A Column Generation Heuristic for the Dynamic Rebalancing Problem in Bike Sharing Systems

Marte Dybendal Gleditsch
Kristine Hagen

Industrial Economics and Technology Management

Submission date:	June 2018
Supervisor:	Kjetil Fagerholt, IØT
Co-supervisor:	Henrik Andersson, IØT

Norwegian University of Science and Technology
Department of Industrial Economics and Technology Management

Problem Description

The purpose of this thesis is to model and implement a solution method capable of solving realistic problem sizes of the Dynamic Bicycle Rebalancing Problem (DBRP) for Bike Sharing Systems (BSS). A BSS consists of self-service rental stations distributed around the city. Users can rent a bicycle at a station, ride it, and lock it at a station nearby their destination. Bike sharing is offered to the customers as long as there are available bicycles and locks. However, due to customer interactions and unpredictable demand, stations get empty or full regularly, and some customers are left unsatisfied. A set of service vehicles are utilized to re-distribute the bicycles and to maintain a balanced system. The DBRP involves determining optimal routes and loading quantities for the service vehicles.

Preface

This master thesis is part of our Master in Science at the Norwegian University of Science and Technology, Department of Industrial Economics and Technology Management. The thesis is written in the spring of 2018 as a part of the master project in TIØ4905 Applied Economics and Operations Research.

We would like to thank our supervisors Kjetil Fagerholt and Henrik Andersson for valuable guidance. Further, we would like to thank Urban Infrastructure Partners for a good partnership, inspiring thoughts and for providing us with necessary data.

Trondheim, 01.06.2018
Kristine Hagen & Marte Dybendal Gleditsch

Abstract

This thesis examines the dynamic rebalancing of a bike sharing system (BSS). A BSS is a service where bicycles are made available to users on a short-term basis. However, imbalanced systems are a significant challenge and often results in unmet customer demand. A set of service vehicles are utilized to re-distribute the bicycles and to maintain a balanced system. The primary objective of this thesis is to implement a model that generates optimal rebalancing strategies for the service vehicles with the aim of reducing violated demand. The BSS in Oslo, operated by Urban Infrastructure Partners (UIP), is used as a sample case in this thesis. As the customer demand is unknown, the real-world problem is both dynamic and stochastic. In our solution method, the problem is simplified by approximating it into a set of smaller deterministic subproblems where the customer demand is assumed known. These smaller subproblems are defined as Dynamic Deterministic Bicycle Rebalancing Subproblems (DDBRS).

Through a comprehensive literature survey, it becomes evident that it is insufficient to use exact solution methods to solve the DDBRS of realistic sizes, and heuristic approaches are necessary. However, there is a lack of efficient heuristic solution algorithms for solving the DDBRS. Column generation heuristics applied to BSSs are not discovered in literature, despite its success on vehicle routing problems (VRPs). Because of the enormous amount of variables in the DDBRS, a column generation heuristic may be appropriate.

An arc-flow model is formulated and differs from others in the literature as we allow the service vehicles to initiate a trip that exceeds the time horizon. By doing this, the idle time of the service vehicle at the end of the period reduces, and a rebalancing strategy beneficial in the long run can be initiated instead of a shorter disadvantageous strategy. Additionally, the service vehicles strive to fulfill an optimal station bicycle level. This inventory level accounts for future demand. Altogether, these aspects incorporate a long-term focus.

A column generation heuristic is developed and consists of an initialization procedure, a master problem, and a pricing problem. We propose three different variations of the master problem. The amount of information predetermined differs for the three master problems. In version 1, the loading quantities are determined in the master problem; in version 2, the loading quantities are predetermined in the initialization and re-evaluated in the master problem; and in version 3, the loading quantities are entirely predetermined. The initialization process consists of a branching algorithm that generates many initial columns, i.e., routes, for each service vehicle, while the master problem determines the optimal combination of columns by allocating one column to each service vehicle. A heuristic pricing problem is developed with the goal of generating new and better columns. For diversification of routes, a model for clustering of stations is implemented.

Parameter tuning for the subproblem is conducted, and the parameters are set to their best-performing values. We observe that the computational time of the subproblem is highly dependent on the branching constant in the initialization, the number of possible visits to a station within the time horizon, and the number of service vehicles. Multiple visits to a station lead to drastic increases in computational time and only a marginal improvement in solution, hence, multiple visits are concluded to be unnecessary. Additionally, the different versions of the master problem are compared. The results shows that version 3, with predetermined loading quantities, outperforms the other versions.

As the subproblem does not account for real-world uncertainties, the results are simulated using an implemented discrete-event simulator. A system configuration is evaluated by observing the average total violations from ten randomly drawn demand scenarios generated based on historical data. Some parameters are re-tested in a dynamic setting, and the final configurations for the column generation heuristic are set. Further, operational insights are gathered, and are intended as a basis for strategic decision-making. Our heuristic is compared to UIP's current rebalancing method, and a reduction in the total violations of 31% is observed. The effect of various strategic decisions regarding the BSS are analyzed, e.g. the number of service vehicles and bicycles, the size of the service vehicles, different prioritization of starvations and congestions, and the effect of enabling geo-fencing. By implementing our column generation heuristic, allowing geo-fencing and increasing the number of bicycles in the system, the total violations can decrease with as much as 81% compared to today's current rebalancing strategies.

Sammendrag

Denne oppgaven tar for seg det dynamiske sykkelflyttingsproblemet til et bysykkelsystem (BSS). Et bysykkelsystem tilbyr en tjeneste hvor sykler er gjort tilgjengelig for kortvarig utlån. En sentral utfordring med BSS er at stasjonene hvor syklene står parkert fort blir fulle eller tomme, noe som fører til at brukeretterspørsel ikke blir møtt. Servicebiler brukes i dag til å redistribuere syklene slik at balansen i systemet forbedres. Formålet med denne oppgaven er å utvikle og implementere en model som genererer optimale rebalanseringsstrategier for servicebilene slik at udekket etterspørsel blir redusert. Oppgaven tar utgangspunkt i bysykkelsystemet i Oslo som er driftet av Urban Infrastructure Partners (UIP). Siden etterspørsel er uforutsigbart, er det faktiske problemet både dynamisk og stokastisk. I vår modell er problemet forenklet ved at det er delt inn i et sett av mindre deterministiske subproblemer hvor en antar kjent etterspørsel. Dette defineres som det dynamisk deterministiske sykkelflyttingssubproblemet (DDBRS).

Gjennom et omfattende litteraturstudie, viste det seg at eksakte løsningsmetoder er utilstrekkelige når problemer av realistisk størrelse skal løses, og at heuristiske algoritmer derfor bør utvikles. Det er derimot mangel på effektive heuristisker for det DDBRS. Hypotesen vår er at en kolonnegenereringsheuristikk kan være passende på grunn av den store mengden variabler i et DDBRS. Kolonnegenereringsheuristikker anvendt på et BSS er ikke oppdaget i litteraturen.

Den matematiske modellen skiller seg fra andre ved at det siste stasjonsbesøket for hver servicebil kan forekomme etter tidshorisonten. Dette gjør at inaktivitet blant servicebilene reduseres, og rebalanseringsstrategier som er bedre på lang sikt genereres til fordel for kortsiktige strategier. Servicebilene prøver i tillegg å opprettholde et optimalt lagernivå av sykler på hver stasjon. Dette lagernivået tar hensyn til fremtidig etterspørsel.

