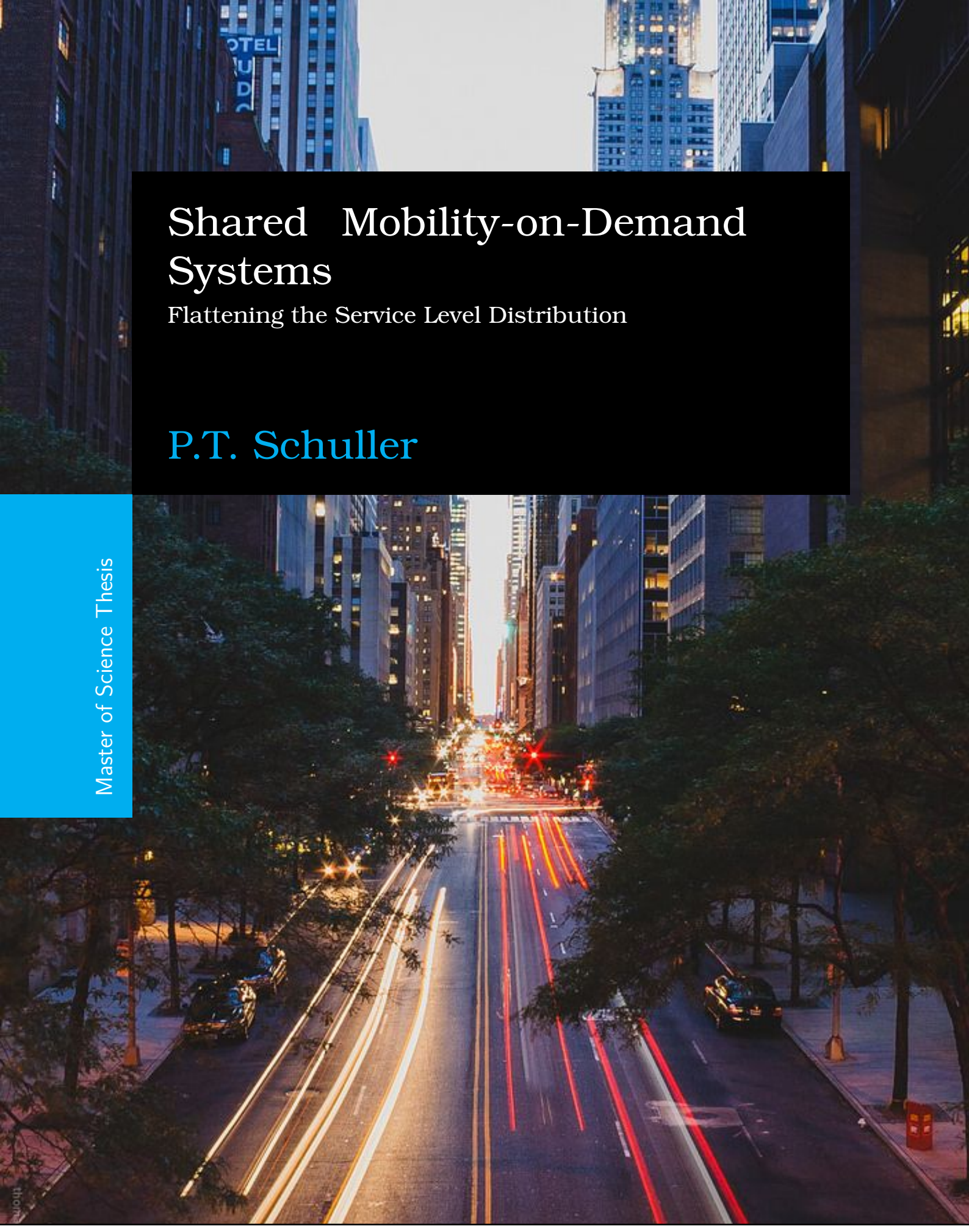


Shared Mobility-on-Demand Systems

Flattening the Service Level Distribution

P.T. Schuller

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Shared Mobility-on-Demand Systems

Flattening the Service Level Distribution

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

P.T. Schuller

February 16, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

In recent years, Shared Mobility-on-Demand systems have emerged as a great method for door-to-door transportation. Studies have shown that it is possible to route vehicles and assign requests to vehicles efficiently in large-scale systems. These studies commonly report one-dimensional performance metrics such as average vehicle occupancy, service rate, or average waiting time. We repeated a case study using a state-of-the-art Fleet Management Framework [2] and focused on the distribution of the service level over the operation area. We observed that the chance of receiving service in a low demand area was much higher than in a high demand area. Going from this observation, this research's objective was to research how the state-of-the-art framework can be adjusted such that the rejection rates are more evenly spread over the operation area. We developed different methods that adjust the decision of which mobility requests are serviced or which trips are selected. The methods work such that a request located in an above-average rejection rate area has an increased chance of being serviced. Similarly, a trip that goes through an area of above-average rejection rate also has priority. We set up a Discrete Event Simulation that simulates a Shared Mobility-on-Demand system to research the effects of our added method compared to the original framework. We simulated an artificial city and New York City. The Gini index was used to measure how evenly the rejection rates were spread over the operation area. In many cases, our methods were able to lower both the average rejection rate and the Gini index. With this work, we showed that the state-of-the-art framework's objective can be extended to a broader goal. This opens up new possibilities to tune the system to match specific transportation needs in different areas of a city.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cities worldwide have been struggling with congested roads due to an ever-growing urban population [41] and more affordable private door-to-door transportation [37]. An illustrative example of this problem is the falling average traffic speed of New York City; it dropped from 10.5 km/h in 2012 down to 7.6 km/h in 2017 [19]. In the US, urban congestion cost travellers a total of 8.8 billion wasted hours stuck in traffic and 3.3 billion gallons of fuel, adding up to a total cost of 166 billion dollars [39]. Increasing the number of people per vehicle has always been one of the main objectives when trying to solve congestion. Traditionally, solutions such as carpooling or the use of public transport have been promoted for this purpose. Due to the rise of the internet and the sharing economy, a new solution has emerged: Shared Mobility-on-Demand (SMoD) systems. An SMoD system consists of a fleet of vehicles that continuously drive around. Vehicles in an SMoD system can be either autonomously or manually driven. Customers can hail these vehicles via an electronic device (e.g. a smartphone) by sending a mobility request. This request contains the desired pickup and drop-off location, and the desired pickup time, which is usually the current time. If possible, a vehicle is dispatched to pick up the customer and drop them off at their desired location. The main difference between an SMoD system and a regular e-hailing taxi system, e.g. Uber[1] or Lyft[2], is ride-sharing. In an SMoD system, customers simultaneously share a vehicle with other customers that happen to go in roughly the same direction. An SMoD system is not the same as carpooling; the drivers continuously ride around for profit.

Operating a large-scale SMoD system in a sizeable city is a daunting task. Every minute, thousands of vehicles need to be routed towards hundreds of requests. Recently, studies have shown that it is possible to route vehicles in such systems efficiently. In [29], it is shown for the city of Beijing, that an SMoD system can serve an additional 25% of customers compared to a regular taxi system. Similarly, the authors of [2] show that, when taxi trips are efficiently shared, 23% of the current New York taxi fleet would be enough to serve 98% of taxi requests. These studies present efficient algorithms. However, the algorithms are not suitable to steer the system towards certain behaviour. Different stakeholders of an SMoD system might desire

[1] https://www.uber.com/
[2] https://www.lyft.com/

different behaviour of the system. The operator of the system could be mainly focused on maximizing profits; in that case, the operator would care about serving as many passengers and reduce operating costs as much as possible. On the other hand, the city council might have different objectives. They might desire an equal service level at any location within a city. Alternatively, the objective could be to increase the service level at train stations to promote public transport. When SMoD systems are used in cities worldwide, it would be beneficial if the touch of a button could achieve such behaviour.

The motivation for the writing of this thesis comes from further analyzing the results of the state-of-the-art Fleet Management Framework (FMF) of [2]. Throughout this thesis, the term FMF is used to describe an algorithm that routes vehicles and assigns requests to vehicles. In [2], the authors perform a case study for the city of New York. The authors mainly report metrics such as vehicle occupancy, service rate, travel delay, or total vehicle mileage. All useful metrics, however, these one-dimensional metrics do not reveal how the system is functioning in different geographical areas. We repeated their case study and focused our attention on the distribution of these metrics over the operation area. We decided to specifically focus on the locations of rejections in the system. We found that the rejections in our simulation are not spread out evenly. The chance of getting rejected in a high demand area is much larger than the chance of getting rejected in a low demand area. When using the FMF of [2] on artificial scenarios, we also found that the location of a node influences the height of the rejection rate at that node. Even when demand is uniformly distributed over the operation area, the outer areas' nodes usually have a higher rejection rate. The need for more flexible algorithms, and the unevenly spread out the service rate of the state-of-the-art FMF of [2] provided us with the motivation to write this thesis.

1-1 Research Question

we research whether the state-of-the-art FMF of [2] can be adapted to achieve a certain desired behaviour. More specifically, we investigate whether it is possible to achieve equal service levels over the entire operating area. What would be the best way to achieve this? How can be dealt with the increased flexibility? How would one even define equal service? Would pursuing even service levels worsen other performance criteria of the system? Or, would balancing the rejection rates over the operation area increase the performance of the system? The goal of this thesis is to provide answers to these questions.

1-2 Methods

We perform our research by developing different techniques that are added to the FMF of [2] to obtain more evenly spread out rejection rates. The reason we choose to use this specific algorithm of [2] is twofold. First, after doing a literature review, we found that the algorithm is currently state-of-the-art. The algorithm is efficient in assigning requests to vehicles and can be tuned intuitively. The latter brings us to the second reason: the algorithm's intuitiveness allows us to add extensions to the algorithm without losing the understanding of how the algorithm works.

Operating a large-scale SMoD system is a highly complex problem. Solutions generally consist of multiple consecutive steps. Therefore, doing this research analytically or theoretically is not tractable. Implementing a solution and simulating the outcome is the only viable option. By doing so, we gather insights into whether our methods achieve the desired results. The Gini index, a measure for statistical dispersion, is used to quantify how evenly spread out the service rate is. We compare the results of our various techniques against the original framework of [2].

We choose to simulate an SMoD system for New York City because of the available data on mobility demand and the road network. Furthermore, as an additional advantage, we can use the knowledge obtained in the work of [2]. We can make use of the same parameter settings of the original algorithm. To model New York in our case study, we use a street network model obtained from [13]. Here, the city's street network is modelled as a weighted graph, where edges represent street sections and nodes represent intersections. The weights on the edges represent the corresponding travel times. Furthermore, we simulate real-life mobility using a publicly available data set containing all trips made by yellow taxi cabs in 2014 [31].

1-3 Contribution

We found that our added techniques for the original framework helped reduce both the Gini index and the rejection rate in a case study for an artificial scenario. Furthermore, we show that our method can consistently achieve a lower Gini index if we allow for a slight increase in rejection rate compared to the original framework results. In the case study using artificial scenarios, we also show that our methods can be tuned such that the focus either lies on decreasing the Gini index or the rejection rate. Lastly, in a case study for New York, we show that our method can reduce the Gini index without increasing the rejection rate.

With this thesis, we show that the framework of [2] can be extended to serve a broader goal. A complex system such an SMoD should not aim at optimizing one-dimensional criteria. A complete view is given by also looking at the system's performance over the operation area. By doing so, the mathematical optimal operation mode is intertwined with the practically preferred one. One region of the operation area cannot suffer from another region. Throughout this thesis, we aimed at minimizing the difference in service level over the operation area as we thought this to be the most socially desired. One can extend our line of thought to other criteria such as, for instance, waiting time. Extending the objectives of these systems ultimately result in a better fit between a city and its SMoD.

1-4 Thesis Outline

First, in Chapter 2, we provide an overview of the literature on SMoD systems and related research areas. Furthermore, we explain the preliminary research upon which this thesis is built. Second, Chapter 3 elaborates on the motivation for the writing of this thesis. Third, in Chapter 4, we explain the various methods that we used. Fourth, Chapter 5 explains how we simulate an SMoD system and how we gathered the required data. In Chapter 6 and

Chapter 7, we present, respectively, the results of our artificial experiments and our real-life case study. Lastly, in Chapter 8, we give our conclusions and present future research directions.

Chapter 2

Literature Review

This chapter first explains our definition of a Shared Mobility-on-Demand (SMoD) system and shows what variants can be found in the literature. Secondly, we draw parallels between the well-known Vehicle Routing Problem (VRP) and its variants, and a SMoD system. Thirdly, we give an overview of common solutions for controlling a SMoD system. Lastly, we explain the preliminaries for this thesis.

2-1 Description a Shared Mobility-on-Demand System

We define an SMoD system as a centrally controlled e-hailing mobility service in which passengers share vehicles simultaneously. The system consists of a fleet of vehicles (autonomous or manually driven) and a control centre. There is perfect knowledge of all vehicles' states (their locations, their planned routes, and their passengers) at the control centre. Customers can send mobility requests to this control centre using an electronic device. A request contains the desired pickup and drop-off location, and the request time. At the control centre, there has to be decided whether to accept or reject the request. If the request gets accepted, a vehicle needs to be dispatched to pick up the passenger. This makes the service on-demand; if possible, a vehicle is dispatched immediately after the customer sends the request. We use the formulation as given in [2] to formally describe the problem of controlling the fleet of vehicles in an SMoD system. We adjust this formulation to make it more general. The adapted formulation is as follows:

Consider a fleet $\mathcal{V}$ of m vehicles of capacity ν. The objective is to assign online travel requests $\mathcal{R}$ to vehicles in the set $\mathcal{V}$ and to route the vehicles so that an objective function C is optimized under a set of constraints Z.

In the next subsections, we explain what these objectives and constraints entail, and we show which are commonly use in the literature on SMoD systems.

2-1-1 Constraints

We assume that three main constraints are always present in the set of constraints Z: 1) customers have to be picked up within a certain amount of time after the trip request was sent; 2) the amount of detour per passenger is limited; and 3) the capacity of a vehicle may not be exceeded at any time. Additional constraints could be added to the system (see Table 2-1); we only consider the three main constraints in this thesis.

Constraint	Description	Ref.
Monetary	A passenger does not pay more than without ride-sharing; a taxi driver does not earn less than without ride-sharing when travelling the same distance.	[30]
Stops	The number of stops, due to ride-sharing, is constrained for each passenger.	[32]
Shared trips	The number of shared-trips for taxis is constrained.	[32]
Fuel	A vehicle can only operate for a certain amount of time.	[22]
Heterogeneity	Passengers have different preferences: their willingness to carpool or sensitivity to ride with a specific gender differs.	[27]

Table 2-1: Overview of extra constraints found in the literature that are sometimes imposed on SMoD systems.

2-1-2 Objective Function

The operator of the SMoD system is generally interested in of two performance metrics: 1) Quality of Service (QoS), customers desire the least amount of discomfort and the fastest travel times; and 2) Operation Cost (OC), the organization wants to minimize the total distance travelled by all vehicles or to maximize the profit [7]. These metrics are generally in conflict with each other, optimizing one means failing the other. Therefore, the objective functions found in the literature are often a combination of both performance metrics. The goal is then to find a Pareto optimal solution. Table 2-2 gives an overview of different objective functions found in the literature.

	Obj.	[14]	[24]	[29]	[30]	[18]	[42]	[22]	[21]	[23]	[32]	[2]	[9]	[16]	[5]	[44]	[27]	[8]	[33]	[40]
QoS	↓ Wait. time	✓	✓				✓✗		✗			✓						✓	✓	✓
	↓ Ride time	✓	✓				✗		✗			✓		✓				✓		✓
	↓ Detour						❙							✓	✓					
	↑ Requests								✓✗			✓	✓						✓	
OC	↑ Profit					✓			✓							✓	✓			✓
	↓ Distance	✓	✓	✓	✓			✓		✓	✓		✓							

Table 2-2: Overview of different objectives found in the literature on SMoD systems. To explain the notation: [14] has the objective to optimize waiting time, ride time and total distance at the same time. [42] proposed multiple cost functions: 1) optimising the waiting time; 2) optimising both waiting time and ride time; and 3) optimising detour length.

2-2 From Vehicle Routing Problem to Shared Mobility-on-Demand System

The problem of controlling a fleet of vehicles in an SMoD system can be seen as a broad generalization of the Vehicle Routing Problem (VRP) as made famous in [12]. The original VRP was used to describe the problem of finding optimal routing for a fleet of gasoline delivery trucks (of limited capacity) between a bulk terminal and multiple service stations. In order to match real-life optimization problems more closely, many extensions to the VRP have been proposed. In the remainder of this subsection, we briefly describe different variations of the VRP and draw parallels to the problem of controlling a SMoD system.

Dynamic The classic VRP assumes that all demand at all locations is known beforehand (static problem). However, this assumption is often unreasonable in real-life problems. Dynamic problems are also common; new demand is revealed while vehicles are already driving around. The degree of dynamism quantifies how close a problem is to being entirely dynamic. This measure is commonly defined as the ratio between dynamic requests and the total number of requests at the end of a transportation process [1]. Figure 2-1 shows a graphical example of a Dynamic VRP. Note the inherent difficulty of Dynamic VRP's; if the truck had been scheduled to drive the route clockwise, it would have had to drive many extra kilometres to serve the two dynamic customers.

Different solution approaches have been proposed for dynamic VRPs, ranging from linear programming to meta-heuristics, depending on the size and type of a problem. SMoD systems can be classified as dynamic since requests are usually put out right before the desired pickup time. Moreover, very few requests are known to the system at the start of a planning horizon. Therefore, the degree of dynamism is high (close to 1); this makes controlling these systems extremely complex.

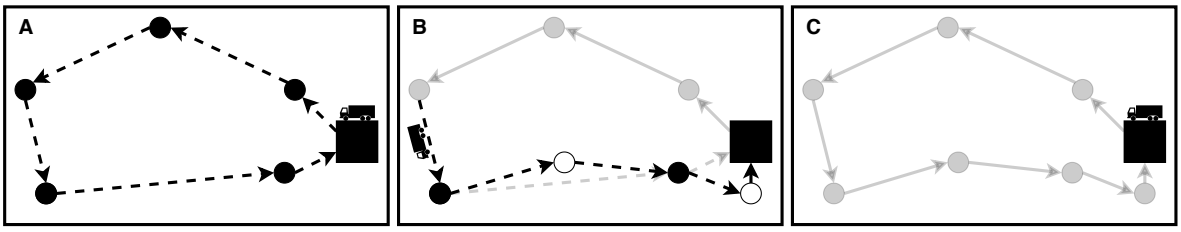


Figure 2-1: Illustration of the Dynamic VRP with one truck. In tile A the truck is at the depot and five customers (black dots) revealed their demand, the dashed lines form the optimal, intended, truck schedule. In tile B the truck has served three customers and is en route to the fourth, at the meantime two new customers reveal their demand and the truck is rescheduled accordingly. In tile C all customers have been served.

