Reservation penalty factor	**1**	**1.5**	**2**	**5**	**1**	**1.5**	**2**	**5**
service rate	0.731	0.729	0.731	0.732	0.708	0.706	0.703	0.702
reservation service rate	1	1	1	1	1	1	1	1
total delay time [s]	268	267	269	269	278	280	281	278

(a) Rejection penalty 5

Rejection penalty = 10 750 vehicles	pledged service for reservations				pledged service for all requests			
Reservation penalty factor	**1**	**1.5**	**2**	**5**	**1**	**1.5**	**2**	**5**
service rate	0.73	0.736	0.736	0.733	0.702	0.701	0.706	0.705
reservation service rate	1	1	1	1	1	1	1	1
total delay time [s]	265	270	270	265	283	283	278	280

(b) Rejection penalty 10

Table B.4: Effect of different rejection penalties with random reservations and the pledged service constraint active. Experiment with 750 vehicles.

Rejection penalty = 5 750 vehicles	pledged service for reservations				pledged service for all requests			
reservation penalty factor	**1**	**1.5**	**2**	**5**	**1**	**1.5**	**2**	**5**
service rate	0.726	0.725	0.724	0.725	0.7	0.693	0.698	0.686
reservation service rate	1	1	1	1	0.995	0.996	0.999	0.999
total delay time [s]	278	279	279	277	287	286	288	290
Distance ratio	1.09	1.1	1.09	1.09	1.06	1.05	1.05	1.04
RTV edges *1e6	7.17	7.2	7.26	7	6.98	7.19	7.43	7.52

(a) Rejection penalty 5

Rejection penalty = 10 750 vehicles	pledged service for reservations				pledged service for all requests			
reservation penalty factor	**1**	**1.5**	**2**	**5**	**1**	**1.5**	**2**	**5**
service rate	0.727	0.719	0.726	0.716	0.696	0.698	0.694	0.693
reservation service rate	1	1	1	1	0.993	0.997	0.999	0.998
total delay time [s]	279	277	277	279	289	287	289	286
Distance ratio	1.1	1.08	1.1	1.08	1.05	1.06	1.05	1.05
RTV edges *1e6	7.2	7.02	6.88	7.24	6.95	6.96	6.86	7.33

(b) Rejection penalty 10

Table B.5: Effect of different rejection penalties with rejections as reservations and the pledged service constraint active. Experiment with 750 vehicles.

Last, the tuning for the heuristics parameters is performed for 1000 and 1250 vehicles. First, the reference value without the application of heuristics is shown. Afterwards, the results for the different heuristics are shown. The most important takeaway is that trends are similar to a scenario with 750 vehicles.

Vehicles	**1000**	**1250**
service rate	0.86	0.948
reservation service rate	0.999	1
RTV build time [s][s]	1128	1298

Table B.6: Performance without heuristics in the reference scenario, 10% random reservations revealed 10 minutes in advance.

max vehicles per request	**1**	**10**	**20**	**40**	**60**
service rate	0.69	0.828	0.847	0.854	0.86
reservation service rate	0.864	0.985	0.999	0.997	0.999
RTV build time [s][s]	30	399	507	769	959

(a)

max vehicle per reservation	**1**	**10**	**20**	**40**	**60**
service rate	0.783	0.829	0.852	0.853	0.851
reservation service rate	0.809	0.989	0.996	0.999	0.998
RTV build time [s][s]	45	322	510	786	1079

(b)

Table B.7: Maximum vehicles per request/reservation, experiment with 1000 vehicles.

max vehicles per request	**1**	**10**	**20**	**40**	**60**
service rate	0.786	0.939	0.944	0.948	0.944
reservation service rate	0.927	0.998	1	1	1
RTV build time [s][s]	27	263	450	739	944

(a)

max vehicle per reservation	**1**	**10**	**20**	**40**	**60**
service rate	0.904	0.941	0.948	0.948	0.953
reservation service rate	0.887	0.998	1	1	1
RTV build time [s][s]	39	254	440	722	1080

(b)

Table B.8: Maximum vehicles per request/reservation, experiment with 1250 vehicles.

prune per layer	**40, 40**	**40,80**	**40, 120**	**80, 80**	**80, 120**	**120, 120**
service rate	0.803	0.81	0.806	0.848	0.842	0.856
reservation service rate	0.958	0.957	0.959	0.979	0.977	0.991
RTV build time [s][s]	422	673	712	792	884	911

(a) 1000 vehicles

prune per layer	**40, 40**	**40,80**	**40, 120**	**80, 80**	**80, 120**	**120, 120**
service rate	0.896	0.901	0.901	0.942	0.936	0.948
reservation service rate	0.979	0.983	0.982	0.984	0.985	0.999
RTV build time [s][s]	438	674	775	805	945	953

(b) 1250 vehicles

Table B.9: Pruning applied per trip size for sizes 1 and 2. Results for 1000 and 1250 vehicles.

Restricted to X best vehicles	1	10	20	40	60
service rate	0.81	0.813	0.826	0.844	0.845
reservation service rate	0.959	0.977	0.99	0.997	0.997
RTV build time [s][s]	35	115	262	508	989

(a) Available for all vehicles 5 minutes in advance

Restricted to X best vehicles	1	10	20	40	60
service rate	0.817	0.808	0.832	0.848	0.85
reservation service rate	0.95	0.978	0.994	0.997	0.999
RTV build time [s]	36	120	271	525	768

(b) Available for all vehicles once emerged

Restricted to X best vehicles	1	10	20	40	60
service rate	0.81	0.806	0.83	0.946	0.95
reservation service rate	0.964	0.976	0.992	1	1
RTV build time [s]	33	119	264	418	625

(c) Only available to selected vehicles

Table B.10: Restricted to X best vehicles heuristic, applied to 1000 vehicles.

Restricted to X best vehicles	1	10	20	40	60
service rate	0.916	0.925	0.937		
reservation service rate	0.983	0.995	0.999		
RTV build time [s]	33	78	177		

(a) Available for all vehicles 5 minutes in advance

Restricted to X best vehicles	1	10	20	40	60
service rate	0.922	0.928	0.941		
reservation service rate	0.988	0.993	1		
RTV build time [s]	33	78	184		

(b) Available for all vehicles once emerged

Restricted to X best vehicles	1	10	20	40	60
service rate	0.915	0.92	0.937		
reservation service rate	0.986	0.995	0.999		
RTV build time [s]	29	78	189		

(c) Only available to selected vehicles

Table B.11: Restricted to X best vehicles heuristic, applied to 1250 vehicles.

max trip size	2	3	4
service rate	0.85	0.855	0.855
reservation service rate	0.999	0.998	0.999
RTV build time [s]	24	459	1100

(a) 1000 vehicles

max trip size	2	3	4
service rate	0.952	0.948	0.947
reservation service rate	1	1	1
RTV build time [s]	28	597	1264

(b) 1250 vehicles

Table B.12: Maximum trip size with 1000 or 1250 vehicles.