En kolonnegenereringsheuristikk er utviklet og består av en initialiseringsprosess, et masterproblem og et prisproblem. Tre ulike versjoner av masterproblemet er presentert hvor forskjellen er mengden av informasjon som blir forhåndsbestemt og brukt som input i masterproblemet. I versjon 1 bestemmes lastemengder i masterproblemet, i versjon 2 bestemmes de i initialiseringen, men masterproblemet har til en viss grad mulighet tli å endre det, og i versjon 3 er lastemengdene satt i initialiseringen. Initialiseringen omfatter en forgreningsalgoritme som genererer initielle kolonner, dvs. ruter, for hver servicebil. Masterproblemet bestemmer deretter den optimale kombinasjonen av kolonner, og tildeler hver servicebil én rute. Et heuristisk prisproblem er utviklet med mål om å generere nye og bedre kolonner. I tillegg, er en klyngealgoritme for soneinndeling av stasjoner utviklet for å oppnå diversifisering av ruter.

Parameterkalibrering for subproblemet er gjennomført, og parameterene er satt til de verdiene som resulterer i de beste løsningene. Vi observerer at kjøretiden til subproblemet er svært avhengig av antall forgreninger i initialiseringen, antall mulige besøk en stasjon kan få innenfor tidshorisonten, og antall servicebiler. Videre er de forskjellige versjonene av masterproblemet testet opp mot hverandre, og det vises at versjon 3, med forhåndsbestemte lastemengder, er overlegen i forhold til de andre.

Siden subproblemet ikke tar høyde for usikkerheter i den virkelige verden, så er resultatene simulert ved bruk av en diskret hendelsessimulator. En systemkonfigurasjon blir evaluert ved å observere det gjennomsnittlige antallet brukeretterspørsler som ikke er tilfredsstilt ved ti tilfeldig trekte etterspørselscenarier generert basert på historisk data. Noen parametere er re-kalibrert i en dynamisk setting og den endelige konfigurasjonen for kolonnegenereringsheuristikken er satt. Videre samles en mengde operasjonell innsikt. Denne innsikten er ment som et beslutningsgrunnlag for UIP når de skal ta strategiske valg. Vår kolonnegenereringsheuristikk er sammenliknet med rebalanseringsmetodene som brukes av UIP i dag, og en 31% reduskjon av mengden umøtt etterspørsel er observert. Effekten av å variere strategiske valg er analysert, som for eksempel antall sykler og service biler, størrelsen på servicebilene, forskjellig vektlegging av umøtt etterspørsel for sykler og låser, og et konsept kalt geo-fencing som tillater overvekt av sykler på stasjoner. Dersom UIP implementerer vår kolonnegenereringsheuristikk, tillater geo-fencing og øker antall sykler i systemet kan de redusere forventningsverdien av mengden umøtt etterspørsel med opp til 81%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to urbanization, the amount of private motorized traffic increases in cities all over the world, resulting in traffic congestion and environmental pollution. In response to the growing concerns, the interest for green transportation methods, such as bike sharing, has emerged (Brinkmann et al., 2015). The concept of a BSS is simple: a user picks up an available bicycle at a docking station, rides it to their destination, and locks the bicycle at a nearby station. Bike Sharing Systems (BSSs) are widely adopted, and today, as of June 2018, there are 1,608 active systems, and 391 systems under construction (Meddin, 2018). The BSSs have grown considerably in size in the recent years, and are becoming an essential part of their cities public transport systems (Pfrommer et al., 2014).

The primary operation issue for most BSS, is that stations regularly get empty or full due to customer interactions, leading to imbalanced systems and violated demand. To increase the balance in the system, bicycles are re-distributed with *service vehicles* specially designed to transport bicycles. This task is called the *bicycle rebalancing problem*.

The rebalancing problem has become a popular topic in operational research. The rebalancing problem consists of two main objectives; finding optimal routes for the service vehicles, and determining optimal loading quantities at the stations they visit. These decisions form what we call a *rebalancing strategy*. Finding the optimal rebalancing strategies is a fundamental challenge for all BSSs, as the number of possible solutions is vast and the customer demand is uncertain. There is a lack of efficient solution algorithms for BSSs in literature, especially when instances of realistic size are considered. The BSS in Oslo, operated by Urban Infrastructure Partners (UIP), is used as a sample case in this paper. Based on data acquired from UIP, we observe that each station in Oslo is, on average, empty 22% of the time and full 6% of the time. In the morning hours, between 7-9am, these numbers are as high as 26% and 7.4%, respectively. This emphasizes the need for better rebalancing strategies.

The purpose of this master thesis is to develop a solution algorithm that is capable of solving the dynamic bicycle rebalancing problem with large instances. Due to the complexity and large solution space in bicycle rebalancing problems, it is inefficient to use exact solution methods. Hence, the need for a heuristic solution approach is indisputable. The main contributions of this thesis are thus:

- A literature survey on heuristic solution methods applied to BSSs and column generation heuristics applied to VRPs.
- A mathematical model of the dynamic deterministic subproblem.
- A column generation heuristic capable of solving problems of realistic size.
- A simulation framework used to evaluate rebalancing strategies.
- Operational insight of strategic decisions relevant for decision-making at UIP.

The rest of the paper is organized as follows. First, Chapter 2 presents a study on how the bike sharing concept has developed through the years. The largest BSS in Norway is presented, together with BSSs in selected cities around the world. At the end of the chapter, some key challenges are discussed. In Chapter 3, a literature survey on heuristic solution methods applied to BSSs and column generation heuristic applied to VRPs are presented.

Chapter 4 contains a detailed description of the real-world stochastic problem. A discussion regarding the challenges with this problem is presented, and an approximation is suggested. Different aspects and terminology are elaborated, and the main restrictions, decisions and problem assumptions are conferred. In Chapter 5, the mathematical model is presented with a description of all necessary constraints.

Chapter 6 presents a detailed description of our proposed heuristic solution algorithm. Chapter 7 presents the test instances and the key input data used in our implementation. In addition, our interpretation and implementation of UIP's current rebalancing strategies are presented. Chapter 8 includes a detailed description of the simulation framework. In Chapter 9, parameter tuning is conducted, and the different versions of the heuristic are compared. Chapter 10 presents the results obtained from simulating how the column generation heuristic performs with real-world uncertainty. In addition, the effect of various strategic decisions regarding the BSS are analyzed. At the end of the thesis, concluding remarks and future research opportunities are presented in Chapter 11.

Chapter 2

Background

Extensive usage of private vehicles in urban areas has led to increased traffic congestion, carbon emissions, and usage of non-renewable resources. These concerns have led to emerging attention towards BSSs as an answer to lessening the environmental impacts of transportation, and the public's desire to increase bicycle usage as a mean for everyday transport (Ghosh et al., 2015). Additionally, bicycles contribute to public health; they can reach underserved destinations; they help tourist explore new cities, and they are relatively inexpensive and affordable to maintain (DeMaio, 2008). In Section 2.1, the bike sharing concept is introduced. The system's history and a presentation of newer variations of the system are elaborated in Section 2.2 and 2.3, respectively. Descriptions regarding how bike sharing is operated in Norway and other countries are presented in Section 2.4 and 2.5. Challenges with a BSS are discussed and presented in Section 2.6.

2.1 Bike Sharing Concept

A BSS is a service where bicycles are made available to users on a short-term basis. Bike sharing is a quick and comfortable mode of transportation used to fill the gaps in public transit, making the whole intracity mobility offer better. A modern BSS has self-service rental stations distributed within the city center, where each station has a finite number of locks. Some make use of mobile applications that provide the users with an overview of available bicycles and locks. For a customer to be able to rent a bicycle or return a rented bicycle, there must be an available bicycle or lock, respectively, at the station. Due to unpredictable demand, there is often either *congestion*, i.e. no available locks at the station, or *starvation*, i.e. no available bicycles. In this paper, both of these scenarios are referred to as *violations*. To avoid violations, the bike sharing companies use service vehicles to redistribute the bicycles.