Stochastic More often then not, a real-life VRP is not deterministic. For instance, travel times could be stochastic, or the size of a customer's order could be stochastic Figure 2-2 shows the three different types of stochasticity encountered in stochastic VRPs, note that any combination of the different types can also occur. Finding good solutions to these types of problems is difficult. Just using averages to solve the problem as a regular VRP results

in arbitrary bad solutions [4]. A real-life SMoD system also has stochastic components; for example, travel times of vehicles are stochastic as well as requests that customers put out. There has to be decided whether to take all types of stochasticity into account. For instance, taking stochastic travel times into account might complicate the problem more than it helps, on the other hand, using historical demand data to predict new demand might prove beneficial. Solving stochastic models is generally computationally expensive, so, the trade-off between solution quality and running time needs to be considered.

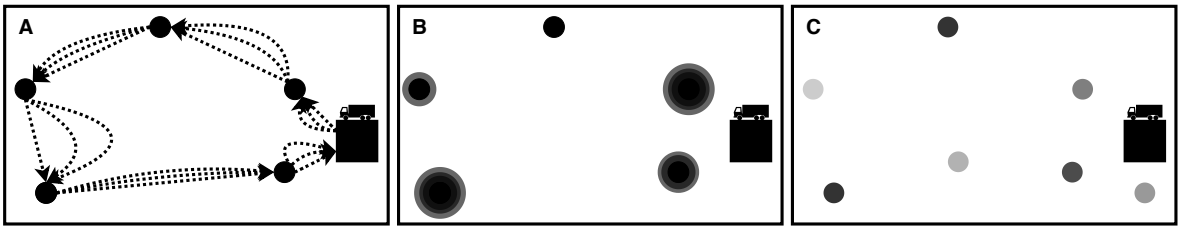
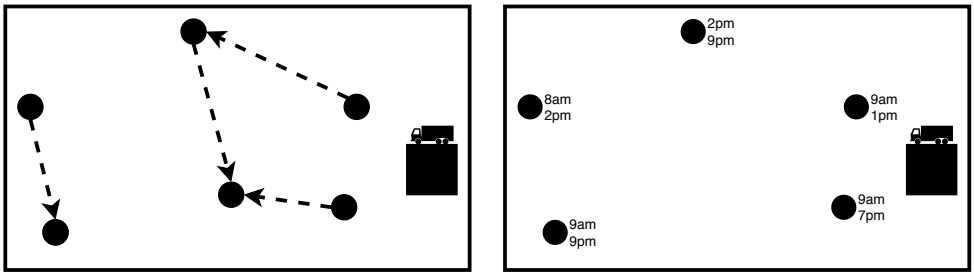


Figure 2-2: Illustration of different uncertainties in the stochastic VRP. In tile A the travel times are uncertain, in tile B the size of demand per customer is uncertain (larger dots represent more demand) and in tile C customers are uncertain (more faded dots represent lower probabilities).

Pickup and Delivery Door-to-door transportation of elderly people is the most common example of a VRP with pickup and delivery. Figure 2-3(a) shows an illustration of such a problem. In these problems, goods/people have to be transported from one place to another. Solutions methods to these types of problems can be exact [15, 10] or heuristic-based [11], depending on the size of the problem. SMoD systems are quite similar to these type of problems. Therefore, solutions methods can be used for inspiration. For instance, a common technique when solving VRPs with pickup and delivery is using insertion methods. The same insertion methods could be used when re-routing a vehicle in an SMoD.

Time Windows The last extension we would like to discuss is time windows. In real-life VRPs it cannot be assumed that a vehicle can arrive at any time. Usually, a vehicle should arrive within a desired time window. Figure 2-3(b) shows an illustration of such a problem. Heuristics are often used due to the complexity of VRPs with time windows [6]. SMoD systems make use of time windows as the operator cannot allow a customer to wait indefinitely. Therefore, the heuristics developed in the literature on VRPs with time windows could be used to control a SMoD system.



(a) Pickup and delivery (b) Time windows

Figure 2-3: Graphical representation of a pickup and delivery problem (a), and a VRP with time windows (b).

2-3 Control Approaches

Throughout the literature, several different solution approaches have been proposed to control a fleet of vehicles in an SMoD. In this subsection, we organise the different approaches in a framework and explain them. In Figure 2-4, the different works we found in the literature are organised in a diagram showing the different approaches.

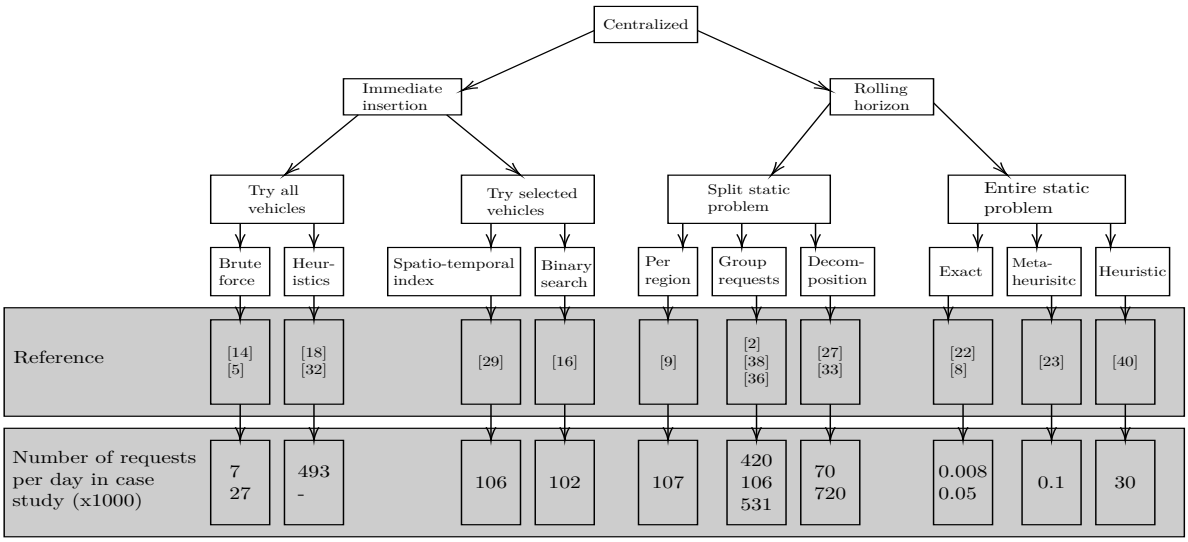


Figure 2-4: Overview of different Fleet Management Frameworks found in the literature.

2-3-1 Immediate Insertion

The first general control approach we discuss are the immediate insertion methods. The idea of these methods is to insert a new request in the vehicle's schedule as soon as it is received. This insertion can be tried on a subset of vehicles or the entire fleet. Furthermore, when trying the insertion on a vehicle, it can be done using heuristics or brute force. The immediate insertion approach is often justified with the following arguments: 1) no pre-booking, the system has only knowledge of current travel requests, therefore optimization can hardly happen on a

global scope [29]; 2) optimization for a fleet of vehicles in NP-hard [29]; and 3) in practice reordering of optimal schedules rarely occurs due to tight constraints [16]. The immediate insertion approach has the advantage of being computationally inexpensive since only one request is taken into account per optimization. Furthermore, many computations can be done in parallel (e.g. trying insertions on different vehicles). However, a major disadvantage of this method is that potentially sub-optimal decisions can be made because optimization is not performed globally.

2-3-2 Rolling Horizon

The second common solution approach is to use a rolling horizon framework. When using this strategy, requests are batched when they come in. Then, the batch of requests is assigned to vehicles at ones. This static problem that needs to be solved is generally complex. Therefore, different solution methods have been proposed that can solve different sizes of static problems. First, there is the idea of splitting the static problem into smaller problems. This can be done based on geographical regions, by clustering/grouping vehicles or requests, or by decomposing the static problem. Alternatively, the static problem could be solved as one problem. In that case, one could use exact methods or (meta-) heuristics. Solving the entire static problem at once in generally intractable for large SMoD systems. If the rolling horizon approach is used one should carefully choose the horizon length: if the horizon is too short sub-optimal decisions can be made, however, if the horizon is too long the model might differ from the real system too much. In the latter case, customers have to wait too long until they are assigned to a vehicle.

2-4 Preliminaries

In this section we discuss three important preliminaries for our research: 1) the state-of-the-art fleet management framework from [2], 2) a clustering method to cluster a road network into regions [43], and 3) the Gini index [17].

2-4-1 State-of-the-art Fleet Management Framework

In this section, we explain the state-of-the-art FMF as proposed in [2]. This algorithm can manage a fleet of vehicles in large-scale SMoD systems. The main advantage of this framework is its modularity; it can be extended with additional constraints, the objective function can be altered, or a different rebalancing algorithm can be used.

The method maintains a pool of requests that is constantly updated. Requests are added to the pool when they are received. Requests are deleted from the pool when they are picked up or rejected. Then, given this pool of requests, the following objective function is minimized in each iteration:

$$C(\Sigma) = \sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{P}_v} \delta_r + \sum_{r \in \mathcal{R}_{ok}} \delta_r + \sum_{r \in \mathcal{R}_{ko}} c_{ko}, \tag{2-1}$$

with $\mathcal{V}$ the fleet of m vehicles of capacity ν, $\mathcal{P}_v$ the set of passengers (picked-up requests) for vehicle $v \in \mathcal{V}$, δ_r the total travel delay for requests r, $\mathcal{R}_{ok}$ the set of requests that have

been assigned to some vehicle, $\mathcal{R}_{ko}$ the set of unassigned requests, and c_{ko} a large constant. The delay of a request is defined as the waiting time (pickup time minus request time) plus the detour time (time spent in-vehicle minus travel time of shortest path). The objective of (2-1) is to minimize the sum of delays and the number of rejected requests. This optimization happens under the constraints that:

1. for each request, the total travel delay δ_r is lower than the maximum travel delay Δ,
2. the waiting time ω_r is below a maximum waiting time Ω,
3. for each vehicle v, the number of passengers cannot exceed its capacity τ at any time.

Searching for an exact solution to the entire static problem is computationally intractable. Therefore, the authors of [2] proposed to split the optimization into four consecutive steps. In the coming four paragraphs, we go over these four steps. Consecutive iterations of the entire optimization procedure are linked by the request pool that is maintained.

1. Request-Vehicle-Graph The first step is to build a request-vehicle-graph (RV-graph); this is an extension of the shareability graphs as proposed in [34]. It represents which requests can be pairwise shared and which vehicles can serve which requests. In this graph, an arc exists between requests r_1 and r_2 if these requests can be both serviced by a single vehicle. In other words, an empty vehicle starting at the origin of one of these requests can service both requests within the constraints. Similarly, an arc exists between a request r and a vehicle v, if the vehicle can service the request r without violating any constraints (taking into account the current passengers of v). The arcs between requests and vehicles require feasibility checks that might be computationally exhaustive. Therefore the authors of [2] propose to use heuristics for larger vehicles.

2. Request-Trip-Vehicle-Graph The second step is to build a request-trip-vehicle-graph (RTV-graph). This graph is computed by exploring the cliques (complete sub-graphs) of the RV-graph. This is done by computing feasible trips, for each vehicle, incrementally in trip size. A trip is defined as a set of requests that can be combined to be serviced by one vehicle without violating any constraints. Two types of arcs exist in the RTV-graph. First, an arc between a request r and a trip T exists if T contains r. Second, an arc between a vehicle v and a trip T exists if v can service T without violating any constraints. These arcs have weights that are the associated sum of delays when vehicle v would perform trip T.

3. Assignment The third step is to assign trips to vehicles; this is done by formulating an ILP. In this step it is made sure that each request is assigned to at most one vehicle, and each vehicle is assigned to at most one trip. By solving the ILP, arcs are chosen within the RTV-graph such that the sum of delays associated with these arcs plus penalties associated with ignoring requests is minimized. The authors of [2] introduce the binary variable ϵ_{ij} for each edge $e(T_i, v_j)$ in the RTV-graph, $\epsilon_{ij} = 1$ entails that vehicle v_j is assigned to trip T_i. Furthermore, a binary variable χ_k is introduced for each request r, $\chi_k = 1$ entails that request r_k is ignored. Selecting an edge $e(T_i, v_j)$ has a cost of c_{ij}, which is the associated sum of delays of that edge.

$$
\begin{aligned}
&\text{Initial guess} = \Sigma_{greedy} \\
&\Sigma_{optim} = \arg\min_{\chi} \sum_{i,j \in \mathcal{E}_{TV}} c_{ij}\epsilon_{ij} + \sum_{k \in \{1,\dots,n\}} c_{ko}\chi_k \\
&\quad \text{s.t.} \sum_{i \in \mathcal{I}^T_{V=j}} \epsilon_{ij} \leq 1 && \forall v_j \in \mathcal{V} \\
&\quad \sum_{i \in \mathcal{I}^T_{R=k}} \sum_{j \in \mathcal{I}^V_{T=i}} \epsilon_{ij} + \chi_k = 1 && \forall r_k \in \mathcal{R}
\end{aligned} \tag{2-2}
$$

With $\mathcal{I}^T_{V=j}$ the set of trips that can be serviced by a vehicle j, $\mathcal{I}^T_{R=k}$ the set of trips that contain request k, and $\mathcal{I}^V_{T=i}$ the set of vehicles that can service trip i. The greedy initial guess is made by iteratively assigning vehicles to trips that have been sorted in decreasing size and increasing cost. The first constraint ensures that each vehicle is assigned to at most one trip. The second constraint ensures that each request is assigned to at most one vehicle or is ignored.

4. Rebalancing After the third step, not all requests are assigned to a vehicle, and some vehicles remain idle. In the fourth step, the idle vehicles are rebalanced towards ignored requests. This is achieved by solving an ILP that assigns idle vehicles to ignored requests, with the constraint that all idle vehicles get assigned to an ignored request or all ignored requests get assigned an idle vehicle. The objective is to minimize the total travel time of the idle vehicles. The ILP is as follows:

$$
\begin{aligned}
&\Sigma_{rebalance} = \arg\min_{y} \sum_{v \in \mathcal{V}_{idle}} \sum_{r \in \mathcal{R}_{ko}} \tau_{vr} y_{vr} \\
&\quad \text{s.t.} \sum_{v \in \mathcal{V}_{idle}} \sum_{r \in \mathcal{R}_{ko}} y_{vr} = \min\left(|\mathcal{V}_{idle}|, |\mathcal{R}_{ko}|\right) \\
&\quad \sum_{v \in \mathcal{V}_{idle}} y_{vr} \leq 1 && \forall r \in \mathcal{R}_{ko} \\
&\quad \sum_{r \in \mathcal{R}_{ko}} y_{vr} \leq 1 && \forall v \in \mathcal{V}_{idle} \\
&\quad y_{vr} \in \mathbb{B} && \forall v \in \mathcal{V}_{idle}, \forall r \in \mathcal{R}_{ko}.
\end{aligned} \tag{2-3}
$$

With $\mathcal{R}_{ko}$ the set of unassigned requests and $\mathcal{V}_{idle}$ the set of idle vehicles. The binary variable $y_{vr} = 1$ if idle vehicle v should rebalance towards ignored request r, τ_{vr} is the associated travel time of this rebalancing action. The constraint matrix of this ILP is unimodular, therefore, relaxing the binary constraint and solving the resulting LP results in an integer solution.

Extensions

The FMF of [2] has been extended in several works. In [43], the authors propose a rebalancing algorithm that estimates real-time demand per region and assigns idle vehicles to the regions using a linear program. A different rebalancing algorithm is proposed in [25]; here the authors

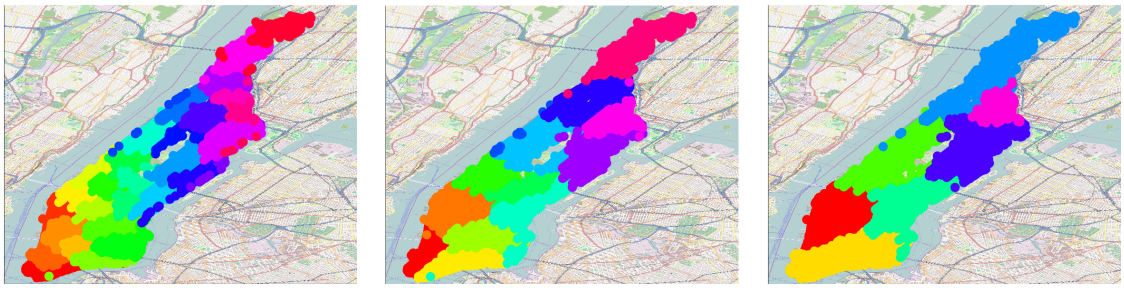
propose to add predicted requests to the pool of ignored requests such that idle vehicles make progress towards them. The authors of [3] propose to make the FMF proactive by including predicted future requests in the request pool. Similarly, the authors of [20] propose balancing the current fleet distribution with the (estimated) future demand distribution by introducing a penalty term in the assignment step.

2-4-2 Clustering Nodes into Regions

In this section, we will explain the clustering method as proposed in [43]. We explain this method because we sometimes use average metrics from regions instead of individual metrics from nodes in this thesis. The reason being that average metrics fluctuate less than individual ones. The method of [43] is an intuitive way to cluster a road network into regions. The only parameter that needs to be decided a priori for this method is t_{max}, which is defined as the maximum travel time between any node in the graph and the cluster's region centre. The method aims at minimizing the number of clusters given this parameter t_{max}. The input for this method is a graph $G(V, E)$ representing a road network and a matrix T where T_{ij} is the travel time from node i to node j. The first step is to create a reachability matrix R, where $R_{ij} = 1$ if $T_{ij} \leq t_{max}$ and $R_{ij} = 0$ otherwise. Then, a binary variable z is introduced, where $z_i = 1$ if node i is a region centre and 0 otherwise. The following ILP is formulated:

$$\begin{aligned} \min_{x} \quad & \sum_{i=1}^{|V|} z_i \\ \text{s.t} \quad & z_i \cdot R_{ij} \geq 1 \quad \forall j \in [1, |V|]\,. \end{aligned} \tag{2-4}$$

The objective function is to minimize the number of regions. The constraint ensures that each node in the network is within t_{max} minutes of at least one region centre. After solving the ILP, each node is assigned to the region centre its closest to. Unlike other clustering methods, this method provides a clear intuition behind the chosen clusters' size and distribution. In Figure 2-5, regions are shown, for the city of New York, for three different values for t_{max}.



(a) $t_{max} = 5$ min, 32 regions (b) $t_{max} = 7.5$ min, 13 regions (c) $t_{max} = 10$ min, 7 regions

Figure 2-5: Regions obtained using clustering method of [43] for different values of t_{max}.

2-4-3 Gini Index

In this section, we explain the concept of the Gini Index. Throughout this work, we use the Gini index as a measure for how equally distributed the rejection rates are. The Gini index, as first proposed in [17], is a measure for statistical dispersion. It is commonly used to measure income inequality in a country. A Gini index equal to 1 means perfect inequality; one person has an income, and all the other people have no income. On the other hand, a Gini index of 0 means perfect equality; everybody earns the same income. Commonly, the Gini index is calculated based on the Lorenz curve [26]. This curve graphically shows the portion of total income earned by the bottom $x\%$ (see Figure 2-6). The Gini index is the ratio of the area between the line of equality and the Lorenz curve (area A), and the area underneath the Lorenz curve (area B).