Payment schemes vary between the systems; some may be free of charge, have periodic subscription fees, deposit payment or a fee where the amount is dependent on the length of the rental. Short journeys are often encouraged by charging the customers an amount proportional to the length of the lease. The common feature is that it is an affordable option, compared to public transport or driving a private car. Most BSSs are still unable to adequately cover their investment and operational costs entirely from customer fees. Thus, the BSSs are usually administrated by advertising companies, such as Clear Channel and JCDecaux, or relying on government subsidies, in addition to user payments (DeMaio, 2008).

For most BSSs, the rebalancing operation is the most significant operating expense. However, the capital cost, such as installment of stations and bicycle investments, is also substantial. There are also costs related to maintenance of bicycles, stations, and IT infrastructure. (Andersen, 2016).

2.2 History

The concept of bike sharing is said to have started in Amsterdam in 1965. A group of Dutch activists introduced the so-called “Witte Fietsen,” i.e. the White Bikes, as a mean to revolutionize public transport and counter the rise of air pollution and cars. They gathered together a dozen bicycles, painted them white and left them unlocked around the city for anyone to use free of charge. Sadly, the project was unsuccessful. Not only did most of the bicycles get stolen or broken, but the city council rejected the plan as they meant bicycles belonged to the past, whereas cars were the future. In spite of considerable adversity, Luud Schimmelpennink, the man behind the "White bikes," did not let the idea slip by. He reintroduced the concept in 1975, ten years later, with a few electric cars, one station, and with user memberships. Unfortunately, this project also failed due to financial circumstances (Van der Zee, 2016).

Schimmelpennink never stopped believing in the power of a BSS, so when a couple of Danes asked for his help to build a BSS in Copenhagen thirty years later, he was easy to persuade. The result was the world's first large-scale BSS with designated docking stations and a coin-deposit system (Goodyear, 2015). The users would drop a coin on the bicycle, and when returned, they got their coin back. Similar to the White Bikes in Amsterdam, a lot of bicycles got stolen since the user could remain anonymous (Van der Zee, 2016).

The first system attempting to overcome the bicycle theft problem was a small BSS at Portsmouth University in 1996. They utilized the technological improvements by introducing the use of an individual, magnetic swipe card. This way the students with an unreturned bicycle were identified (Clemitson, 2017). Similarly, a BSS in Rennes, France, called Vélo à la Carte, adapted this idea two years later. In addition to tracking the users, Vélo à la Carte used the individual cards and Radio Frequency Identification Technology (RFID) to gather information about the users' cycling patterns. Hence, they were able to distribute the bicycles to suit the users' needs better. The system was also characterized by electronic docking stations and mobile phone access (Clemitson, 2017).

The BSSs underwent significant development from 1965 to 1998 but were still in relatively small scale. When Vélo'v, the smart bike sharing system in Lyon, France, was launched in 2005, it was with a whole new scale with 1,500 bicycles. Therefore, Vélo'v is said to be the catalyst of the urban BSS. Despite this, it is the Parisian BSS, Vélib', introduced two years later and inspired by Vélo'v, which has become the benchmark for all successful BSSs, with a total of 18,000 bicycles and 1,230 stations (Clemitson, 2017).

After 2007, an increasing number of cities around the world have adopted the concept of bike sharing. The system first settled in Europe, but today China has become the world leader when it comes to the total number of bicycles (Gray, 2017). The BSSs have embraced the technology developed through the years, and especially mobile applications, smart card systems, Global Position System (GPS), electric bicycles and machine learning.

2.3 New Variations of the Bike Sharing System

Today, the most usual and traditional type of a BSS is where the bicycles are locked to strategically placed docking stations throughout the city. Users can collect and drop bicycles by using a credit card, a personal smart card, or a mobile application. With a mobile application, the users can also locate available bicycles or locks at their preferred station.

In recent years, we see other concepts and variations of the BSS begin to flourish. An exciting trend is bicycles with technology and software integrated directly, so-called smart bicycles. These bicycles can communicate through a GPS, wireless connection, and electrical locks mounted directly in the bicycle (Antoniades and Chrysanthou, 2009). This enables new concepts like geo-fenced stations and free-floating systems.

In a *geo-fenced system*, the stations are defined as geographical areas rather than physical racks. The users are allowed to park their bicycle anywhere within this area. Accordingly, the congestion issue is reduced as it no longer is a problem if there are no locks available. These geo-fences are at fixed locations or moved as needed. An extended version of this is the free-floating BSS where the users can pick up or leave the bicycle wherever they like within a large area. These systems can act as standalone concepts or in combination with the traditional station-based BSS. Geo-fenced or free floating BSS introduce new possibilities for the rebalancing problem as the number of locks no longer is a restriction when it comes to unloading or parking bicycles. However, the situation where there are no available bicycles at a station is still a problem, and the company running the BSS is still faced with the challenge of rebalancing the system. Hamilton in Ontario and VeloGo in Ottawa are examples of BSSs that have adopted the geo-fence system (Social Bicycles).

Another concept enabled through technology is the electric bicycle. Electric bicycles are especially relevant to cities with steep topography, or of considerable size, as the users can ride longer distances. The bicycles are usually charged at the docking station. Tokyo is an example of a city with electric bicycles.

Another case, addressed by Fricker and Gast, is incentivizing the users to contribute towards solving the rebalancing problem. In exchange for monetary advantages, users can return their bicycle to the least loaded station among neighboring stations. This mechanism can equalize the one-directional flows of travels, for example from residential areas to work areas in morning rush hours, or the travel flow in steep regions (Fricker and Gast).

Lastly, we see that the trend is to have a system that responds and adapts to the individual's behavior using machine learning and other modern technologies. Johan Høgåsen-Hallesby, the CTO of UIP, believes that "combining that focus with the development towards station-less bike sharing, reveals new opportunities and changes the role bike sharing can play for public transport and urban mobility" (Høgåsen-Hallesby, 2017).

2.4 Bike Sharing in Norway

As one of the fastest growing cities in Europe, and with one of the most tech-savvy population in the world, Oslo has become a hub for urban innovation. The Digital Economy and Society index (DESI) of 2017 shows that the Norwegian population is more willing to adapt to digitalization than average and that the people put a great trust in mobile payment (European Commission, 2017). Norwegians also tend to like activities and solutions that are healthier for themselves, their city and their environments. These factors are what make Norway, and especially Oslo, an ideal place for developing shared, intelligent infrastructure (European Commission, 2017).

UIP is a Norwegian company that finances, delivers, and manages shared urban infrastructure, including BSSs (Urban Infrastructure Partners, 2017a). UIP is currently running three BSSs in Norway with stations in Oslo and Trondheim, and with ambitions of expanding to Bergen and other cities. The bicycles are illustrated in Figure 2.1. As illustrated in Figure 2.2 the users can get a real-time overview of available bicycles and locks through the mobile application *Bysykkel*. Figure 2.3 shows the map of all active stations in Oslo. In Oslo in 2017, UIP's 50,000 subscribers completed over 2.6 million bicycle trips. As of June 2018, over 700,000 trips are made in Oslo, and the bicycles are more popular than ever (Oslo Bysykkel, 2017).