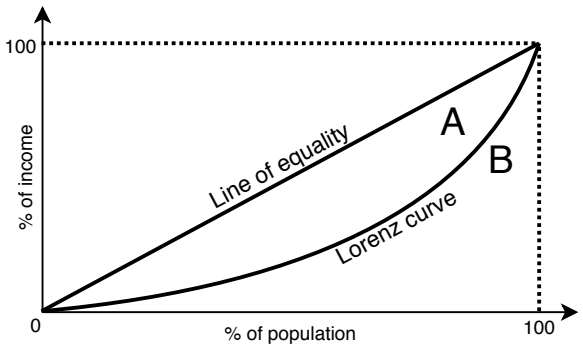


Figure 2-6: Visualisation of the Lorenz curve. The Gini index is defined as the area of A divided by the total area of A and B.

Alternatively, the Gini index can be calculated as half of the relative mean absolute difference:

$$\text{Gini index} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}|x_i - x_j|}{2\sum_{i=1}^{n}\sum_{j=1}^{n}x_j} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}|x_i - x_j|}{2n\sum_{j=1}^{n}x_j}, \tag{2-5}$$

where x_i is the income of person i. This is mathematically equivalent to the definition based on the Lorenz curve [35]. In this thesis, we use the Gini index to measure the equality of the rejection rates. Throughout this thesis, we will not compare Gini indexes resulting from different experiments. This is the case because the Gini index dependeds on the number of nodes and the number of requests/rejections in an experiment. Therefore, it is not useful to compare across experiments. However, within the same experiment setup, comparing Gini indexes is useful. In that case, we can conclude which algorithm achieved a fairer system (lower Gini index).

Chapter 3

Motivation

This research originated from further studying the experiments using the state-of-the-art Fleet Management Framework of [2] (henceforward referred to as the original FMF). We repeated the case study for the Manhattan area, and we ran simple experiments using artificial road networks. In this chapter, we will explain how these observations lead to our research idea. First, we will analyze the results of artificial experiments. Second, we will explain the results of the Manhattan case study. Lastly, we will combine the observations and explain how we intend to improve SMoD systems by taking them into account.

3-1 Artificial Experiments

The first experiments we performed were using the artificial scenarios as described in Chapter 5. This experiment's objective is to compare the distribution of the realized rejection rates with the demand pattern. We simulated the SMoD, using the original FMF, for longer periods to obtain a steady-state. This steady-state gives us a clear image of where in the network the most rejections take place. We selected the number of vehicles (65) and requests (5 per minute) such that we achieved about a 10% rejection rate. Figure 3-2 contains heatmaps showing the rejection rate at each node after running the experiment. The brighter the pixel, the higher the rejection rate at that node.

We observe two things from looking at Figure 3-1. First, the rejection rates seem to be highest in areas with high demand. For instance, for all versions of the 10L2R experiments, the left half of the network has a higher rejection rate than the right half. Similar observations can be made for the C2S experiments. Second, the rejection rates seem to be higher around the edges/corners of the road network. For instance, in the RAND experiments with all two-way streets, the highest rejection rates appear on the corners of the road network. Also, in the 10L2R-experiments, the nodes on the far left of the operation area have higher rejection rates than those just left of the middle. We conclude that both the demand pattern and the road network layout influence the rejection rate distribution.

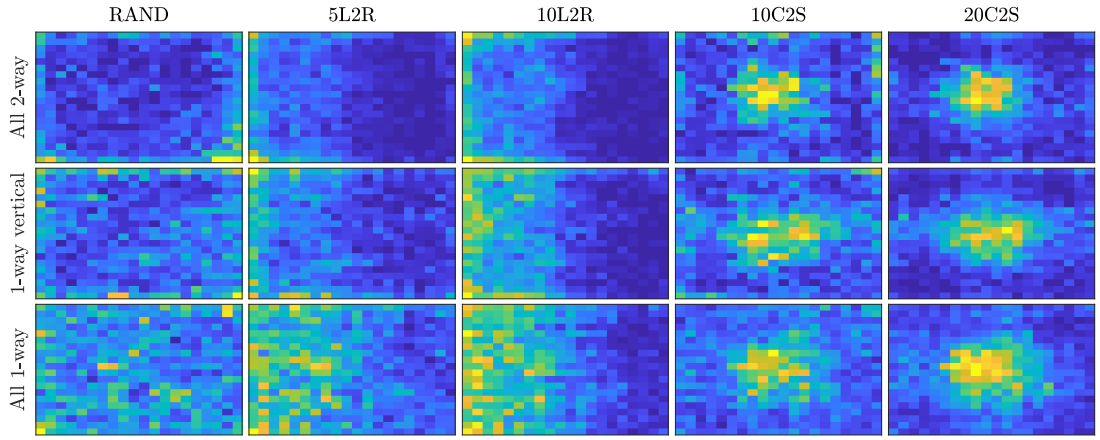
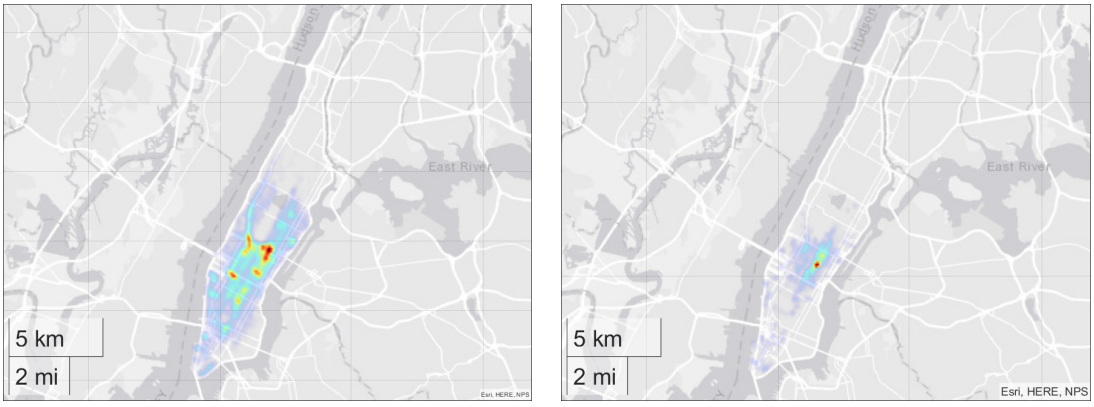


Figure 3-1: Heatmaps of rejection rates after running original FMF for fifteen different artificial scenarios (3 road networks combined with 5 demand patterns) on a 20-by-20 grid. The brighter a pixel, the higher the rejection rate at the corresponding node. In all experiments, 65 vehicles were used and 5 requests were generated per minute.

3-2 New York Case Study

In the previous section, we have seen that both the demand pattern and the road network layout influence rejections' locations when using the original FMF in artificial experiments. Now, we want to research whether the same conclusions can be drawn for a real-life SMoD system. Therefore, we set up a large-scale SMoD simulation for the Manhattan area and recorded the locations of all the rejections in the system. Figure 3-2(a) shows a heatmap of the origins of demand that occurred in the simulation, and Figure 3-2(b) shows a heatmap of the rejection rates per node.



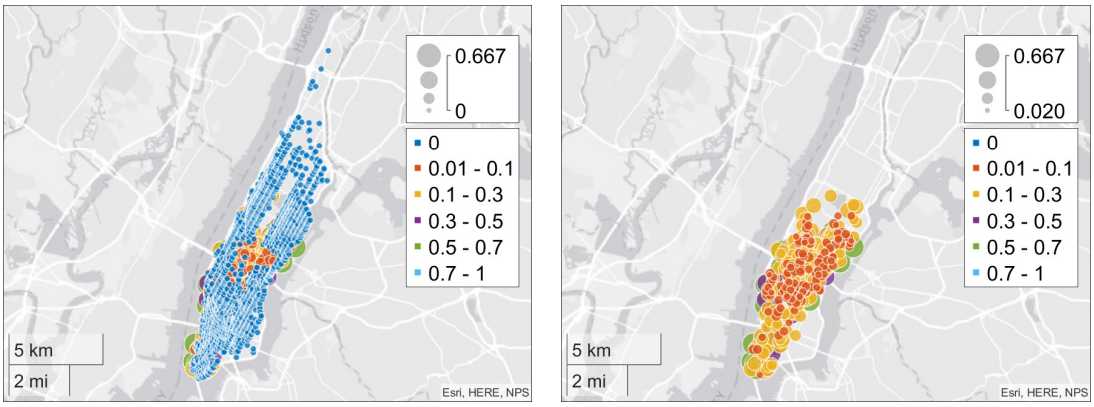
(a) Request origins (b) Rejection rates

Figure 3-2: Heatmaps of request origins and rejection locations for large-scale Manhattan case study using the original FMF. The redder an area, the higher the density.

From Figure 3-2(a), we conclude that the most requests occur in the centre and southern parts of Manhattan. We notice a clear hot spot of requests around the intersection of Park

Avenue and 57^{th} street; this makes sense as this is one of the busiest and most touristic areas in Manhattan. Coincidentally, the hot spot of the rejection rates is in the same area. Again, we see that the highest rejection rates occur in areas with high demand. The effect of the road network layout is less noticeable in these plots. The demand pattern seems to be dominant in determining where rejections take place.

We further research this by creating bubble graphs that show the number of rejections per request for each node in the network. These charts can be seen in Figure 3-3. The left figure shows the rejection rate for each node that has had at least one request. The right figure shows the same but with the zero rejection rate nodes left out for clarity. Again, we see that the rejections follow a similar pattern as the demand in Figure 3-2(a). The effect of the road network seems to be less noticeable. The more remote areas (for instance, the entire upper area) even have a very low overall rejection rate. We conclude that the demand pattern mostly determines where rejections will take place.



(a) All nodes (b) Nodes with rejections

Figure 3-3: Bubble charts showing the rejection rate per node for a large-scale Manhattan case study using original FMF. Nodes that did have any requests throughout the simulation are omitted.

3-3 Expected Benefit

In the previous sections, we have seen that the rejection rates per node are unevenly spread over the operation area. Furthermore, we have seen that this is closely related to skewed demand patterns. Our research goal is to create an FMF that more evenly spreads the rejection rates. The hypothesis is that this improves an SMoD in two ways. First, a social benefit (Section 3-3-1). And second, a performance benefit (Section 3-3-2).

3-3-1 Social Benefit

The first expected benefit we discuss is a benefit from a social point of view. Having evenly spread out service levels would help reduce the issue of transport disadvantages for low-income groups and communities. This issue is explicitly linked to further problems such as

social exclusion and a lack of access to key services [28]. Also, equally spreading the rejection rates increases the perceived reliability of the system. Using the original FMF in a case study for New York, the service rate could be 100% in some areas, while it could be as low as 30% in other areas. Customers that have been frequently rejected in a certain area can develop a sense of discrimination when they become aware of this skewed distribution. In the end, those customers might stop using the system. This would hurt the profitability of the SMoD system.

3-3-2 Performance Benefit

The second benefit we expect is a benefit we are not explicitly aiming at. Furthermore, it is to be seen through experimentation, whether this benefit emerges at all. This potential benefit is that the overall service rate increases from steering vehicles towards high rejection rate areas. The original FMF has almost 100% service rate in low demand areas and much lower service rates in high demand areas. This might be the case because too many vehicles are kept driving around in these low demand areas. They do not become idle as there is always the next customer to service. Because they are not idle, they will not be rebalanced towards the high rejection rate areas. The idea to actively steer vehicles to high rejection rate areas might subtract these "near-idle" vehicles and put them to better use in the high demand areas. This would potentially increase the overall service rate of the system. Whether this is the case is to be seen through experimentation.

Chapter 4

Methods

The goal of this thesis is to achieve a fairer system by adapting the FMF of [2]. This can be done in several ways. In this chapter, we discuss the two strategies we used and their variations. The main idea behind both strategies is to adapt the weights in the assignment step's objective function in the original framework. The original objective function was given as:

$$\Sigma_{\text{optim}} = \arg\min_{\chi} \sum_{i,j \in \mathcal{E}_{TV}} c_{ij}\epsilon_{ij} + \sum_{k \in \{1,\dots,n\}} c_{ko}\chi_k. \tag{4-1}$$

Here, c_{ij} can be seen as the cost of performing trip T_i using vehicle v_j. Furthermore, c_{ko} can be seen as the penalty for rejecting a request, the standard value for this is p_{KO}. We propose two techniques in this chapter. First, the so-called rejection penalization technique. The idea behind this technique is to determine the height of the penalty based on the request's location. The second technique is the so-called trip penalization technique. Here, the height of the trip costs is adjusted based on the stops visited within this trip. Table 4-1 shows the symbols and their definition used throughout this chapter.

Table 4-1: Symbols and definitions used throughout this chapter.

Symbol	Definition
p_{KO}	Rejection penalty original framework
δ	Tuning factor request penalization technique
λ	Tuning factor trip penalization technique
P	Tuning factor lower bound trip penalization technique
Δ_{r_k}	Difference rejection rate origin request r_k and average rejection rate
Δ_{T_i}	Difference average rejection rate trip T_i and average rejection rate

The methods we discuss in this chapter have been selected after researching many different methods. During this research, we experimented with these methods in similar experiments as will be presented in Chapter 6 and Chapter 7. Due to limitations on available computation power, we could not thoroughly experiment with every idea. The time constraints on this

work required us to converge to a limited number of methods quickly. We have selected the methods in this chapter because they performed best in this early testing phase. In Appendix B, we have given an overview of all disregarded methods to give the reader an idea of the different methods we have experimented with.

4-1 Rejection Penalization

The rationale behind the rejection penalization technique is simple. In the original framework, the penalty for rejecting a request was the same for each request. Our idea is to change the height of a penalty based on the corresponding request. Now, we have created a mechanism to influence which request gets serviced and which request is ignored. Assigning a large penalty to a request will result in not rejecting that particular request (in most cases), while assigning a small penalty to a request increases the possibility of rejecting that request. When using the rejection penalization method, we now solve the following objective function during the assignment step:

$$\Sigma_{\text{optim}} = \arg\min_{\chi} \sum_{i,j \in \mathcal{E}_{TV}} c_{ij}\epsilon_{ij} + \sum_{k \in \{1,\dots,n\}} c^{k}_{ko}\chi_k. \tag{4-2}$$

The added superscript k implies that the penalty differs for each request r_k. Now, we have to define how c^k_{ko} is calculated given a request r_k. Ideally, we would like to incorporate three main ideas in this function:

1. The penalty needs to be large for a request that should not be rejected, and it needs to be smaller for a request that may be rejected. Linking this to our original goal, requests in areas with a high rejection rate should receive a higher penalty than requests in areas with a historically low rejection rate. Therefore, the penalty's height should be dependent on the height of the rejection rate at the associated node, and the average rejection rate of the entire operating area.
2. The height of the penalty c_{ko} in the original framework was equal to p_{KO}. This value was chosen such that it is typically larger than the most expensive trip. After all, the system's service rate would drop if it is often cheaper to reject a request than perform a trip. We want to retain this principle. Therefore, the penalty should always be higher than the most expensive trip.
3. Ideally, our method should be tuneable for different cases. Therefore, the penalty height should also be dependent on a tuning factor to achieve different desired behaviour.

4-1-1 Linear Rejection Penalization

We have come up with a straightforward function to calculate the individual penalties' height that retains the three ideas as described above. The function is as follows:

$$c^k_{ko} = \max\left(\max_{ij}\{c_{ij}\},\, p_{KO} + \delta \cdot \Delta_{r_k}\right), \tag{4-3}$$

where δ is a tuning factor and Δ_{r_k} is the difference between the current rejection rate at origin node of request r_k and the rejection rate of the entire operation area. $\Delta_{r_k} \in (-1, 1)$

since we can identify two extreme cases: first, the rejection rate at the origin of the node is 1 and the average rejection rate is close to 0 (Δ_{r_k} is just smaller than 1), and second, the rejection rate at the origin of the node is 0 and the average rejection rate is close to 1 (Δ_{r_k} is just larger than -1). Now, if the origin node of request r_k has an above-average rejection rate, Δ_{r_k} will be positive, and the penalty will be increased. Similarly, if the rejection rate at the origin of the request is below average, the penalty will be lowered. A visualisation of this can be seen in Figure 4-1. Henceforward, this particular rejection penalty method will be referred to as the R-method.

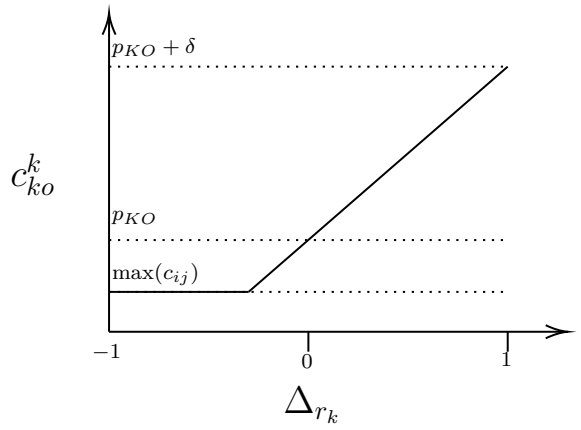


Figure 4-1: Plot of (4-3), the R-method. The slope of the line is determined by δ.

4-1-2 Non-linear Extensions

In the previous section, we proposed a rejection penalization method that adjusts p_{KO} linearly using Δ_{r_k}. In this case, the height of the adjustment scales linearly with increasing Δ_{r_k}. It might be useful to introduce a non-linear relationship here. For instance, we might want to increase the penalty of a request in an above-average rejection by a lot. Similarly, we might not care that much about requests originating from average rejection rate areas. Therefore, we could extend our formula by introducing a quadratic component:

$$c^k_{ko} = \max\left(\max_{ij}\{c_{ij}\},\, p_{KO} + \delta \cdot \operatorname{sgn}(\Delta_{r_k}) \cdot \Delta^2_{r_k}\right). \tag{4-4}$$

Henceforward, this particular rejection penalty method will be referred to as the R^2 method. Here, the adjustments to the penalties of requests originating from low or high rejection rate areas are steep, whilst they are small for average requests. On the other hand, we might care equally about any request originating from a non-average area. We can model this idea similarly by introducing a square root:

$$c^k_{ko} = \max\left(\max_{ij}\{c_{ij}\},\, p_{KO} + \delta \cdot \operatorname{sgn}(\Delta_{r_k}) \cdot \sqrt{|\Delta_{r_k}|}\right). \tag{4-5}$$

Henceforward, this particular rejection penalty method will be referred to as the $R^{\surd}$ method.

4-2 Trip Penalization

A second way to achieve a fairer system is adjusting the costs of performing a trip T_i with vehicle v_j. This idea follows a similar rationale to the rejection penalization method. We want to increase the costs of a trip that mostly stops in areas with low rejection rates, and we want to decrease the costs of trips that pass through high rejection rate areas. By doing so, we steer the assignments in a direction that favours high rejection rate areas. When using this trip penalization method, we now solve the following objective function during the assignment step:

$$\Sigma_{\text{optim}} = \arg\min_{\chi} \sum_{i,j \in \mathcal{E}_{TV}} c_{ij}^{new} \epsilon_{ij} + \sum_{k \in \{1,\dots,n\}} c_{ko} \chi_k. \tag{4-6}$$

the added superscript new implies that we have adjusted the original value for c_{ij}. Now, we have to define how c_{ij}^{new} is calculated given c_{ij}. Similarly to the rejection penalization technique, we would like to incorporate three key ideas implicitly:

1. It is a bit more difficult to decide whether a trip passes through a relatively high rejection rate area or a low rejection rate area. This is the case because a trip often consists of multiple stops spanning different areas. We use the average rejection rate of the stops in a trip to determine whether it passes through a low or a high rejection rate area.
2. The adjusted trip costs should not be negative. This would alter the workings of the original framework too much.
3. Similarly to the rejection penalty technique, we would like this method to be tuneable for different cases. Therefore, we want to make use of a tuning factor.

4-2-1 Linear Trip Penalization

The most straightforward function that incorporates the three principles described above is given as:

$$c_{ij}^{new} = \max\left(\frac{c_{ij}}{P},\, c_{ij} - \lambda \cdot \Delta_{T_i}\right), \tag{4-7}$$

where P is a tuning factor that determines the lower bound (typically equal to 2, 3 or 4), and λ is the tuning factor that tunes the method's aggressiveness. The variable Δ_{T_i} is the difference between the average rejection rate of the stops in trip T_i and the entire network's average rejection rate. Unlike the rejection penalty technique, we now subtract this Δ_{T_i} times a tuning factor instead of adding it. This is the case because we want to *lower* the cost of a trip that goes through a *high* demand area in order to increase the chance of selecting that trip. A visualisation of (4-7) is shown in Figure 4-2. Henceforward, this trip rejection penalty method will be referred to as the T_P method.

4-2-2 Non-linear Extensions

Similarly to the rejection penalisation method, it might be useful to use non-linear operators. Again, we could use a square operator:

$$c_{ij}^{new} = \max\left(\frac{c_{ij}}{P},\, c_{ij} - \lambda \cdot \text{sgn}(\Delta_{T_i}) \cdot \Delta_{T_i}^2\right). \tag{4-8}$$

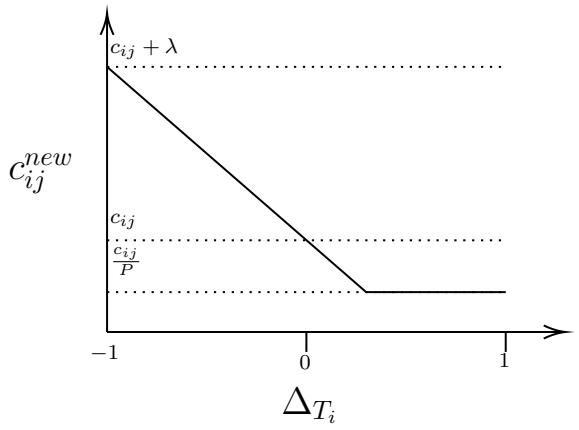


Figure 4-2: Plot of (4-7). The slope is determined by λ.

Henceforward, this particular trip penalty method will be referred to as the T_P^2 method. A root operator could also be used:

$$c_{ij}^{new} = \max\left(\frac{c_{ij}}{P},\, c_{ij} - \lambda \cdot \text{sgn}(\Delta_{T_i}) \cdot \sqrt{|\Delta_{T_i}|}\right). \tag{4-9}$$

Henceforward, this particular trip penalty method will be referred to as the $T_P^{\surd}$ method.

4-3 Combining Methods

The methods can be combined and used simultaneously; the objective function that gets solved in the assignment step now becomes:

$$\Sigma_{optim} = \arg\min_{\chi} \sum_{i,j \in \mathcal{E}_{TV}} c_{ij}^{new} \epsilon_{ij} + \sum_{k \in \{1,\ldots,n\}} c_{ko}^{k} \chi_k. \tag{4-10}$$

The difficulty that stems from combining both methods is the increased difficulty of correctly tuning the method. Now, both the tuning factor for the rejection penalization (δ) and the trip penalization (λ) have to be tuned simultaneously. One could also choose to set λ equal to δ.

To ensure that rejecting a request remains more expensive than the most expensive trip, the order of adjusting the original values has to be taken into account. First, the trip costs should be adjusted (using for instance (4-7)). Afterwards, the request penalties should be adjusted (using for instance (4-3)). One makes sure that the correct lower bound is used when calculating the heights of the individual rejection penalties.

Chapter 5

Simulation Setup and Data

In this chapter, we explain how our experiments have been set up. First, we describe how we implemented our simulation. Second, we describe how we obtained the data to model artificial scenarios and a case study for New York.

5-1 Simulation Setup

We simulate an SMoD system using a Discrete Event Simulation (DES) with fixed-increment time progression implemented in MATLAB[1]. We decided to implement the simulation in MATLAB because we could use an available simulation setup from earlier research. The simulation model consists of three main components:

1. road network,
2. fleet of vehicles,
3. set of mobility requests.

The road network remains static throughout our simulation. The set of requests and the fleet of vehicles are updated in each time step. The state of the SMoD system is given by the pool of requests that are waiting to be picked up, and all individual states of the vehicles. We will further explain the three main components in the next subsections.

5-1-1 Road Network

The road network is represented as a directed graph in which the nodes represent intersections and edges represent road segments. The weights of these segments are the associated travel times in minutes. We assume that the road network does not change throughout the

[1] www.mathworks.com

simulation, i.e. no arcs are deleted or created, and the travel times remain the same. Therefore, we are not simulating any changing traffic conditions such as rush hours. It would be possible to simulate this by constantly updating the graph describing the road network. For this research, we decided to not further complicate our simulation model with this option. To speed up simulations, shortest paths between all nodes are calculated a priori using Dijkstra's algorithm. In Section 5-2, we explain how we constructed a road network for an artificial scenario and a New York case study.

5-1-2 Fleet of Vehicles

A vehicle has the capacity to transport up to ν passengers, or picked up requests, at once. Throughout this thesis, we set $\nu = 4$. The state of a vehicle is defined by:

1. The location of the vehicle, represented as a 3-dimensional vector. The three entries in the vector are the last node visited, the next planned node, and a percentage λ indicating the percentage that still needs to be covered between those nodes.
2. List of planned stops.
3. A list of requests that have been assigned to the vehicle. Moreover, the list contains information whether a request still needs to be picked up, is picked up, or is already dropped off.

5-1-3 Mobility Requests

Each request consists of an origin node, a destination node, a request time, and the number of passengers. In our research, we will always assume that the origin and destination of a request appear on a node. Furthermore, we will always assume that the number of passengers is equal to one. Moreover, we define the maximum waiting time $\Omega = 7$ minutes and the maximum delay time $\Delta = 14$ minutes.

5-1-4 Overview

This subsection, provides an overview of where the individual components fit in the simulation, as seen in Figure 5-1. In this figure, the data consists of a graph describing the road network and a set of mobility requests. The simulation parameters consist of the capacity of each vehicle ν, the maximum waiting time Ω, and the maximum delay time Δ. The first set of optimization parameters are weights used in the original FMF of [2]. The second set of optimization parameters, λ and δ, will be introduced in Chapter 4. These parameters are used in the extensions we propose for the original FMF.

We simulate the system using steps of one minute for a total of T minutes. Each time step, we perform one iteration of the FMF. After the FMF determines all vehicles' planned routes, we move the vehicles and update the system states.

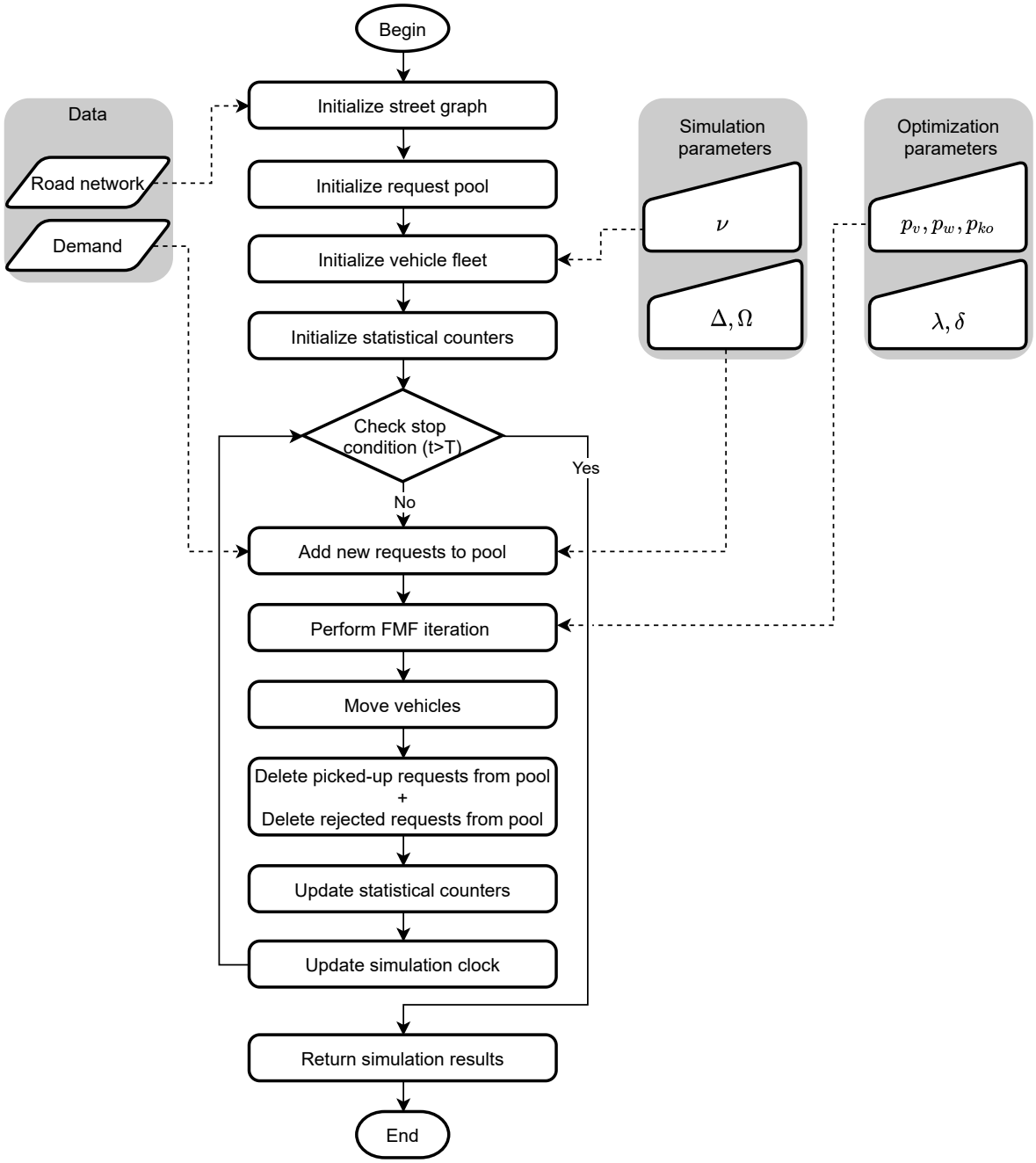


Figure 5-1: Flowchart of an SMoD system simulated as a DES implemented in MATLAB.

5-2 Data

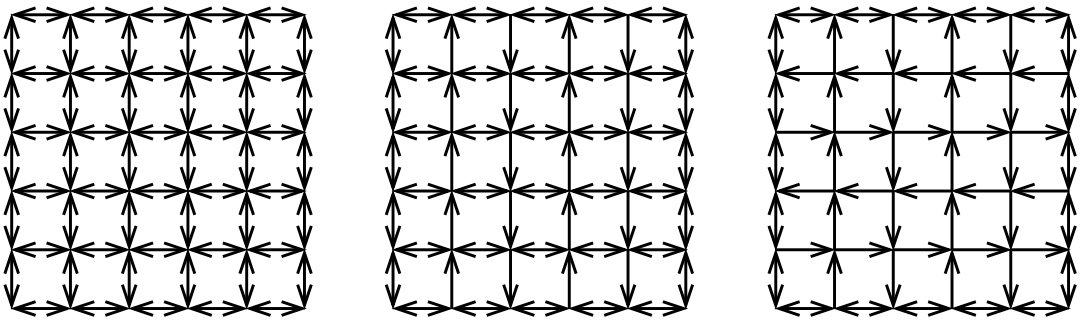
This section describes the different data we used throughout this thesis. First, we explain how this data is created for artificial scenarios. Second, we show how we obtained the data to simulate a real-life SMoD system for New York.

5-2-1 Artificial Scenarios

We have developed multiple artificial small-scale scenarios to experiment with different FMFs. These scenarios can provide us with valuable insights without requiring too much computational power. A scenario consists of a road network and a demand pattern.

Road Network

We have created three different road networks. Each road network consists of a square grid. These road networks differ by the number of one-way streets. The first road network has only two-way streets, the second has one-way vertical streets (except the roads on the edges), and the third has only one-way streets (except the roads on the edges). The streets on the edges are always two-way to simulate the effect of having a ring road around a city. Furthermore, it was needed to avoid creating dead ends. The different road networks are designed to resemble the city block system of North American cities. Moreover, the square blocks make it easier to visualize performance metrics for each node. In Figure 5-2, the three different road networks can be seen. This image shows the road networks as a 6-by-6 grid; our actual road networks are 20-by-20 but follow the same pattern. The drive time between each node and its adjacent neighbour is 1.5 minutes.



(a) All two-way (b) One-way vertical (c) All one-way

Figure 5-2: Visual of three types of artificial road networks used in artificial experiments. Here the networks are shown as 6-by-6, in the actual experiments the networks are 20-by-20 but follow the same pattern.

Demand Data

Besides the road network, a scenario is also determined by the set of mobility requests. We have created three basic request patterns that can be combined into a demand pattern for

a scenario. The first basic pattern is the Random (RAND) pattern. Each request that is created using this pattern is from a random node in the network towards any other node in the network. The second pattern is the Left-2-Right (L2R) pattern. A request is always from a random node in the left half-plane to a random in the right half-plane. Lastly, we created the Centre-2-Surroundings (C2S) pattern. A request is always from a random node in the centre area towards a random node in the outer area. We created five different demand patterns by combining these basic patterns. Table 5-1 gives an overview of these demand patterns. To explain the notation; if we create 100 requests using the 20C2S pattern, then, 20 requests will be created using the C2S pattern, and 80 requests will be created using the RAND pattern.

Table 5-1: Five artificial demand patterns used throughout this thesis.

Demand pattern	RAND	L2R	C2S
RAND	100%		
5L2R	95%	5%	
10L2R	90%	10%	
10C2S	90%		10%
20C2S	80%		20%

Using three different road networks and five different demand patterns, we can create a total of fifteen different scenarios. We use all fifteen of them in Chapter 3 to elaborate on the motivation for the writing of this thesis. We use just three of them in Chapter 6 to benchmark our methods; this is to reduce our experiments' computational load..

5-2-2 New York Scenario

The artificial scenarios are useful for experimenting with different FMFs. However, they are not representative of a real-life SMoD system. To experiment with different FMFs in real-life scenarios, we model an actual city and real mobility demand by creating a scenario that represents the area of Manhattan. In the next sections, we will explain how we achieved this for the street network and the mobility demand.

Street Network

Similar to the artificial scenarios, we model the street network as a weighted directed graph. We use a data set that contains the coordinates of all intersections in New York and the drive times between each intersection for each hour of the day [13]. We adapt this data set by averaging drive times over the 24 hour period. We use these average times during our simulations. We do so because the original data had large fluctuations between consecutive hours. Figure 5-3(a) shows the 4091 nodes of the data set plotted over the map of New York.

Demand Data

We simulate real-life mobility needs by using the 2014 New York City Cab Data set [31]. This publicly available data set contains roughly 165 million taxi trips made by yellow cabs in New

(a) Nodes in Manhattan (b) Fence around Manhattan

Figure 5-3: Vertices of graph network and polygon fitted over Manhattan.

York. We use the following features of this data set: 1) pickup date and time, 2) the number of passengers, and 3) pick up and drop-off coordinates.

We only have a road network model for Manhattan, not the entire city of New York. Therefore, we filtered all taxi trips that started and ended inside a polygon fitted around Manhattan, see Figure 5-3(b). Next, we assigned the origin and the destination of each trip to the closest node in our street network model using the euclidean distance. We filter all trips with less than three minutes of drive time to get rid of any erroneous records in the data set. Lastly, we selected only the trips with just one passenger. For our experiments, we selected Friday, March 7^{th} as an average day at random to be used in our experiments. Figure 5-4 shows the number of requests per minute on this day.

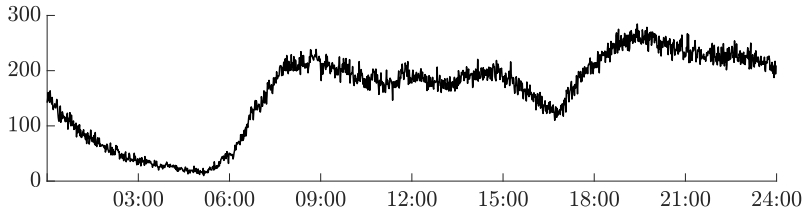


Figure 5-4: Number of requests per minute in Manhattan on March 7^{th} after filtering.

Our experiments will not always be performed using all the requests on March 7^{th}. This is often not possible due to limited available computing power. In some cases, we will only simulate a given time window on that day, or select a subset of requests. In the latter case, we will scale the number of vehicles in the experiment accordingly.

Chapter 6

Results Artificial Scenarios

In this chapter, we discuss the results of our methods for artificial scenarios. The experiments presented were performed on the artificial 20-by-20 grid with all two-way streets. We ran each experiment with three different demand patterns: RAND, 10L2R, and 20C2S. We constantly use all these three different demand patterns to get an understanding of how our methods perform across different scenarios. Each pattern has a different average trip length. This results in different rejection rates and therefore different Gini indexes. By comparing the results of our experiments for the different demand patterns, we might be able to recognize strengths and weaknesses in our methods. Each experiment is performed for 100 hours such that the system reaches a steady-state, the justification for this can be seen in Appendix A. Furthermore, each experiment is repeated three times, and the average results are reported in this chapter. The set of requests in each run is the same. However, the initial vehicle distribution is varied for these three runs. Several experiments with different numbers of vehicles and requests are presented in this chapter. To help the reader understand how the different experiments are related to each other, we will plot the results of the "standard settings" on a light grey background. Results from experiments with different numbers of vehicles and requests are plotted on a white background. We define the standard number of vehicles as 65 and the standard number of requests as 5 per minute. Before we present the various experiments' results, we first introduce the idea of the posterior trivial solution and the naming conventions for the methods introduced in Chapter 4.