While many BSSs focus on the number of bicycles they have in operation, UIP focuses on utilizing their resources by maximizing the number of trips each bicycle in the system has on average in a day. This focus on mobility, rather than just the novelty factor, is making them one of the most efficient BSS in the world (Urban Infrastructure Partners, 2017b). UIP's BSS is currently financed by customer subscription, advertisements on the racks, and sponsorship (Oslo Bysykkel, 2017). Service vehicles are used to rebalance the system, but the rebalancing strategies are solely based on the experience and gut-feeling of the drivers (Oslo Bysykkel, 2017). UIP is experimenting with different ways to encounter the imbalance issue. In 2017, they initiated a test project called *morning birds*. By analyzing historical data, they identified which stations that are first congested in the morning rush hours. The solution was to place UIP employees, *the morning birds*, at these stations for a few hours in the morning. Their task was to remove bicycles, and stack them up next to the racks as the station was filling up (Urban Infrastructure Partners, 2017b).

Figure 2.1: Picture of bicycles parked at a station in Oslo. Source: (Oslo Bysykkel, 2017)

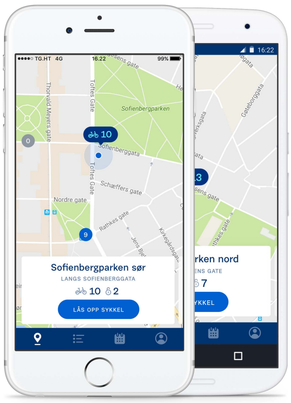


Figure 2.2: The *Bysykkel* mobile application. Source: (Oslo Bysykkel, 2017)

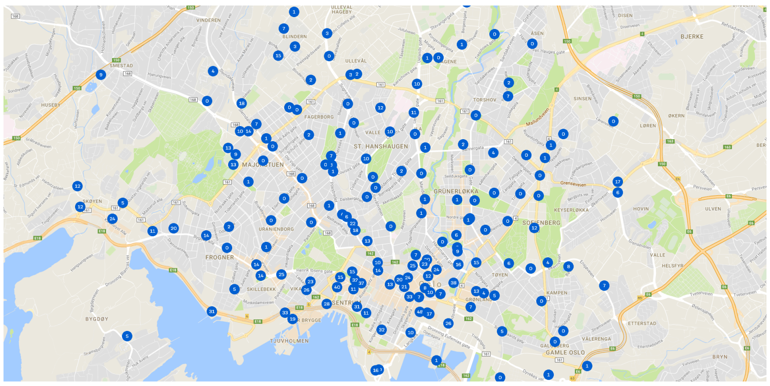


Figure 2.3: Spatial distribution of bicycle stations in Oslo. Source: (Urban Infrastructure Partners, 2017a)

2.5 Bike Sharing in Other Countries

As mentioned in Section 2.2, the simple concept of BSS has swept across the globe in a matter of few years (Goodyear, 2015). As of June 2018, 1,608 cities worldwide have a BSS (Meddin, 2018). In the next paragraphs, some of the world's most interesting BSSs are introduced.

The Chinese system, Hangzhou Public Bicycle, has, since its launch in 2008, expanded to 3,572 stations and 84,100 bicycles as of May 2016 (ESCI, 2016). The system is the second biggest by size, but probably the most extensively used. As long as the bicycle is returned within an hour, the user is not charged. One of the reasons this system has become so successful is because of its integration with public transit; a single card grants access to subways, bus, ferry, taxi, and bicycles. By 2020, it is projected to have 175,000 bicycles (ESCI, 2016). The system is owned and maintained by the government with intentions of decreasing traffic.

Mexico City launched a BSS, called EcoBici, in February 2010. Since the launch in 2010, the system has grown by 400 %, and, as of 2016, the system consists of 452 stations and more than 6,500 bicycles (EcoBici, 2016). Many refer to this as one of the most impactful BSSs, as it has made a dramatic change to the chronic congestion problem in the city with its nine million inhabitants. EcoBici's long-term goal is to convert five percentage of city journeys to cycling (Popova, 2016). The users are required to purchase a card that gives them access to the system for one year to use the bicycles.

In 2015, Hamilton, Ontario, introduced the first large-scale, city-wide smart bicycle system in the world. Their service area is defined by a large geo-fence, holding 100 smaller geo-fenced stations. The users can decide whether to park the bicycle in a geo-fence station, or, for a small fee, anywhere within the larger geo-fence. Likewise, if a user picks up a bicycle from outside a geo-fenced station, they receive extra credit on their account if they return the bicycle to a geo-fenced station. More perks include the ability to reserve a bicycle (Hughes, 2017).

2.6 Challenges Encountered

Companies running BSSs have to make a lot of decisions when it comes to station density, bicycles per capita, the system area, etc. Many companies have encountered difficulties in determining how big of a system to develop. To limit the risk of failure, a lot of cities have budgeted for undersized systems. However, this has been shown only to increase the chance of failure (Hughes, 2017).

The biggest problem bike sharing companies face, is to ensure bicycle and lock availability. Avoiding starving and congested stations have been a major challenge. Transportation researcher, Colin K. Hughes, conveys “Sometimes, I’d ride to my office to find no available locks, forcing me to ride halfway back home to leave the bicycle at nearest open lock, and then walk many blocks.” Hughes also states that “Reliability is a key factor in any user’s modal choice,” and emphasizes the importance of a balanced bicycle system. Users of the system also communicate that “We want the system to be reliable, not only for weekend trips but as a standard commuting option anywhere in a city at peak hours when it’s needed the most” (Hughes, 2017). Figure 2.4 illustrates the imbalance in the system in the morning rush hours. Blue circles indicate empty stations, whereas red circles indicate full stations.

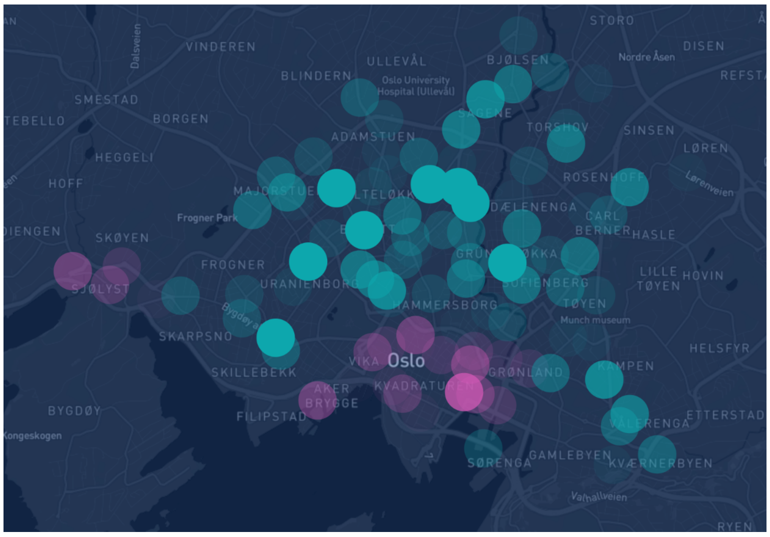


Figure 2.4: Illustration of imbalance during morning rush hour. The blue circles indicate empty stations, and the red circles indicate full stations. Source: (Oslo Bysykkel, 2017)

In November 2017, we visited UIP in Oslo to observe the daily operation. As mentioned earlier in this chapter, they do not utilize any mathematical program to determine routes for the service vehicles. What they expressed as the biggest challenge is that the inner city center experiences massive demand for locks in the morning hours as people ride their bicycle to work. They conveyed that a lot of the bicycles end up at the same stations every day, resulting in people leaving their bicycle unlocked alongside. Hence, their primary focus is on collecting the overflow of bicycles to avoid theft, as each bicycle is worth approximately 7,000 NOK.