6-1 Posterior Trivial Solution

This research aims to find a way to spread the rejection rates evenly over the operation area. We use the Gini index as a measure for this. A secondary benefit we are expecting is a decrease in the overall rejection rate as well. However, it might be possible that our method increases the rejection rate while decreasing the Gini index. In that case, we to compare the results between the original algorithm and our adjusted version. We cannot simply compare the Gini index obtained by our method with the Gini index obtained by the original method

if the original method's rejection rate is lower. We have to adjust the original method's rejection rates so that the rejection rate of both methods becomes comparable. We call this the posterior trivial solution: adding artificial rejections at the right nodes such that the rejection rates become equal and the Gini index is minimized. This posterior trivial solution is obtained by solving a non-linear integer program. The objective function for this program is the Gini index, which is defined as:

$$G = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}|x_i - x_j|}{2n\sum_{i=1}^{n} x_i}, \tag{6-1}$$

where x_i is the income of person $i = 1, \ldots, n$. In our case, a 'person' is the same as a node, and its 'income' is the rejection rate at that node. The rejection rate at node i is defined as the number of rejections divided over the number of requests at node i (t_i). The number of rejections consists of the number of rejections that occurred throughout the experiment (r_i), and the number of artificial rejections added (a_i). The number of artificial rejection rates added to each node is our decision variable. The maximum number of artificially added rejections A, is defined as the maximum number of rejections that can be added such that the overall rejection rate does not increase more than $x\%$. From here we create the following Non-linear Integer Programming problem to find the lowest posterior trivial Gini index (G^*):

$$\begin{aligned}
\min_{a} \quad & \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left|\frac{r_i+a_i}{t_i} - \frac{r_j+a_j}{t_j}\right|}{2n\sum_{i=1}^{n}\frac{r_i+a_i}{t_i}} \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_i \leq A \\
& r_i + a_i \leq t_i && \forall i = 1, \ldots, n \\
& a_i \in \mathbb{N} && \forall i = 1, \ldots, n.
\end{aligned} \tag{6-2}$$

The objective function in this program is simply the Gini index. The first constraint ensures that no more artificial rejections are added than the maximum number. The second constraint ensures that, for each node, the number of rejections (real and artificial) does not grow past the total number of requests at that node. The above problem is difficult to solve since the objective function is non-linear, and the decision variables are integer. However, as this is method is solely used for comparing our results, it is outside our scope to find the global minimum. Therefore, we use a straightforward heuristic to find a reasonable solution. This heuristic is displayed as Algorithm 1 in Appendix D. The heuristic adds artificial rejections to nodes one at a time. At each iteration, it searches for the node with the lowest rejection rate that will not surpass the overall rejection rate if an artificial rejection is added.

6-2 Naming Conventions

In the remainder of this thesis, we will refer to the different penalization techniques using the names as given in Table 6-1.

Table 6-1: Overview of names used for different penalization functions used throughout this thesis.

Type	Name	Variant	Function
Rejection	R	Linear	$c^k_{ko} = \max\left(\max\{c_{ij}\},\, p_{KO} + \delta \cdot \Delta_{r_k}\right)$
	R^2	Squared	$c^k_{ko} = \max\left(\max\{c_{ij}\},\, p_{KO} + \delta \cdot \text{sgn}(\Delta_{r_k}) \cdot \Delta^2_{r_k}\right)$
	$R^{\surd}$	Root	$c^k_{ko} = \max\left(\max\{c_{ij}\},\, p_{KO} + \delta \cdot \text{sgn}(\Delta_{r_k}) \cdot \sqrt{\lvert\Delta_{r_k}\rvert}\right)$
Trip	T_P	Linear	$c^{new}_{ij} = \max\left(\frac{c_{ij}}{P},\, c_{ij} - \lambda \cdot \Delta_{T_i}\right)$
	T^2_P	Squared	$c^{new}_{ij} = \max\left(\frac{c_{ij}}{P},\, c_{ij} - \lambda \cdot \text{sgn}(\Delta_{T_i}) \cdot \Delta^2_{T_i}\right)$
	$T^{\surd}_P$	Root	$c^{new}_{ij} = \max\left(\frac{c_{ij}}{P},\, c_{ij} - \lambda \cdot \text{sgn}(\Delta_{T_i}) \cdot \sqrt{\lvert\Delta_{T_i}\rvert}\right)$
Combined	RT_P	Linear	$c^{new}_{ij} = \max\left(\frac{c_{ij}}{P},\, c_{ij} - \lambda \cdot \Delta_{T_i}\right)$ **and** $c^k_{ko} = \max\left(\max\{c_{ij}\},\, p_{KO} + \delta \cdot \Delta_{r_k}\right)$

6-3 Varying Tuning Factor

The first set of experiments were performed to get an understanding of the effects of the tuning parameters. In these experiments, we simulated the SMoD system using 65 vehicles, and we generated 5 requests per minute using three different demand patterns. We experimented with five different penalization formulas: 1) two trip penalization functions with different lower bounds, 2) a rejection penalization function, and 3) two combined functions with different lower bounds. The results of these experiments can be seen in Figure 6-1. This figure shows the obtained rejection rates (top row) and Gini indexes (bottom row), for different tuning factors. Furthermore, this figure also shows the rejection rates and the original framework's Gini indexes (black horizontal lines).

Figure 6-1 shows that, for each method and each demand pattern, the Gini index decreases when increasing the penalisation methods' tuning factors. However, on average, the rejection rate increases for higher tuning factors. Fortunately, there are still a few cases where a lower or equal rejection rate was achieved compared to the original framework without any penalties. Therefore, we have shown that our method can be tuned to lower both the rejection rate and the Gini index in the artificial scenarios.

The differences between demand patterns are clearly visible. The 10L2R scenario results in the highest rejection rates and also the highest Gini indexes. The 20C2S scenario gives the lowest rejection rates and Gini indexes. Furthermore, we notice that the difference between the lowest and the highest obtained rejection rate seems to be the largest in the 10L2R scenario, whereas it seems to be the smallest in the 20C2S scenario.

Although there are a few instances in which a lower rejection rate was obtained, we notice that, in most cases, the rejection rates of the penalization methods are slightly higher than the rejection rate obtained by the original framework. Here, we have to use the idea of the

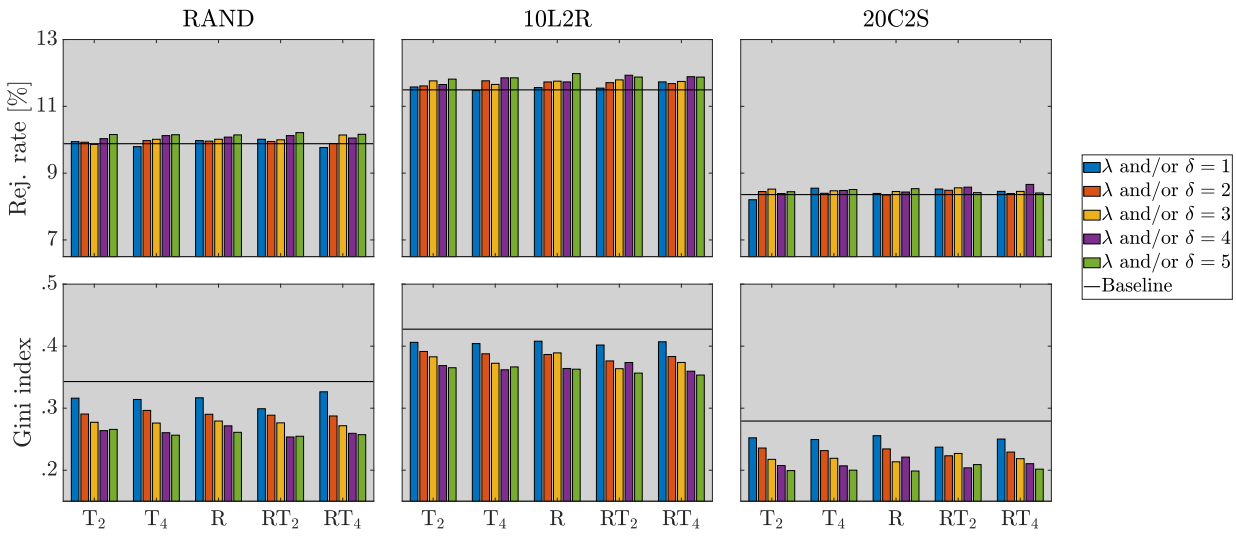


Figure 6-1: Average rejection rate and Gini index, for five different penalization techniques and the original framework, for different values of λ and δ, for three scenarios. For the combined methods, the legend shows the values for both λ and δ. For the individual methods, the legend shows the corresponding parameter values for that method. 65 vehicles and 5 requests per minute were used in these experiments.

posterior trivial solution to assess whether our method works in those cases. Figure 6-2 is created to compare our methods against the posterior trivial solution. This figure is created by selecting, for each method, the lowest Gini indexes in Figure 6-1 that do not have a corresponding rejection rate increase of more than $x\%$ (0, 1, or 2%) compared to the original framework. Each row in the figure now shows the lowest Gini index without increasing the rejection rate past $x\%$. If no bar is shown, then the method did not achieve, for any tuning parameter setting, a rejection rate lower than $x\%$.

Figure 6-2 shows that most of our methods outperform the posterior trivial solution if a 1% or 2% increase in rejection rate is allowed. In those cases the tuning parameter was generally high. Figure 6-2 also shows that some methods even achieve a lower rejection rate (0% increase) than the original algorithm. In those cases, the tuning parameter was generally low (smaller or equal to 3). Therefore, we have shown that our methods can be tuned to outperform the original framework in terms of rejection rate and Gini index for these experiments.

From Figure 6-2, we conclude that the T_2 method achieves the best performance in terms of lowering the Gini index and the rejection rate. The T_2 method achieved a lower rejection rate than the original algorithm in 2 out of 3 scenarios. Furthermore, when allowing a 1 or 2% increase, the method provides us with the lowest Gini index compared to the other methods (in most cases). The figure also shows that the R method and the combined methods (RT_2, RT_4) often increase the rejection rate for any the tuning parameter value. This can be seen in the top row of the figure. Here, some bars were omitted for these methods indicating that they could not achieve a lower rejection rate than the original method.

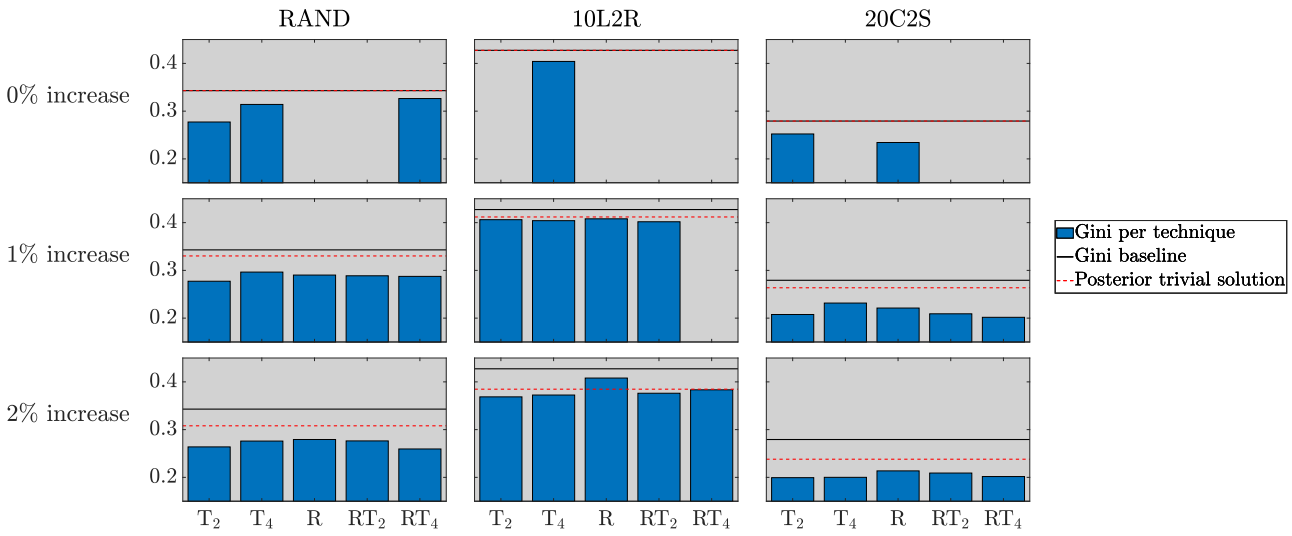


Figure 6-2: Lowest average Gini index obtained by each penalization technique without increasing the rejection rate with more than x percent (0, 1, or 2%) compared to the original algorithm (black line) and the original algorithm with the trivial solution (red dashed line).

6-4 Sensitivity

In this section, we research the sensitivity of our methods to varying experiment sizes. We present three sets of results. First, an experiment in which we vary the number of requests while keeping the number of vehicles the same. Second, an experiment in which we vary the number of vehicles while keeping the number of requests the same. Last, an experiment in which we vary the number of requests and vehicles while keeping the ratio between these the same. Again, we perform each experiment for three different demand patterns. Similar to the previous section, this helps us to assess the performance of our methods across different scenarios. Throughout this section, we use the same five different penalization methods as in the previous section.

6-4-1 Varying Number of Requests

Our first sensitivity experiment consists of varying the number of requests per minute while keeping the number of vehicles the same. By doing so, we will gather insights into our methods' performance for different heights of rejection rates.

The results of this experiment can be seen in Figure 6-3. For each number of requests per minute, Figure 6-3 shows the lowest obtained Gini indexes by our methods, which did not increase the corresponding rejection rate compared to the original framework. If there is no bar shown in the figure, the method could not achieve a rejection rate lower than the original method.

From Figure 6-3, we observe that, in the experiment with 3 requests, most techniques could achieve a rejection rate that was lower than the original algorithm. Furthermore, in those cases, the methods were also able to lower the Gini index slightly. Roughly the same observation can be made for the 4 request experiment. This is no coincidence, as these experiments

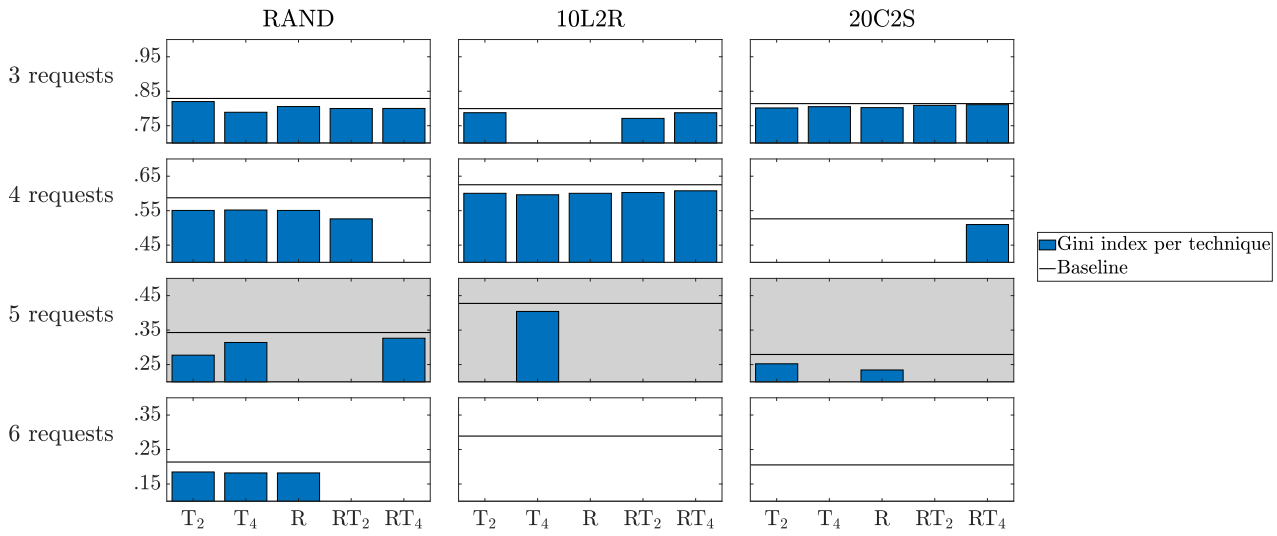


Figure 6-3: Lowest Gini indexes obtained *without* increasing the rejection rate compared to the original framework, for different number of requests per minute, for three different scenarios. The Gini index of the original framework is the horizontal line in each plot. If there is no bar shown, the method could not achieve a rejection rate lower or equal to the original framework. The row plotted in grey indicates the "standard settings" for the experiments; 5 requests per minute and 65 vehicles.

result in similar overall rejection rates; 1% and 3% for, respectively, 3 and 4 requests per minute. Our methods are not drastically influencing the original framework in low rejection rate scenarios as Δ_{T_i} and Δ_{r_k} are generally low. Therefore, the original values of the assignment objective function are not altered by large amounts. Furthermore, it is worth noticing that the Gini indexes are generally high. This is the case because, if there are few rejections, it does not matter where they are placed as they will always result in a high Gini index.

In the high rejection rate experiments (5 or 6 requests per minute), we notice that our methods were often unable to achieve a lower rejection rate than the original algorithm. This could be due to our algorithm being too aggressive in those cases. The values for Δ_{T_i} and Δ_{r_k} can be high. Therefore, the original objective function is changed significantly resulting in losing too much of its original information. Lowering the tuning factors might provide a solution.

6-4-2 Varying Number of Vehicles

In this subsection, we present an experiment in which we varied the number of vehicles while keeping the number of requests fixed at 5 requests per minute. By doing so, we again have experimented with different rejection rates, giving us insights into our methods' workings. In this experiment, we are using the same five penalization methods.

The results can be seen in Figure 6-4. For each number of vehicles, this figure shows the lowest obtained Gini indexes by our methods, which did not increase the corresponding rejection rate compared to the original framework.

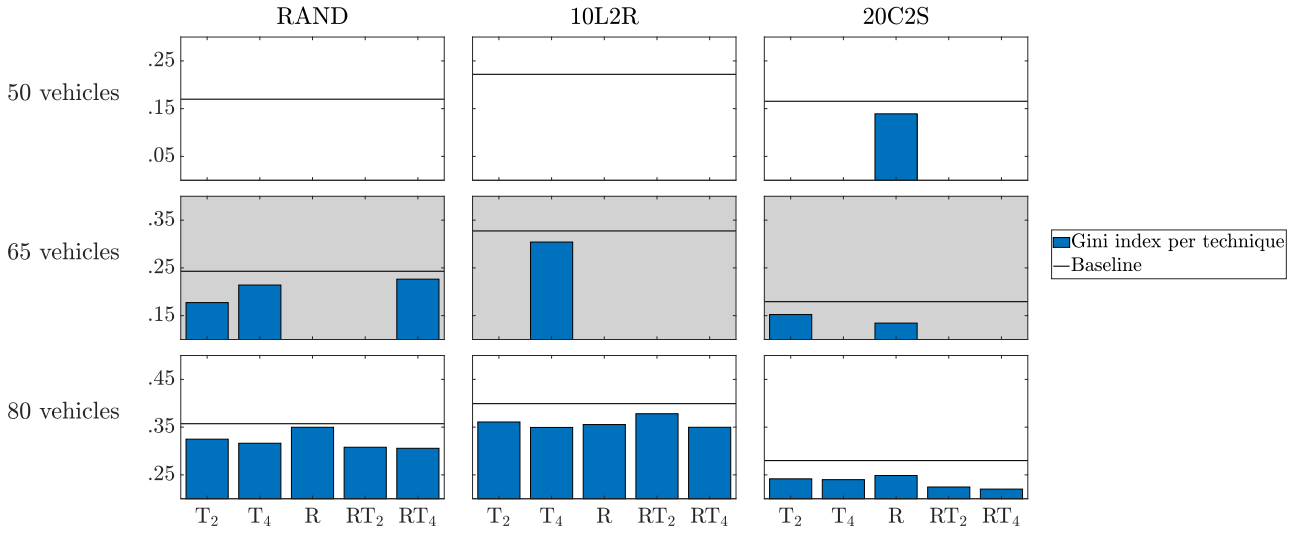


Figure 6-4: Lowest Gini indexes obtained *without* increasing the rejection rate compared to the original framework, for different number of vehicles, for three different scenarios. The Gini index of the original framework is the horizontal line in each plot. If there is no bar shown, the method could not achieve a rejection rate lower or equal to the original framework. The row plotted in grey indicates the "standard settings" for the experiments; 5 requests per minute and 65 vehicles.

From Figure 6-4, we make similar observations as from Figure 6-3. Our methods are better suited for low rejection rate experiments (80 vehicles) than for high rejection rate experiments (50 vehicles). The experiment using 80 vehicles shows that our methods were consistently able to reduce the Gini index and the rejection rate. While the experiment using 50 vehicles shows that our methods were rarely able to achieve a rejection rate equal to the original framework.

6-4-3 Same Ratio of Requests and Vehicles

In the previous subsections, we researched the effect of different fleet sizes and request rates. These two metrics influence the rejection rate of an experiment. From these experiments, we have seen that this has a large influence on our methods' performance. In this subsection, we investigate whether there is also an effect of the scenarios' overall size. Therefore, we present an experiment in which we varied the number of vehicles and requests simultaneously while keeping the ratio of requests to vehicles the same; 1 request per minute for every 13 vehicles. Again, we experiment with all five different penalization methods.

The results of these experiments can be seen in Figure 6-5. For each number of vehicles and requests per minute, Figure 6-5 shows the lowest obtained Gini indexes by our methods, which did not increase the corresponding rejection rate compared to the original framework.

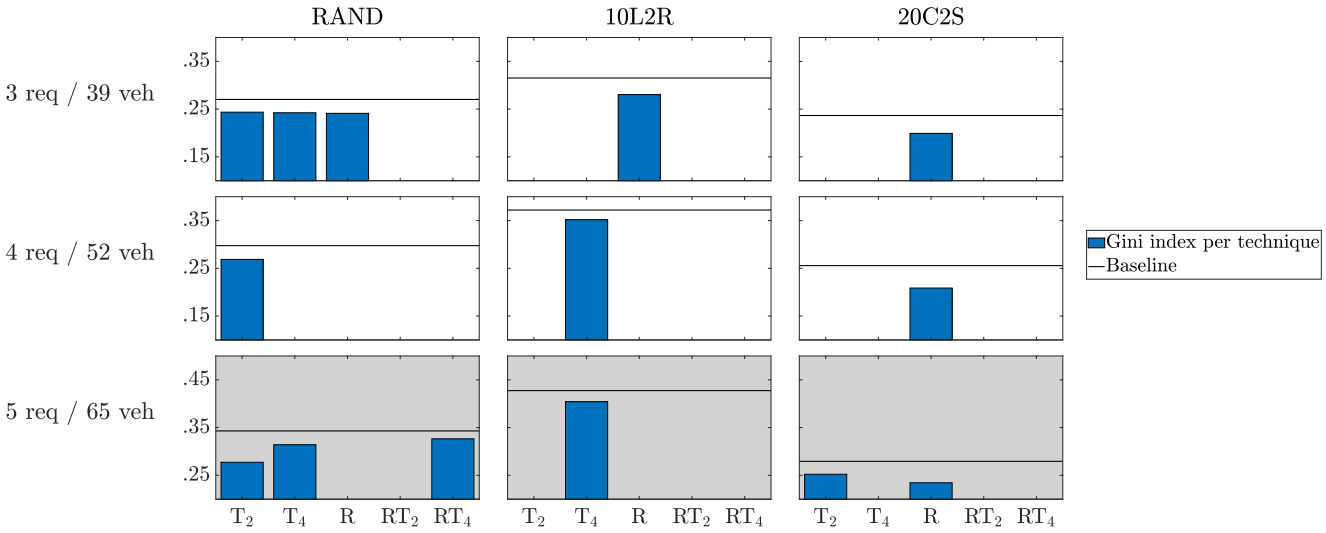


Figure 6-5: Lowest Gini indexes obtained *without* increasing the rejection rate compared to the original framework, for different number of vehicles and requests while keeping the ratio the same, for three different scenarios. The Gini index of the original framework is the horizontal line in each plot. If there is no bar shown, the method could not achieve a rejection rate lower or equal to the original framework. The row plotted in grey indicates the "standard settings" for the experiments; 5 requests per minute and 65 vehicles.

From Figure 6-5, we conclude that the scenarios' overall size has no real impact on the methods' performance. Our methods can achieve a rejection rate lower than the original framework in roughly the same amount of cases.

6-5 Non-linear Methods

In this section, we present the results of the nonlinear versions of the penalty techniques. We present the results of the rejection penalization method with added non-linear operators (Section 6-5-1) and of the trip penalization method with added non-linear operators (Section 6-5-2). Each experiment in this section was performed using 5 requests per minute and 65 vehicles.

6-5-1 Rejection Penalty

In this subsection, we present the results of our rejection penalization method with and without non-linear operators. The results of these experiments can be seen in Figure 6-6. The method using the square operator ($\Delta_{r_k}^2$) has comparable performance to the standard linear version. In roughly the same amount of cases, a rejection rate lower than $x\%$ compared to the original algorithm was found. Furthermore, the obtained Gini indexes seem to be mostly equal. Therefore, we conclude that using a square operator can be beneficial in particular cases. However, overall, it has no large positive influence on the performance of the system. The method using the root operator ($\sqrt{|\Delta_{r_k}|}$) gives a different result. In a few cases, it achieved a solution with a rejection rate lower than $x\%$ compared to the original algorithm. If it was able, the resulting rejection rate was, on average, similar to the linear version. Therefore, we conclude that using a root operator has no added benefit.

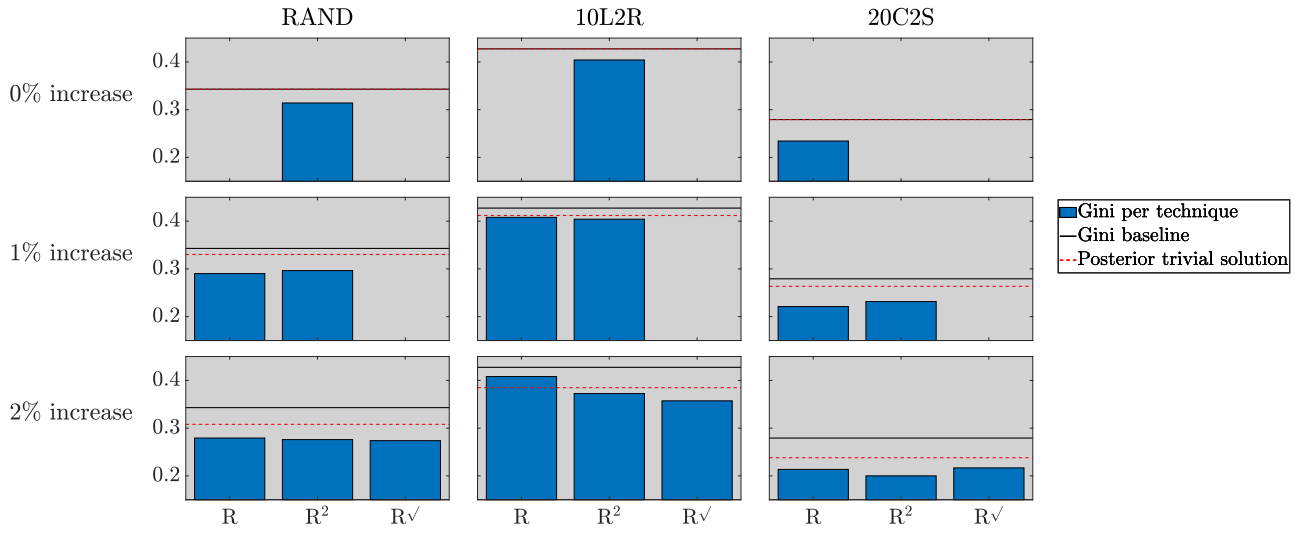


Figure 6-6: Lowest average Gini index obtained by each penalization technique without increasing the rejection rate with more than x percent (0, 1,or 2%) compared to the original algorithm (black line) and the original algorithm with the trivial solution (red dashed line).

We have shown that using a square root operator has no large impact on our method's performance. However, using a root operator has a large negative influence. This implies that a rejection penalization method works best if nodes with rejection rates further away from the average rejection rate should be penalized relatively more.

6-5-2 Trip Penalty

In this subsection, we present our trip penalization method results with and without non-linear operators (square and root). The results of these experiments can be seen in Figure 6-7. Again, the square operator achieved a lower rejection rate in roughly the same amount of cases as the non-linear method. However, in all these cases, the associated Gini index was always higher. It was sometimes even higher than the posterior trivial solution. Therefore, we conclude that using the square operator has no added benefit. A similar conclusion can be made for the root operator. Again, using a root operator resulted in less often achieving a rejection rate below the original method.

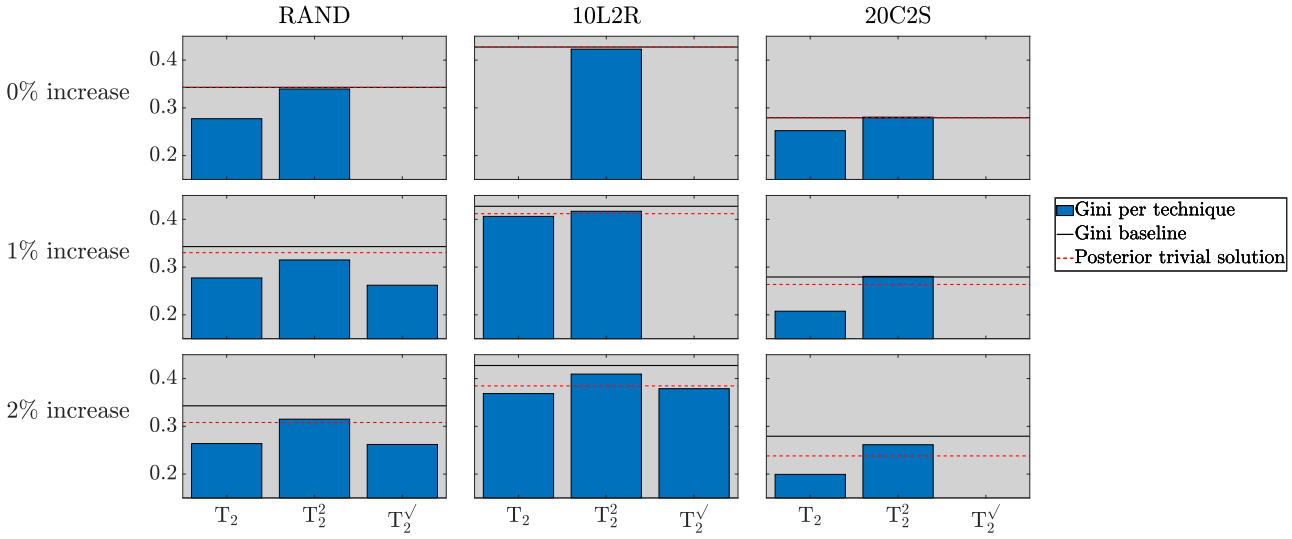


Figure 6-7: Lowest average Gini index obtained by each penalization technique without increasing the rejection rate with more than x percent (0, 1, or 2%) compared to the original algorithm (black line) and the original algorithm with the trivial solution (red dashed line).

From these observations, we can conclude that a trip penalization method performs best if the height of a penalty is linearly related to the difference of the average rejection rates in the trip and the average overall rejection rate.

6-6 Concluding Remarks

From the three sensitivity experiments' combined results, we conclude that the T_2 method performs slightly better than the T_4 method. The T_2 method achieved a rejection rate lower than the original framework more often than the T_4 method. Furthermore, in the cases both methods achieved a lower rejection rate, the T_2 method often gave a lower Gini index. Using a similar analysis, we can conclude that the R method performed worse in terms of Gini index and rejection rate than the T_2 method. The combined methods generally performed worse than either the T_P methods or the R method. Based on these experiments, we decided to still further experiment with a method from each category. Therefore, we have decided to further test the T_2 method, the R method, and the RT_2 method in the NY case study.

Chapter 7

New York Case Study

In this chapter, we present the results of our methods used in a simulation of Manhattan. Unless stated otherwise, in each experiment, we simulate a SMoD in Manhattan from 12:00 until 24:00. Between 12:00 and 18:00, the original framework without any penalization was used in each experiment. We do not start using the penalization methods right away because we want the system to reach a steady-state first. If we used our method from the start, the method would be unstable because each new rejection greatly influences system averages and thus our penalties. Then, at 18:00, we start using a penalization method. We report the results of the system between 21:00 and 24:00. We do this because, between 18:00 and 21:00, the system is still transitioning from "steady-state to steady-state". We are generally not interested in the results of this transition phase.

Due to limitations on the available computation power, for most of the experiments in this chapter, we only simulate the system using a subset of the requests that occurred on March 7^{th}, 2014. We scale down the number of vehicles accordingly. We assume that a full-scale experiment has 100% of requests and 3000 vehicles. In the last section, we present an experiment that simulates our methods using 1500 vehicles.

7-1 Same Ratio Vehicles to Requests

In this subsection, we present a set of experiments in which we vary the number of vehicles and requests while keeping the ratio of these the same. We have chosen to perform this experiment to understand better how our methods perform in different sized Manhattan simulations. Whereas in the previous chapter, we used the different demand patterns as a robustness analysis, we now use different sizes of simulations. In these experiments, we use three penalization methods: 1) a trip penalization method, 2) a rejection penalization method, and 3) a combined method.

As we can no longer assume that each node receives a fair amount of requests, we can no longer calculate penalties at the level of nodes. This is the case because nodes might fluctuate between a rejection rate of 0 and 100% in one iteration. Our methods would then react

aggressively to these insignificant changes. As presented earlier, we use a clustering method to group sets of nodes into regions. Then, the group average of these nodes is used when calculating a penalty. We set the penalty cluster size parameter, t^p_{max}, equal to 2.5 min (146 clusters). Similarly, we also calculate the Gini index based on clusters of requests and not based on individual nodes. The cluster size parameter for calculating the Gini index, t^G_{max}, is also set at 2.5 min (146 clusters). The results of this experiment can be seen in Figure 7-1.

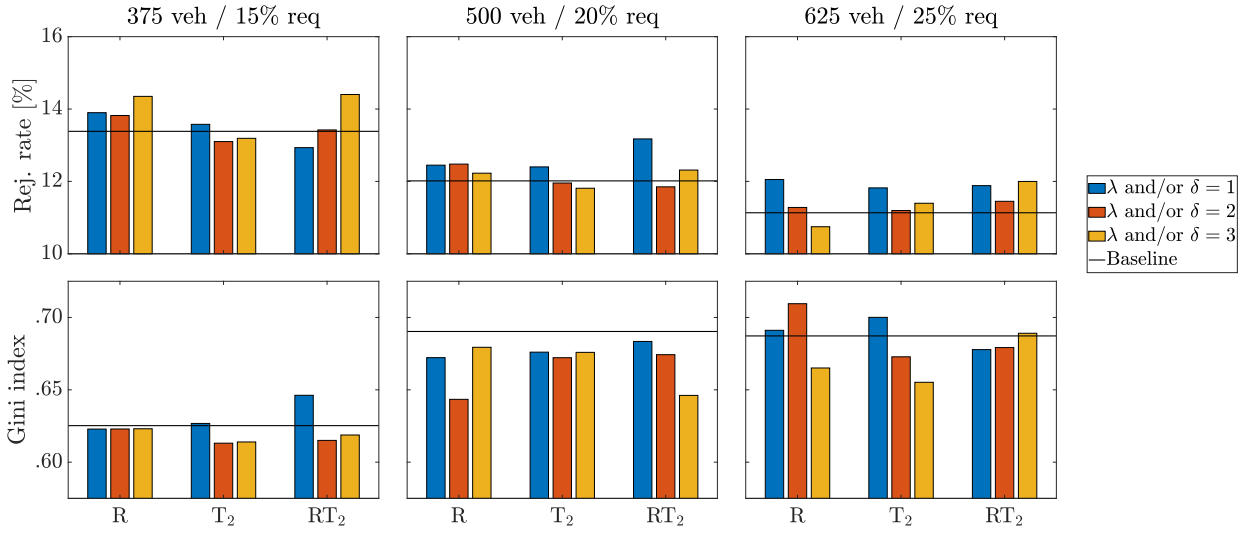


Figure 7-1: Comparison between rejection rate and Gini index of three different penalization techniques and the original framework, for three different sizes of case studies of the Manhattan simulation. Penalties and Gini indexes are calculated using $t^p_{max} = t^G_{max} = 2.5$. For the combined methods, the legend shows the values for both λ and δ. For the individual methods, the legend shows the corresponding parameter values for that method.

Figure 7-1 shows that, for some tuning factor settings, in some cases, both the rejection rate and the Gini index can be decreased compared to the original algorithm using one of our methods. However, in many cases, the rejection rate is roughly similar or higher. Furthermore, we also notice that the rejection rate no longer always increases for increasing tuning parameters. Similarly, the Gini index no longer always decreases for increasing tuning factor. This might be a result of the inherent difficulty of the sparsely and skewed distributed demand in Manhattan. The system is less stable; sub-optimal decisions can have a large influence. To better understand this effect's impact, we have repeated the experiment with 375 vehicles two more times. The averaged results of this experiment can be seen in Appendix C. In this plot, we notice that the Gini index now decreases for an increasing tuning factor for all three methods. For both the T-method and the RT_2-method the rejection rates now also show the expected increasing pattern. However, this plot also shows that the results for the R-method are still irregular. This again has to do with the difficulty of our method in combination with the sparse demand in Manhattan.

Like for the artificial experiments, we are also interested in how our methods perform compared to the posterior trivial solution. Figure 7-2 shows us the Gini indexes of our methods that were achieved without increasing the number of rejections by more than $x\%$. The figure shows us that, in most cases, our methods reduced the Gini index when just 1% or 2% more rejections were allowed. Also, it shows us that, in some cases, our methods were able to both reduce the Gini index and the rejection rate.