A challenge regarding the balancing is predicting the demand for bicycles and locks at each station at a particular time. The bike sharing companies have no way of knowing the real historical demand since lost demand is not recorded. Demand prediction is also problematic, as the bicycle usage pattern varies with several factors as time of day, weather, social events, and traffic, and users usually choose a station near their origins or destinations on an ad-hoc basis (Chen et al., 2016).

Chapter 3

Literature Survey

There is an emerging interest in BSSs worldwide, especially within the field of operational research. The decisions related to BSSs are divided into multiple levels: a strategic level, a tactical level, and an operational level. Relevant decisions at the strategic level include location and sizing of the stations. At the tactical level, decisions regarding optimal inventory level and detection of broken bicycles are made, and at the operational level, rebalancing strategies are decided. Our main focus in this literature survey is the operational level of the BSS, and how the bicycles are rebalanced by service vehicles to satisfy the customers. The rebalancing task is two-folded and consists of finding the optimal geographical routes for the service vehicles, as well as determining the loading and unloading quantities at each station visit in the routes.

First, a previously conducted literature survey regarding exact solution methods is summarized in Section 3.1. In Section 3.2, a literature survey regarding heuristic solution methods applied to the BSS is presented. In Section 3.3, column generation heuristics applied to the vehicle routing problem are elaborated. The motive of this literature survey is to show the broad specter of solution approaches, and further pinpoint our motivation and purpose for this thesis. The main source for literature is www.sciencedirect.com. Keywords like *bike sharing systems heuristic* and *column generation heuristic VRP* are used and gave a total of 276 and 247 search results, respectively. Relevant, distinctive articles are selected and reduced to seven papers on heuristic methods and six papers on heuristic column generation. In Section 3.4, a summary of the literature survey and our motivation for this thesis is presented.

3.1 Exact Solution Methods

In general, the bicycle rebalancing problem is either static or dynamic, as illustrated in Table 3.1. The static rebalancing neglects customer interactions. The system is often static if rebalancing by the service vehicles is performed at night when closed for customer usage, or if the demand is considered so low that it can be neglected. In contrary, the dynamic problem aims to manage rebalancing operations during the day, hence the user demand is not negligible, and the system is changing over time. The customer demand is either deterministic or stochastic, i.e. known beforehand or random with a known probability distribution, respectively.

Table 3.1: Taxonomy

		Information quality	
		Deterministic input	**Stochastic input**
System evolution	**System is constant**	Static and deterministic	n/a
	System changes over time	Dynamic and deterministic	Dynamic and stochastic

A comprehensive study on the static rebalancing problem is found in Espegren and Kristianslund (2015). The static solutions are useful if the demand pattern is stable and predictable. However, if the demand varies considerably with time, the stations get imbalanced during the day and static rebalancing is insufficient (Ghosh et al., 2016). Hence, dynamic bicycle rebalancing is required. In the dynamic bicycle rebalancing problem, user demand is changing the system constantly and the complexity is increased.

An extensive literature survey on dynamic rebalancing studies is found in our project thesis (Gleditsch and Hagen, 2017). One of the main findings from this literature survey is that most of the models previously developed are shortsighted when rebalancing strategies are decided. When solving problems with a rolling horizon, it is common to divide the problem into multiple subproblems with a shorter time horizon. We observed that most of the models in previous studies prioritize decisions leading to instant good results at the expense of long-term performance. For this reason, the main contribution in our project thesis was a model that emphasizes long-term performance. Additionally, many of the studies utilize clustering of stations to reduce the size of the problem.

Solving a bicycle rebalancing problem is challenging as inventory management of bicycles at the stations and at the service vehicles also must be determined in addition to the geographical routing strategies. As concluded in many of the studies, it is inefficient to use exact solution methods to solve large, realistic rebalancing problems in general as the solution space becomes too large, and thus unsolvable for commercial software.

3.2 Heuristic Solution Methods

Heuristics are commonly used to obtain a good solution in reasonable time when the optimization problem at hand is intractable. Several heuristics may be suitable for the dynamic rebalancing problem. A list of the heuristic solution methods applied to the dynamic bicycle rebalancing problem found in the literature is as follows.

- Artificial bee colony algorithm with a rolling horizon framework
- Large neighbourhood search
- Destroy and repair algorithm
- Chemical reaction optimization
- GRASP with path re-linking
- Iterated Tabu search
- Variable neighbourhood search

The heuristic literature survey examines both static and dynamic rebalancing problems, and focuses on solution procedure, generation of initial solution(s), determination of loading quantities and evaluation of performance.

Shui and Szeto (2017) propose an enhanced artificial bee colony (EABC) algorithm that solves the static problem in a rolling horizon framework. The algorithm is inspired by the intelligent behavior of honeybees' foraging process. Initial food sources, i.e. routes, are randomly generated. The algorithm involves three types of bees with different roles in the exploration and exploitation of food sources. In each iteration of the algorithm, each food source is assigned to a bee, while the bee searches in the neighbourhood for better options. The best food source is determined, and a greedy loading heuristic is applied. The service vehicles are assumed to load or unload as many bicycles as possible. Additionally, a route truncation heuristic is executed to handle infeasible routes exceeding the time horizon. The EABC algorithm works well for instances with 180 stations, and they conclude that shorter stage lengths yield better solutions.

Ho and Szeto (2017) propose a hybrid large neighbourhood search (LNS) with multiple service vehicles. First, initial routes are created using a construction heuristic. This heuristic alternates between assigning pickup and delivery stations to a route until a time constraint is violated. Then, the hybrid large neighbourhood search improves the initial routes by destroying part of the routes by a removal operator to further repair them with a insertion operator. They present five removal and five insertion operators, which rely on simple mechanisms, to diversify and intensify the search. A tabu search is further applied to the most promising routes. Further, the loading quantities are determined with respect to spare capacity, remaining time, initial inventory and optimal state. All the elements of the algorithm are highly reliant on station characteristics. The algorithm is tested on instances up to 518 stations and five service vehicles. The computational results confirm that their heuristic performs better when tabu search is incorporated, and that each insertion and removal operator contributes to solution accuracy.

Dell et al. (2016) destroy and repair the solutions iteratively, while they also improve the solutions by local search until a stopping criteria is reached. The initial solutions are generated from the well-known savings algorithm by Clarke and Wright (Clarke and Wright, 1964), where first n routes are created consisting of one station each. Further, two routes are selected in each iteration and merged into a single route. This procedure is repeated until no more merging is possible. They define load windows as the feasible interval for load on the service vehicle before and after a station visit. Additionally, they introduce the concept of loss of flexibility in a merged route. This is the difference between the amount of feasibility in the merged route, and in the two original routes. A negative loss of flexibility reduces the load window for the merged route. Local search procedures are implemented to improve solutions, and include moving a station inter or intra-route, swapping two stations, swapping pairs of stations, or merging sequences of routes. The algorithm is able to solve all the small-size instances to optimality within an average time of three seconds.