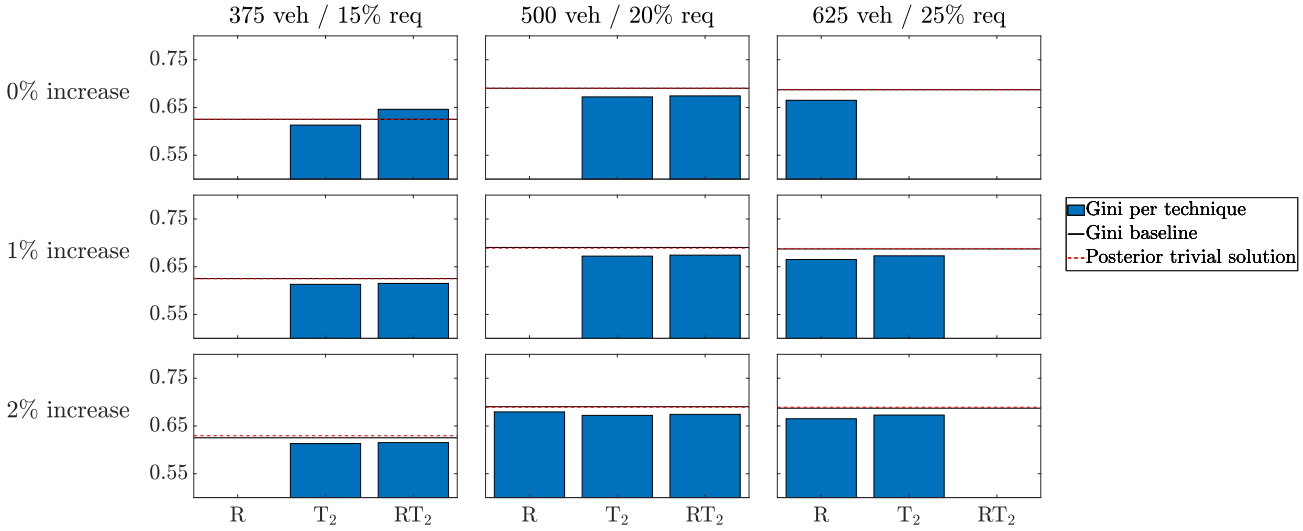


Figure 7-2: Lowest average Gini index obtained by each penalization technique without increasing the rejection rate with more than x percent (0, 1, or 2%) compared to the original algorithm (black line) and the original algorithm with the trivial solution (red dashed line).

Figure 7-2 also shows the posterior trivial solution for these simulations. We notice that the posterior trivial solution does not decrease as much as in the artificial scenarios. The posterior trivial solution was calculated by adding extra rejections to nodes. However, in this figure, we show the resulting Gini index calculated for the clusters of nodes. We conclude that our heuristic that calculates the posterior trivial solution at the level of nodes does not necessarily minimize the Gini index at the level of regions.

7-2 Varying Cluster Size

In this section, we present experiments that were performed to analyze the sensitivity of the cluster size parameters t^p_{max} and t^G_{max}. We experiment with two different values for t^p_{max} for the penalty calculations, as well as five different values for t^G_{max} when calculating the resulting Gini indexes. We do this for two different penalization functions: one rejection penalization method and one trip penalization method.

The cluster parameter t^G_{max} is seen as exogenous. This would make sense in a real-life SMoD system as it is up to the operator or city council to decide how to divide an area into regions. It is futile to tune the resulting Gini index using t^G_{max}, most values for the Gini index can be achieved by choosing any value for t^G_{max}. However, this does not change anything to the underlying results of the simulation. On the other hand, it makes sense to tune the cluster

parameter t^p_{max}. This parameter should be tuned such that our methods provide the best results given the parameter t^G_{max}.

7-2-1 Rejection Penalization

In this subsection, we present the results obtained when using the R-Pen method using $\delta = 1, 5, 10$. We ran the Manhattan experiment using 625 vehicles and 25% of actual demand. The results can be seen in Figure 7-3. In this figure, we see that different values for t_{max} result in roughly similar rejection rates. Furthermore, we see that, for increasing t^G_{max}, the calculated Gini index decreases.

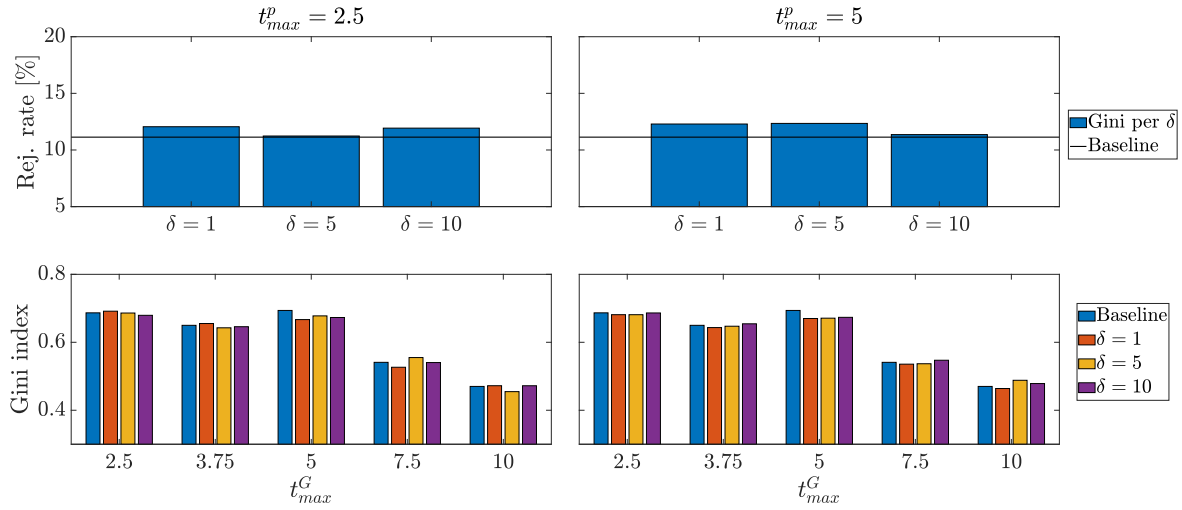


Figure 7-3: Comparison between rejection rate original framework and rejection rates obtained by the rejection penalization technique for different values of δ and different values of t^p_{max}. Also, the resulting Gini index is calculated using different values for t^G_{max}.

Again, this figure's results do not show the clearly defined pattern as in the artificial scenarios. This shows the difficulty of assessing the uniformity of the spread of the rejection rate for the Manhattan case. For some values for t^G_{max} the original method achieved the lowest Gini index, for other values our method did.

7-2-2 Trip Penalization

This subsection presents the same experiment as in the previous subsection but now using the T_2-Pen method. The results can be seen in Figure 7-4. Again, we notice that varying the value for t^p_{max} has a limited influence on the rejection rates since they are roughly equal. Similar to the results of the R-Pen method, the results do not show the clear pattern of the artificial scenario. The Gini index does not necessarily decrease for increasing λ, and the rejection rate does not decrease. Also, differing t^G_{max} changes which method or value for λ achieves the lowest Gini index.

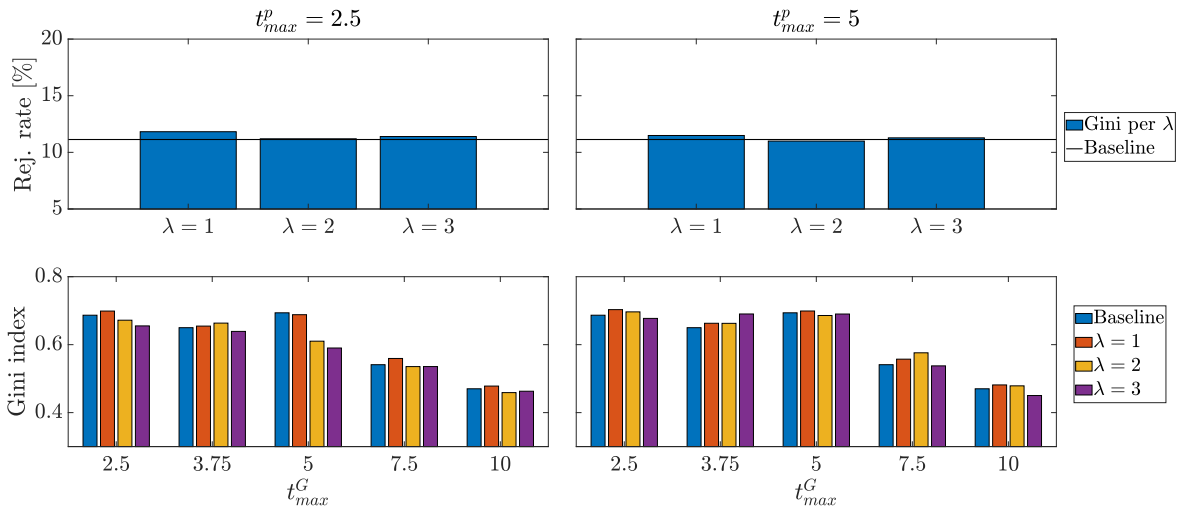


Figure 7-4: Comparison between rejection rate original framework and rejection rates obtained by the trip penalization technique for different values of λ and different values of t^p_{max}. Also, the resulting Gini index is calculated using different values for t^G_{max}.

7-3 Large-Scale Experiment

The experiments previously presented in this chapter can be considered to be experiments of a relatively small scale. In this section, we present the results of our methods for a real-life large-scale experiment. We simulated the Manhattan case study using 1500 vehicles and 60% of demand on March 7^{th} between 16:00 and 21:00 (52,802 requests). We started the simulation with empty vehicles at 16:00. We simulated the system using the original framework until 19:00, at which point we started using one of our penalization techniques. We report the results of the system between 20:00 and 21:00. The reasoning for this particular setup is similar to the argumentation given at the start of this chapter. We decided not to perform the full-scale experiment (3000 vehicles, 100% of demand) due to constraints on available computational power. We found that our machine needed exponentially more time to run such an experiment. Therefore, the only feasible option for running a simulation spanning multiple hours was to decrease the number of vehicles and demand. The results of the large-scale experiment can be seen in Figure 7-5. Here, the rejection rate and the Gini index for the different penalization techniques and the original framework are shown. Again, similar as in Section 7-1, the results do no show a clear trend. Repeating the experiments might mitigate this problem. However, due to these experiments' run times, this is not feasible within the time limits of this thesis.

The R method slightly increases the rejection rate compared to the original framework for all values of δ. However, it also reduces the Gini index for all values of δ. The absence of a clear pattern for different values of δ prohibits us from drawing any further conclusions as to what the ideal value for δ would be. The T_2 method only increases the rejection rate for $\lambda = 2$, for the other values, it achieves the same rejection rate. Only for the $\lambda = 1$ the Gini index is roughly the same, it is reduced for the other values. We can cautiously conclude that our T_2 method keeps the rejection rate roughly equal, but the Gini index is reduced. Lastly, the

RT_2 method provides us with a clear pattern. For increasing tuning parameters, the rejection rate is increased while the Gini index is decreased. However, as for the other methods in this experiment, we have to be careful with drawing any conclusion based on these results.

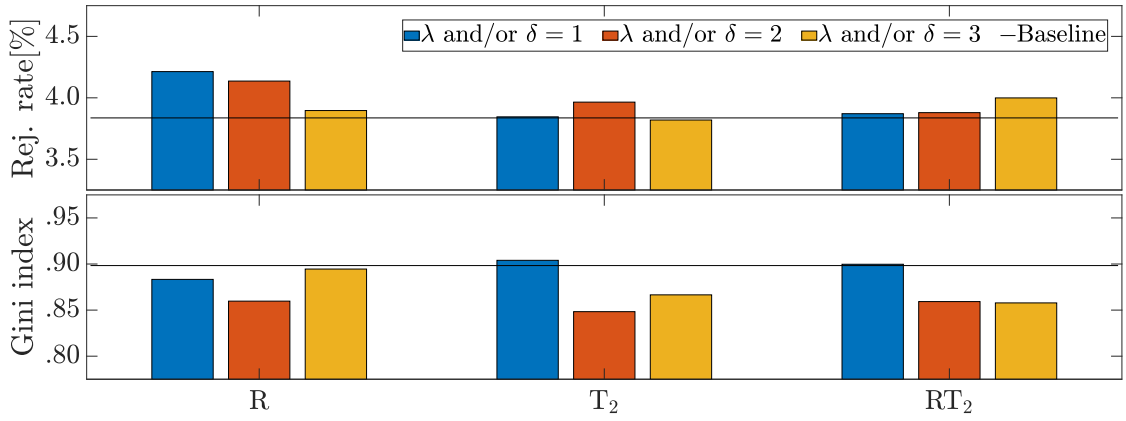


Figure 7-5: Comparison between rejection rate and Gini index of three different penalization techniques and the original framework, for the large-scale Manhattan case study using 1500 vehicles and 60% of demand. Penalties and Gini indexes are calculated using $t^p_{max} = t^G_{max} = 2.5$.

7-4 Concluding Remarks

The various experiments in this chapter have revealed that our penalization techniques perform irregularly in the Manhattan case study. The difficulty arises from the large number of nodes and highly skewed demand. We dealt with the large number of nodes by executing our methods at the level of regions. Similarly, the resulting Gini index was also calculated at the level of regions. Comparing Gini indexes using different regions resulted in different best performing techniques. Therefore, the regions for calculating the Gini index should be seen as exogenous. It is a futile exercise to define the best parameter for this. A method should be able to lower the Gini index no matter the chosen regions.

The experiments in this chapter seemed to be largely influenced by the experiment initialization. Repeating an experiment could lead to very different results. Unfortunately, due to time constraints on this work, these valuable repeat experiments could not be performed. Furthermore, we could only do a small brute force search for the best parameter settings. Therefore, we have to be careful when drawing conclusions based on our experiments. However, we have seen that our methods were able to achieve desired results. In many cases, our methods achieved a comparable rejection rate and a lower Gini index than the original framework. This was achieved by tuning the tuning parameters using brute force. We lack a robust method to find the best parameters settings more consistently. We suggest that finding ways to tune our methods more robustly for large-scale scenarios should be future research.

Chapter 8

Conclusions

The motivation for the writing of this thesis stemmed from repeating a New York case study using a state-of-the-art Shared Mobility-on-Demand (SMoD) Fleet Management Framework (FMF) [2]. In this repeat experiment, we noticed that the rejection rates are unevenly distributed over the operation area. Furthermore, in experimentation using artificial scenarios, we observed that this uneven distribution was caused by skewed demand and the layout of the street map. These observations led us to our research question; how can we add extra functionalities to the state-of-the-art FMF such that the rejection rates are spread more evenly? Being able to evenly spread rejection rates would be a useful functionality for real-life SMoD systems if this can be achieved. We hypothesized two potential benefits of lowering the Gini index: a social benefit and a performance benefit. First, the social benefit; customers might perceive the SMoD system as fair when service levels are evenly spread out over the operation area. Secondly, we observed that high rejection rates occur in high demand areas. Therefore, it could be beneficial for the system's overall performance if vehicles from low rejection rate areas are actively sent towards areas of high demand. These vehicles were near-idle in these low rejection rate areas; they might be put to better use in the high demand areas. We used the Gini index as a measure for how evenly spread out the rejection rates are.

Due to the complexity of SMoD systems, we decided to do our research using simulations. We implemented a Discrete Event Simulation in MATLAB. Using this simulation environment, we could compare the results obtained by the standard FMF of [2] and our altered version. The algorithms differed only in the assignment step, whereas the standard framework only took the total delay of trips into account, our algorithm also took the location of trips and/or requests into account. We adjusted the original framework in two ways. First, we developed the trip penalization method. This method decreased/increased the cost of a trip based on the difference between the average rejection rate of the entire operating area and the average rejection rate of the stops in the trip. If the average rejection rate of the operation area was lower, then the cost of the trip was decreased, and vice versa. Second, we developed the rejection penalization method. This method increased/decreased the penalty of rejecting a request based on the location of the request. This adjustment's height is based on the difference between the rejection rate at the origin of the request and the average operating

area's rejection rate. If the entire operation area's rejection rate was lower, then the penalty was increased, and vice versa.

As stated, we assessed our techniques' performance by benchmarking them against the standard framework in a simulation. We also created a method to fairly compare our methods against the results of the standard framework when the rejection rate of the standard framework was lower. We made this comparison by calculating the posterior trivial solution. This solution represents the lowest attainable Gini index for a given rejection rate. We experimented with our methods in artificial and real-life case studies. In the artificial case studies, we found that our methods can be tuned such that both the Gini index and the rejection rate decreased. Also, we found that our methods fared better in low rejection rate scenarios (either relatively few vehicles or requests). Furthermore, we found that our methods outperform the posterior trivial solution in most cases. The results for the New York case study were less definitive. The clear patterns we found in the results of the artificial cases were not visible. The outcomes of the experiments had high variability in the New York case study. Different initial setups lead to different results without a clear pattern. Furthermore, using regions to decide the height of the penalty increased the variability in the experiments. Using clusters also made the posterior trivial solution less meaningful, as it no longer represented the lowest attainable Gini index for a given rejection rate. However, our experiments still showed that, in some cases, using our methods had a positive impact on either the overall rejection rate or the Gini index.

When combining the results of the artificial scenario experiments and the New York case study, we conclude that the trip penalization method performs best. This technique achieved a comparable or lower rejection rate than the original FMF of [2], while consistently lowering the Gini index. The rejection penalty method performed second best. Using this method more often resulted in increasing the rejection rate. Lastly, the combined method performed worse. This method was overly aggressive and increased the rejection rate too much. Information stored in the original weights of the objective function was lost when using the combined method. For all methods, the tuning parameters performed as indented. This was best visible in the results of the artificial experiments. Increasing the tuning parameter resulted in increasing the rejection rate but lowering the Gini index. This research lacked a robust method to tune the tuning parameters for the real-life case study. Further research should be aimed at adjusting our methods to such scenarios. The sparsely distributed demand is the main issue in these scenarios. We solved this issue by aggregating nodes into clusters. However, other methods might do better. Furthermore, different settings for the tuning parameters might be desired throughout the day. In our experiments, we kept the tuning parameters the same. Further research should be aimed at researching methods to automatically adjust these parameters based on the system's general state.

Fleet management algorithms for SMoD systems have usually been aimed at optimizing a one-dimensional performance indicator. Profit and service level were the most common objectives. In this thesis, we have shown that the objective can be extended to a broader goal. One-dimensional metrics cannot capture a highly complex system such as an SMoD system. One should assess the performance of a system based on the performance in individual regions within the operation area. Doing very good overall but very bad in one specific region does not constitute a good performance. We have shown that it is possible to extend existing methods to aim for a broader objective. Moreover, our idea can be extended such that the system strives for a certain distribution of the service level. This distribution can be designed

such that it nicely complements the fixed-line public transport in an area. For instance, the service levels in areas that are hard to reach using public transport can be increased. On the other hand, areas that can be easily accessed by public transport might require lower service levels. It might also be desired to offer higher service levels in low-income communities since many households do not own a private car. In the end, following this line of research gives governments the tools to make efficient transportation available in all areas of the city.