Szeto et al. (2016) present a chemical reaction optimization (CRO) method to solve the static problem, and a subroutine method to determine the loading quantities. The major components of CRO are molecules and elementary reactions. Molecules consist of molecular structure, i.e. a solution, potential energy, i.e. objective value, and kinetic energy, i.e. the ability to escape local optimum. Two sets are introduced and used in the initialization to reduce the solution space, and contain pickup stations and delivery stations, respectively. The station randomly picked from the best three in the first set is chosen as the first station visit. Then a random station from set two is chosen as the second station visit. This is repeated until the time horizon is reached. Further, the CRO is applied and the elementary reactions include modifying, splitting and merging different molecules. Local search is implemented to improve the solution. Station visits are added if the time horizon is not fully utilized, infeasible solutions are repaired, and visiting order is re-arranged. The CRO algorithm is able to quickly obtain a good solution with instances up to 300 stations.

Ho and Szeto (2016) propose a method where GRASP is used to produce diverse solutions. There are two phases in each iteration. In the first phase, a new solution is constructed based on greediness and randomness, and the second phase improves the solution by local search. The disadvantage with GRASP is that the iterations are independent of each other. Hence, Ho and Szeto (2016) propose the concept of path-relinking, which generates new solutions in iteration i by combining elements from elite solutions found by GRASP in iteration $i-1$.

Ho and Szeto (2014) use tabu search, which is known for being efficient when solving routing problems. However, rebalancing requires loading quantities in addition to geographical routing decisions, hence new tabu search operators are added to handle the extra variables. The initial solution is generated by sorting stations in groups of pickup and delivery stations, and alternating between assigning pickup and delivery stations to a route until a time constraint is violated. To improve solution quality, the tabu search is embedded into an iterative framework where intensification and diversification mechanisms are applied.

Rainer-Harbach et al. (2013) address the rebalancing problem by using a variable neighbourhood search (VNS), and an embedded variable neighbourhood descent (VND) that exploits various neighbourhood structures. An initial solution for each vehicle is built by choosing among imbalanced stations that is reached without exceeding the time horizon. For diversification, neighbourhood operators, like move, swap, and remove, are applied randomly to solutions from the VNS neighbourhood. The VND neighbourhood engages problem specific operators, e.g. removal of unnecessary visits, insertion of imbalanced stations etc. Corresponding loading quantities are derived either by a greedy approach, a maximum flow network, or by a linear program. The greedy method is shown to be fastest, and delivers solutions of good quality. The LP method has the advantage of finding optimal quantities, but Rainer-Harbach et al. (2013) argue that the added flexibility cannot compensate for larger computational time.

To summarize this section, different heuristics may be suitable for the bicycle rebalancing problem. However, many of these methods do not consider the same problem properties and station characteristics as we do in our thesis. Most consider the static problem with one service vehicle, while we aim to solve the dynamic multi-vehicle problem. A heuristic not implemented on the bicycle rebalancing problem in literature is column generation heuristic. Column generation heuristics are shown to perform well on vehicle routing problems (VRPs), and we want to examine how it performs when applied to the BSSs. Our hypothesis is that a less random and a more controlled problem-specific approach may give good results. In the next section, a literature survey on heuristic column generations applied to VRPs are presented.

3.3 Heuristic Column Generation

The column generation (CG) algorithm decomposes the problem into a master problem and a pricing problem. Initially, a set of columns, i.e. possible solutions, are passed to the master problem. The method iterates between solving a reduced master problem with only a subset of all possible columns, and a pricing problem that finds new columns that can improve the objective function in the master problem. For CG to be considered an exact solution algorithm, both the master problem and the pricing problem must be solved to optimality in order to ensure that there are no non-included columns that can improve the best-found solution. However, the development of an exact solution procedure for the pricing problem is challenging for complex problems. Hence, it is appropriate to incorporate heuristic algorithms into the CG framework. This literature survey examines how a selection of papers regarding heuristic CG approaches the master problem, the pricing problem, and the initialization of columns. As literature regarding heuristic CG applied to BSSs was not discovered, literature regarding CG applied to VRPs are surveyed.

Pinto et al. (2018) solve the capacitated VRP, and propose four different heuristic approaches for creating initial columns. The basic idea is to make the solution obtained from LP relaxation of the reduces master problem feasisbile by fixing some variables. The first approach forces all variables from the LP solution to take an integer value. The second approach begins similarly as the first approach, but in addition, the column corresponding to the route with the highest usage is added to the final solution and the size of the master problem is reduced. The third approach suggests adding the customer to a route that is assigned to most routes first. The fourth approach is similar to the third approach, but suggests adding the customer that is assigned to less routes first. The master problem is solved with an exact solution algorithm as a set partitioning problem. The pricing problem is solved with variable neighbourhood search techniques.

Victoria et al. (2016) solve the capacitated VRP with time-dependent demand. The CG algorithm starts by solving the pricing problem in each iteration. The pricing problem constructs promising columns, and is solved as a longest-path problem using dynamic programming with heuristic dominance rules to speed up the algorithm. When no improving columns are found by the heuristic dominance rules, a full dominance check is performed. The problem is reformulated using Dantzig-Wolfe decomposition to obtain a master problem which is solved to its exact value. The pricing problem was tested with and without heuristic dominance rules. CG without heuristic dominance rules found three out of 15 optimal solutions, while CG with heuristic dominance rules found nine out of 15 optimal solutions.

Mahvash et al. (2015) propose two different approaches to the pricing problem with the aim of speeding up the CG algorithm. In the first approach, the pricing problem is solved as an elementary shortest path problem to find the routes that have a negative reduced cost. A heuristic is enforced on the identified routes to verify feasibility in terms of loading constraints. The second approach is developed to speed up the algorithm further. The approach involves a heuristic that attains feasible routes in terms of loading constraints with negative, but not necessarily minimal, reduced cost.

Guedes and Borenstein (2015) propose a state space reduction procedure combined with CG for solving large instances on the multi-depot vehicle scheduling problem. With many variables, most of them have small chances of being considered as good candidates for the final solution and are, thus, extracted through a state space reduction. The goal of this procedure is to identify relevant variables that are selected by any solution in the single-depot problem. If a variable is not chosen in the single-depot problem, then it would have small chance of beinge chosen as a candidate solution when considering the multi-depot problem. The reduced state space is used to generate initial columns. Guedes and Borenstein (2015) emphasize the importance of good initialization as this improves the convergence process. The master problem is solved right after a new column is created to accelerate convergence and to avoid generation of duplicate columns. Good initialization and state space reduction significantly increased the convergence process of the model.

Beheshti and Hejazi (2015) propose a hybrid method of Dantzig-Wolfe decomposed CG and an electromagnetism algorithm (EMA) that takes advantage of the attraction-repulsion mechanism of electromagnetic theory. The algorithm starts with a random population of particles, i.e. solutions, with individual charges, i.e. objective values. A better solution with lower charge attracts other particles, while bad solutions repel them. The procedure is split into an integration phase where the pricing problem of the CG is solved, and a collaboration phase where the CG and EMA are parallelized and information is exchanged to find better solutions. Initial columns are randomly generated, but Beheshti and Hejazi (2015) conclude that heuristic approaches should be used for initialization.

Venkateshan and Mathur (2011) develop a specialized CG routine that reduces the combinatorial explosion in the number of routes generated and the number of nodes explored. Venkateshan and Mathur (2011) prevent regeneration of sub-optimal routes with identical route vectors by having restricted routes. Additionally, techniques that helped the BB-three fathom quicker and a hierarchy of branching were presented. The minimization problem is solved with branch-and-bound, where the lower bounds are obtained with CG. The initial solutions are so-called infeasible or feasible dummy solutions. Each of these solutions satisfies the full demand of one node, but at the expense of a high cost.

3.4 Conclusion and Motivation of the Thesis

The most important characteristics of column generation heuristics surveyed in this section are summarized in Table 3.2. First, it is evident that BSSs have received increased attention from the academic community the last decade. We believe this is due to the attractiveness of these systems for the users and the society, and the significant operational complexities for the operators.

As discussed, it is inefficient to use exact solution methods to solve large, realistic rebalancing problems, and heuristic methods should be implemented. Column generation heuristics applied to BSSs are not discovered in literature, despite its success on VRPs. Our hypothesis is that a method with a low degree of randomness, and a higher degree of controlled problem-specific approach may give good results. As reviewed in Section 3.3, complex VRPs are solved quickly with column generation heuristic, and should be tested on a BSSs. As observed in literature, it is common to solve the master problem with an exact solution algorithm, while the initialization or/and pricing problem are solved heuristically. Some studies generate initial columns randomly. However, as Beheshti and Hejazi (2015) conclude, the initialization of columns should be based on a smart algorithm that creates sensible routes, as smart initialization makes the algorithm converge faster. In addition, our hypothesis is that good initialization diminishes the importance of an advanced pricing problem heuristic since good solutions are more likely to exist among the initial columns.

A general challenge with the bicycle rebalancing problem, compared to traditional VRP, is that, in addition to the routes, determination of loading and unloading quantities are required at each station visit. This complicates the problem significantly. Many of the models surveyed solve this by implementing a separate loading instruction heuristic with a greedy approach. Although an exact solution algorithm has the advantage of finding optimal loading quantities, it cannot compensate for the considerably larger computational time. Our hypothesis is that by having in-depth knowledge about the problem, we are able to generate a loading quantity heuristic that determines loading quantities that are as good, or even better in the long run, as the loading quantities determined with an exact algorithm. Lastly, as briefly mentioned in the literature survey, clustering of stations into zones are often used to reduce the solution space.

Table 3.2: Comparison of Column generation-based heuristic studies

Author	Problem	Initialization	Master problem	Pricing problem	Multiple visits to same station
Pinto et al. (2018)	Capacitated VRP	LP Relaxation of set partitioning problem with rounding heuristics.	Set partitioning problem	Elementary Shortest Path Problem solved with VNS	No
Victoria et al. (2016)	Capacitated VRP with time-dependant demand	n/a	Dantzig-Wolfe decomposition solved to optimality	ESPPRC solved with DP and heuristic dominance rules	No
Mahvash et al. (2015)	Capacitated VRP with 3D loading constraints	Assigning one customer to just one vehicle	Dantzig-Wolfe decomp. + set-partitioning problem	DP + Heuristic pricing to ensure feasibility	No
Guedes and Borenstein (2015)	Multi-depot vehicle scheduling problem	Paths obtained by the state space reduction heuristic	Dantzig-Wolfe decomp. + set-partitioning problem	Shortest path problem using the Small-label-first algorithm	No
Beheshti and Hejazi (2014)	VRP with soft time windows	Randomly generated	Dantzig-Wolfe decomp. solved to optimally.	ESPPRC solved with a quantum-inspired evolutionary algorithm + an Electromagnetism algorithm	Yes
Venkateshan and Mathur (2011)	Fleet size and mix VRP	Infeasible dummy solutions that satisfy all demand	B&B with CG	Time-horizon-constrained SPP solved with negative-cost cycle in a network with time windows	Yes

Chapter 4

Problem Description

The problem we aim to solve in this thesis is the dynamic stochastic bicycle rebalancing problem. This problem is described formally in Section 4.1. As the dynamic and stochastic routing problem is incredibly complex, we approach this problem by solving a series of smaller deterministic subproblems. This way the deterministic version of the problem is solved iteratively as new relevant information gets revealed. This subproblem is presented in Section 4.2. Central assumptions, constraints, and decisions are specified.

4.1 Dynamic Stochastic Bicycle Rebalancing Problem

In this Section, the real-world dynamic stochastic problem is elaborated together with an example problem. The real-world problem is dynamic as information concerning changes in the distribution of bicycles and service vehicles is not known beforehand, but becomes available during operation. Furthermore, the problem is stochastic as new arrivals of customers requesting locks or bicycles are assumed to be random variables with known probability distributions.

As the future distribution of the bicycles are partly dependent on a stochastic demand variable and partly by a controlled rebalancing decision, the problem is formulated as a *Markov Decision Process* (MDP). A MDP is a discrete-time stochastic control process and a framework for modeling decision making in situations where outcomes are both random and controlled by a decision maker (Mes and Rivera, 2017). A MDP, based on Bellman (1957), is formulated as shown in Formula (4.1).

$$s_t \xrightarrow{x_t} s_t^x \xrightarrow{\omega_t} s_{t+1} \tag{4.1}$$

An MDP consists of a set of possible states S for the environment, together with a set of decisions X. A probability distribution for the outcome of the stochastic variable is also known. This probability distribution may depend on the current state and the decision that is made. This problem is solved over a given set of decision points t in the horizon T. The transition process starts in state $s_t \in S_t$. Based on the information that is known in this state, a decision $x_t \in X_{S_t}$ is made. This decision leads to a post-decision state s_t^x. The exogenous information is now revealed, represented by the stochastic variable $\omega_t \in \Omega_t$. This information is memoryless, meaning it is only dependent on the present state, not on any previous information, and thus, satisfies the Markov Property. The process is now transitioned into the next state s_{t+1}, which gives the decision maker a corresponding reward or penalty depending on the expected future value of being in state s_{t+1}.

In the bike sharing problem, it is natural to have a decision point when a service vehicle arrives at a station. The state s_t contains information about the distributions of service vehicles and bicycles, while the decision x_t, determines the service vehicles' rebalancing strategy. The deterministic post-decision state s_t^x contains information about how the system is after the decision x_t is made. The transition to the next decision state s_{t+1} happens when a service vehicle arrives at a station, and the exogenous information ω_t is realized. ω_t contains information about accumulated customer demand since the last station visit s_t. The penalty considers the number of unsatisfied customers, i.e. violations, between the decision points s_t and s_{t+1}. In addition to this, the penalty evaluates the new state and its expected future violations.

As customer demand is uncertain, violations cannot be minimized directly. Instead, the expected number of violations is minimized. The goal is hence to find an optimal policy $\pi \in \prod$, that maps a decision x_t to every state s_t over the horizon T, where $\prod$ is the set of potential policies. The overall goal is to map the decisions that minimize the expected number of violations.

4.1.1 Example Problem

Figure 4.1 illustrates an example of a transition from state s_t to the next state s_{t+1}. In this example, there are three stations and one service vehicle. The small circles indicate the capacity. A filled circle means that there is a bicycle in the slot, and an open circle indicates an available slot. The stations have six slots each, whereas the service vehicle has five slots. In state s_t, depicted on the left side of Figure 4.1, station 1 contains three bicycles, station 2 contains zero bicycles, and station 3 contains four bicycles. The service vehicle is positioned at station 1 and is loaded with one bicycle. The time is 3:00 pm. The decision x_t is now decided based on the information presented in state s_t. This decision contains information about how many bicycles to pick up or deliver at the current station, and where to drive next. The resulting post-decision state s_t^x is shown in the center of Figure 4.1, where the decision x_t is to pick up two bicycles and drive to station 2.

While driving from station 1 to station 2 the stochastic demand variable ω_t is revealed. This variable contains information about the accumulated demand from time 3:00 to 3:15 pm. As shown on the right side of Figure 4.1, one bicycle has been picked up at station 3 at 3:05, and one bicycle has been delivered to station 1 at 3:11 pm. In addition to this, there has been one customer arriving at station 2 at 3:10 with intentions of getting a bicycle. As there were no available bicycles at this point, one violation has occurred. The next state s_{t+1} contains information about the system after customer demand has been revealed.