Appendix A

Steady-State Artificial Scenario

Figure A-1 shows the most important performance metrics for a simulation of an SMoD system. The metrics are shown over time. We use these metrics to decide at what time step our system has reached a steady-state. We decided to perform our artificial experiments for a total of 100 hours. This decision was mostly based on the Gini index (left-top plot), as the other metrics stabilize much earlier. The Gini index has almost stopped decreasing at this point.

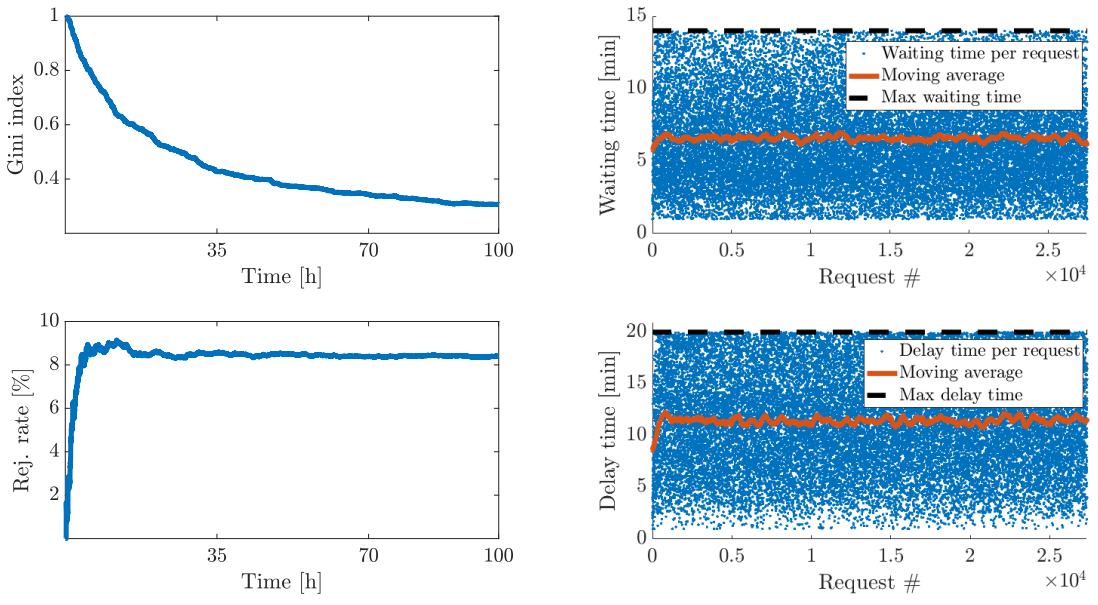


Figure A-1: Overview of important metrics example simulation artificial scenario. Demand pattern '20C2S' was used in this case. 65 vehicles were used and 5 requests were drawn per minute.

Appendix B

Disregarded Ideas

Throughout this thesis, we present the results of a limited number of methods. These methods were selected after experimenting with various functions in 'quick and dirty' experiments. Due to time limitations, the experiments were not documented. The experiments were merely used to quickly point at in promising direction. After these experiments, we thoroughly experimented with the most promising functions (Chapter 6 and Chapter 7). We decided to include the disregarded ideas in an appendix to still give the reader an understanding of what ideas we have tried throughout this study. We divide the methods again into two categories; 1) rejection penalization, and 2) trip penalization.

The different functions all look somewhat similar. However, their effects are vastly different. Especially because the tuning parameters impact most functions differently. Throughout our experimentation phase, we have taken into account these differences and adjusted the parameters accordingly. Furthermore, we have also experimented with combining different ideas. The techniques presented in the following subsections are merely the most basic forms.

B-1 Rejection Penalization

The different rejection penalization functions we experimented with can be roughly divided into two categories: 1) adjusting the original value in both directions (increase/decrease), and 2) adjusting the original value in one direction.

B-1-1 Both Directions

First, the functions that adjust the rejection penalty in both directions. The R-method we thoroughly experimented within this thesis belongs to this category. However, we also experimented with rejection penalty functions that had no lower bound:

$$c_{ko}^{k} = p_{KO} + \delta \cdot \Delta_{r_k}, \tag{B-1}$$

functions that had a lower bound of zero:

$$c_{ko}^{k} = \max\left(0,\, p_{KO} + \delta \cdot \Delta_{r_k}\right), \tag{B-2}$$

or variable lower bounds:

$$c_{ko}^{k} = \max\left(\frac{p_{KO}}{\delta},\, p_{KO} + \delta \cdot \Delta_{r_k}\right). \tag{B-3}$$

For many values of δ and Δ_{r_k} these functions are the same. However, the SMoD systems behaved differently as a result of the extreme values. We found that the system's performance started to take a turn for the worse as soon as rejecting a request became cheaper than the cost of performing a trip. The more often this occurred, the larger the effect.

B-1-2 One Direction

We also experimented with rejection techniques that altered the original penalty value, p_{KO}, in just one direction. For instance, we experimented with only increasing the original penalty:

$$c_{ko}^{k} = \max\left(p_{KO},\, p_{KO} + \delta \cdot \Delta_{r_k}\right). \tag{B-4}$$

The above function had minimal effect on the system. We hypothesized that, since the rejection penalty is already much larger than the trip costs, increasing the penalty even further has no effect. The opposite function:

$$c_{ko}^{k} = \min\left(p_{KO},\, p_{KO} + \delta \cdot \Delta_{r_k}\right) \tag{B-5}$$

suffered from the problem that the penalty could become lower than the cost of certain trips. Again, this had a large negative effect on system performance. To counter this, we also experimented with:

$$c_{ko}^{k} = \max\left(\max_{ij}\{c_{ij}\},\ \min\left(p_{KO},\, p_{KO} + \delta \cdot \Delta_{r_k}\right)\right). \tag{B-6}$$

The above function performed similarly to the R-method. We eventually opted to further experiment with the R-method due to its simplicity.

B-2 Trip Penalization

The trip penalty functions we experimented with can be divided into the same categories as the rejection penalty functions. Moreover, a third category is introduced: scaling methods. We have also experimented with combining scaling methods with any method in the first two categories.

B-2-1 Both Directions

Adjusting the trip costs in both directions is what happens in the T_P-method. We have experimented with several other functions, such as a function with no lower bound:

$$c_{ij}^{new} = c_{ij} - \lambda \cdot \Delta_{T_i}, \tag{B-7}$$

a function with a lower bound of zero:

$$c_{ij}^{new} = \min\left(0,\, c_{ij} - \lambda \cdot \Delta_{T_i}\right), \tag{B-8}$$

or a function with no lower bound but with an upper bound:

$$c_{ij}^{new} = \min\left(p_{KO},\, c_{ij} - \lambda \cdot \Delta_{T_i}\right). \tag{B-9}$$

It depends on the tuning parameter values and Δ_{T_i} whether these functions differ from each other or from the T_P-method. We found that discounting trips too much resulted in poor system performance. In those cases, too much of the original information of the weight (the sum of delays) is lost. Increasing trip costs too much seemed to have a less significant negative effect.

B-2-2 One Direction

We also experimented with adjusting the trip costs in just one direction. For instance, only increasing the trip cost:

$$c_{ij}^{new} = \min\left(c_{ij},\, c_{ij} - \lambda \cdot \Delta_{T_i}\right), \tag{B-10}$$

or only decreasing the trip cost:

$$c_{ij}^{new} = \max\left(c_{ij},\, c_{ij} - \lambda \cdot \Delta_{T_i}\right). \tag{B-11}$$

For these functions, we found that the effects on the system were not as large compared to the functions that adjusted the cost in both directions. Both the rejection rate and the Gini index were barely affected.

B-2-3 Scaling

After experimenting with different trip penalization functions, we noticed that trip costs were relatively adjusted differently. An expensive trip could be adjusted by an equal amount as an inexpensive trip. We hypothesized that this practice would result in a loss of original information of the weights. Therefore, we experimented with different functions that scaled the penalty for a trip with the original trip cost. For instance, the scaled version of the T_2-method would be:

$$c_{ij}^{new} = \max\left(\frac{c_{ij}}{2},\, c_{ij} - \lambda \cdot \Delta_{T_i} \cdot c_{ij}\right). \tag{B-12}$$

We found that this offered no significant improvement over the standard T_P-method, we therefore, opted not to use it due to its added complexity.

Appendix C

Repeated Experiment

Figure C-1 shows the average results (three runs) of the Manhattan case study using 375 vehicles and 15% of actual demand. The figure illustrates that the irregular patterns, as described in Subsection 7-1, cancel out when repeating an experiment multiple times. In most cases, the rejection rate increases for increasing tuning factors and the Gini index decreases. The rejection rates using the R-method still seem to be irregular. We expect that running even more experiments would eventually also erase this irregularity.

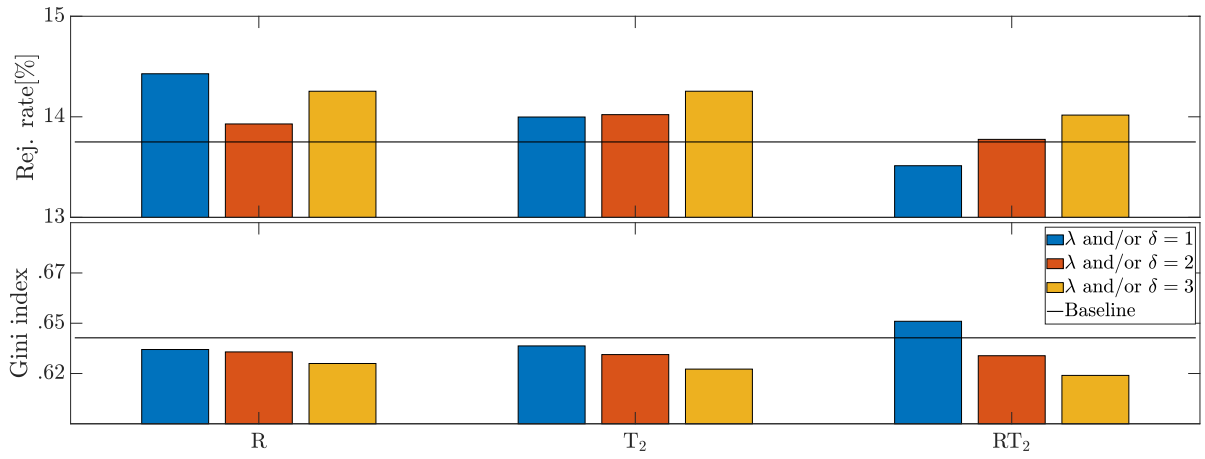


Figure C-1: Average results over 3 experiments for Manhattan case study using 375 vehicles and 15% of actual demand.

Appendix D

Heuristic Posterior Trivial Solution

In this appendix, we explain how we obtained the posterior trivial solution given an allowed increase of $x\%$ in rejection rate. The variables and their definitions we use throughout this appendix can be seen in Table D-1. The heuristic we created to calculate the posterior trivial can be seen in Algorithm 1.

Table D-1: Variables and definitions used in Algorithm 1.

Variable	Definition
r_i	Number of rejections at node i
t_i	Total number of requests at node i
A	Maximum number of allowed added rejections
n	Number of nodes
a_i	Decision variable: number of artificially added rejections at node i

Data: r, t, A
Result: G^*
$addedNodes = 0$;
while $addedNodes < A$ **do**
  $r^c = r$;
  $t^c = t$;
  $i_1 = \arg\min_i \left(\frac{r_i}{t_i}\right)$;
  $nodeFound = 0$;
  while $nodeFound = 0$ **do**
    if $\exists i^*$ **then**
      $r^c_{i*} = [\,]$;
      $t^c_{i*} = [\,]$;
    end
    $i^* = \arg\min_i \left(\frac{r^c_i}{t^c_i}\right)$;
    if $\frac{r^c_{i^*}+1}{t^c_{i^*}} < \frac{1}{n}\sum_{i=1}^{n} \frac{r_i}{t_i}$ **then**
      $nodeFound = 1$
    end
    if $r^c = [\,]$ **then**
      $i^* = i_1$;
      $nodeFound = 1$;
    end
  end
  $r_{i^*} = r_{i^*} + 1$;
  $addedNodes = addedNodes + 1$;
  $i^* = [\,]$;
end
$G^* = gini(r_i, t_i)$

Algorithm 1: Heuristic to solve (6-2).

Bibliography

[1] L. Abbatecola, M. P. Fanti, and W. Ukovich. A review of new approaches for dynamic vehicle routing problem. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 361–366, 2016.

[2] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.

[3] J. Alonso-Mora, A. Wallar, and D. 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, 2017.

[4] J. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Science & Business Media, 2011.

[5] J. Bischoff, M. Maciejewski, and K. Nagel. City-wide shared taxis: A simulation study in Berlin. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 275–280, 2017.

[6] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

[7] M. Cáp, J. Alonso-Mora, H. Kress-Gazi, S. Srinivasa, T. Howard, and N. Atanasov. Multi-objective analysis of ridesharing in automated mobility-on-demand. In *Robotics: Science and Systems*, 2018.

[8] R. Chen and C. G. Cassandras. Optimization of ride sharing systems using event-driven receding horizon control. *arXiv e-prints*, page arXiv:1901.01919, 2019.

[9] X. Chen, F. Miao, G. J. Pappas, and V. Preciado. Hierarchical data-driven vehicle dispatch and ride-sharing. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 4458–4463, 2017.

[10] J. F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[11] J.F. Cordeau, G. Laporte, and S. Ropke. Recent models and algorithms for one-to-one pickup and delivery problems. In *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 327–357. 2008.

[12] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[13] B. Donovan and D. B. Work. Empirically quantifying city-scale transportation system resilience to extreme events. *Transportation Research Part C: Emerging Technologies*, 79:333–346, 2017.

[14] P. d'Orey, R. Fernandes, and M. Ferreira. Empirical evaluation of a dynamic and distributed taxi-sharing system. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 140–146, 2012.

[15] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991.

[16] Jianren Gao, Yuxin Wang, Haoyang Tang, Zhao Yin, Lei Ni, and Yanming Shen. An efficient dynamic ridesharing algorithm. In *2017 IEEE International Conference on Computer and Information Technology (CIT)*, pages 320–325. IEEE, 2017.

[17] C. Gini. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche. [Fasc. I.]*. Tipogr. di P. Cuppini, 1912.

[18] H. Hosni, J. Naoum-Sawaya, and H. Artail. The shared-taxi problem: Formulation and solution methods. *Transportation Research Part B: Methodological*, 70:303–318, 2014.

[19] W. Hu. Your uber car creates congestion. should you pay a fee to ride? *The New York Times*, 2017.

[20] X. Huang and H. Peng. Efficient mobility-on-demand system with ride-sharing. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3633–3638, 2018.

[21] J. Jung, R. Jayakrishnan, and J. Y. Park. Dynamic shared-taxi dispatch algorithm with hybrid-simulated annealing. *Computer-Aided Civil and Infrastructure Engineering*, 31(4):275–291, 2016.

[22] A. Lam, Y.W. Leung, and X. Chu. Autonomous vehicle public transportation system. In *2014 International Conference on Connected Vehicles and Expo (ICCVE)*, pages 571–576, 2014.

[23] A. Lam, Y.W. Leung, and X. Chu. Autonomous-vehicle public transportation system: scheduling and admission control. *IEEE Transactions on Intelligent Transportation Systems*, 17(5):1210–1226, 2016.

[24] Y. Lin, W. Li, F. Qiu, and H. Xu. Research on optimization of vehicle routing problem for ride-sharing taxi. *Procedia-Social and Behavioral Sciences*, 43:494–502, 2012.

[25] Y. Liu and S. Samaranayake. Proactive rebalancing and speed-up techniques for on-demand high capacity ridesourcing services. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–8, 2020.

[26] M. O. Lorenz. Methods of measuring the concentration of wealth. *Publications of the American Statistical Association*, 9(70):209–219, 1905.

[27] S. Lotfi, K. Abdelghany, and H. Hashemi. Modeling framework and decomposition scheme for on-demand mobility services with ridesharing and transfer. *Computer-Aided Civil and Infrastructure Engineering*, 34(1):21–37, 2019.

[28] K. Lucas. Transport and social exclusion: Where are we now? *Transport Policy*, 20:105 – 113, 2012.

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

[30] S. Ma, Y. Zheng, and O. Wolfson. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1782–1795, 2015.

[31] NYC Taxi and Limousine Commission. 2014 yellow taxi trip data. Retrieved from https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

[32] Masayo O., H. Vo, C. Silva, and J. Freire. Stars: Simulating taxi ride sharing at scale. *IEEE Transactions on Big Data*, 3(3):349–361, 2016.

[33] C. Riley, A. Legrain, and P. Van Hentenryck. Column generation for real-time ridesharing operations. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 472–487, 2019.

[34] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. Strogatz, and C. Ratti. Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences*, 111(37):13290–13294, 2014.

[35] M.A. Sen and J. E. Foster. *On economic inequality*. Oxford University Press, 1997.

[36] B. Shen, B. Cao, Y. Zhao, H. Zuo, W. Zheng, and Y. Huang. Roo: Route planning algorithm for ride sharing systems on large-scale road networks. In *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 1–8, 2019.

[37] F. Siddiqui. Falling transit ridership poses an 'emergency' for cities, experts fear. *The Washington Post*, 2018.

[38] A. Simonetto, J. Monteil, and C. Gambella. Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C: Emerging Technologies*, 101:208–232, 2019.

[39] Texas AM Transportation Institute. 2019 urban mobility report. 2019.

[40] M. Tsao, D. Milojevic, C. Ruch, M. Salazar, E. Frazzoli, and M. Pavone. Model predictive control of ride-sharing autonomous mobility on demand systems. In *IEEE Conference on Robotics and Automation*, 2019.

[41] United Nations, Department of Economic and Social Affairs, Population Division. World urbanization prospects: The 2018 revision. 2018.

[42] S. Varone and V. Janilionis. Insertion heuristic for a dynamic dial-a-ride problem using geographical maps. In *MOSIM 2014, 10ème Conférence Francophone de Modélisation, Optimisation et Simulation*.

[43] A. Wallar, M. Van Der Zee, J. Alonso-Mora, and D. Rus. Vehicle rebalancing for mobility-on-demand systems with ride-sharing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4539–4546, 2018.

[44] Y. Wang, B. Zheng, and E.P. Lim. Understanding the effects of taxi ride-sharing – a case study of Singapore. *Computers, Environment and Urban Systems*, 69:124–132, 2018.

Glossary

List of Acronyms

C2S	Centre-2-Surroundings
DES	Discrete Event Simulation
FMF	Fleet Management Framework
ILP	Integer Linear Program
L2R	Left-2-Right
OC	Operation Cost
QoS	Quality of Service
RAND	Random
SMoD	Shared Mobility-on-Demand
VRP	Vehicle Routing Problem