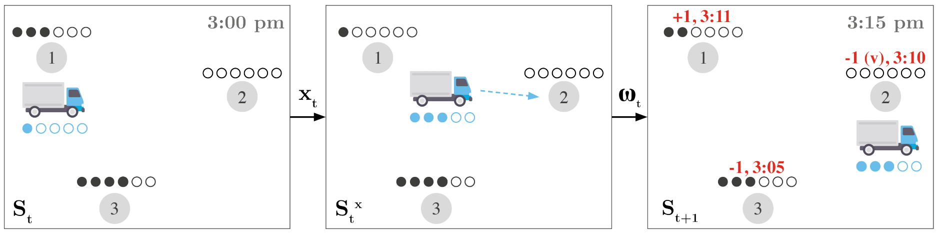


Figure 4.1: Example problem with initial state s_t, decision x_t, post-decision state s_t^x, exogenous information ω_t, and transition to next state s_{t+1}

4.1.2 Complexity of the Dynamic Stochastic Problem

Although formulating an MDP is relatively easy, the world is often so complicated that it is problematic or impossible to determine an optimal policy $\pi \in \prod$ as the number of potential policies are countless. Papadimitriou and Tsitsiklis (1987) analyze the computational complexity of MDPs and show that the problem is NP-complete and that it is no known algorithm that can solve general MDPs in a number of arithmetic operations in polynomial time. Both the state space S_t, the decision space X_t and the outcome space

Ω_t are victims of the curses of dimensionality. There exists a state for every possible distribution of bicycles, a decision for every possible rebalancing strategy, and a stochastic variable for every possible realization of customer demand. As of today, the BSS in Oslo has 1,790 bicycles and 158 stations. As the number of slots in the system is the double of the total number of bikes, each station has on average $\frac{1790 \cdot 2}{158} = 22$ slots. This gives a total of $\binom{1790+22-1}{22} = 2.3x10^{50}$ possible states. Additionally, the time horizon $(t, ..., T)$ can be very long, making the policy even harder to identify. Hence, we usually must settle for an approximation that is easier to solve.

A common technique for solving large-scale discrete-time multistage stochastic processes is to break the problem into smaller subproblems. The optimal policy for the MDP is then the one that provides an optimal solution to all the subproblems (Mes and Rivera, 2017). The generation of subproblems are done by shortening the time horizon from T to $\overline{T}$, where $\overline{T}$ is a suitable short horizon that is chosen to capture important behaviors (Powell, 2014). The daily time horizon for the rebalancing problem is 24 hours, but a time horizon shortened to 30 or 20 minutes might be enough to produce high-quality decisions.

Figure 4.2, shows how the dynamic stochastic problem, formulated as an MDP, is approximated through a series of smaller subproblems. The blue circles indicate decision points. Whereas a decision in the MDP consists of only one instruction about how many bicycles to load to a station and where to drive next, the subproblem provides a strategy for the entire shortened time horizon. Mark, however, even though a rebalancing strategy is made for the entire shortened time period $\overline{T}$ in the subproblem, the subproblem can be re-solved more frequently, and that this figure illustrates an example where the subproblem is re-solved with constant time intervals of $\overline{T}$.

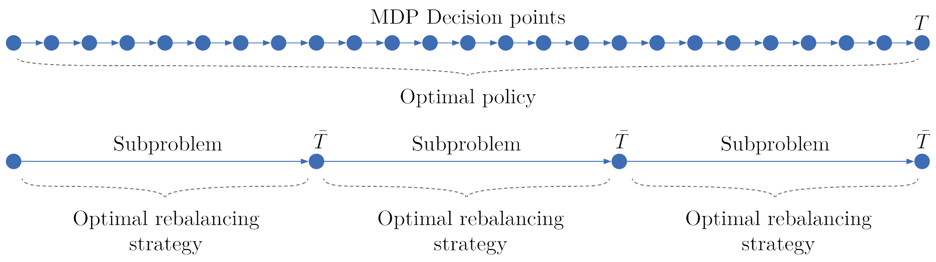


Figure 4.2: Illustration of how a Markov Decision Process is approximated by multiple subproblems.

When the time horizon is shortened, one can assume a known constant rate of customers within that short time period. Hence, the problem shifts from being stochastic to deterministic. Even though we now know the demand beforehand, we still consider it a dynamic problem since customer interactions change the station loads within the time horizon. This approach is closer to the real dynamic stochastic problem compared to a static approach where customer interactions are negligible.

An advantage of the dynamic problem formulated as an MDP is that each decision is made while considering the consequences for the entire horizon T, i.e. the total expected penalty is minimized. If the subproblem is modeled to find the solutions that are optimal for the shortened time period $\overline{T}$, consequences that occur after this period are ignored. In a real-world setting, it is not preferable to carry out rebalancing strategies that are optimal in the current period if they trigger unnecessary imbalance in the following period. While modeling a dynamic stochastic problem as a set of smaller subproblems, it is important to be aware of this potential weakness, and thus, try to incorporate elements in the subproblem that also consider future consequences.

4.2 Dynamic Deterministic Bicycle Rebalancing Subproblem

The dynamic problem is approximated by solving a series of smaller deterministic subproblems over a shorter time horizon. We call this subproblem the Dynamic Deterministic Bicycle Rebalancing Subproblem (DDBRS). In this section, the main assumptions, definitions, and restrictions are elaborated.

When a customer arrives at an empty station with intentions of getting a bicycle, the station is starving. Likewise, when a customer arrives at a full station with intentions of delivering a bicycle, the station is congested. Recall that both of these situations are referred to as a violation. We want to minimize these events.

The initial state describes the situation at the time the subproblem is solved and includes information about the number of bicycles positioned at each station, the current position of each service vehicle, and the number of bicycles at each service vehicle.

Future demand is uncertain; thus, an estimate is made based on historical data. In the subproblem, the stochastic information regarding demand at each station is assumed known and is assumed to follow a constant rate. The *net demand* is defined as the amount of customers that request a lock minus the amount of customers that request a bicycle every time unit. We define each station as either a *pickup station* or a *delivery station*, depending on whether it has a positive or negative net demand, respectively. We assume that it is always reasonable to pick up bicycles from pickup stations and deliver bicycles to delivery stations. The rates of deliveries and arrivals of bicycles depend on the time of the day and the station.

The driving times between the stations are also assumed known and are input parameters to the subproblem. There is a fixed parking time associated with a station visit. This time includes the time it takes to park the service vehicle and get started again. In addition to this, there is a unit handling time. This time is proportional to the number of bicycles that are handled, and is constant regardless of whether the bicycle is being loaded to or unloaded from the station.

Each service vehicle has a constant number of slots, and the number of bicycles that is loaded to or unloaded from a station is therefore restricted to the service vehicle's capacity. The fleet of service vehicles is homogeneous, i.e. each service vehicle has the same, constant number of slots.

The stations have a given number of locks. This number varies for each station. This means that one cannot bring more bicycles to a station than there are locks available. Nor can one pick up more bicycles from a station than there are bicycles currently parked there. Each station has a predefined optimal number of bicycles calculated for each hour. This is defined as the station's *optimal state*. The goal is to fulfill the optimal level of bicycles at each station at the end the time horizon. We want to minimize the difference between the station load at the end of the time horizon and its optimal state. This difference is defined as *deviation*.

To summarize, the objective is to find effective rebalancing strategies for the service vehicles. These decisions are made based on fixed and time-dependent input parameters. A conceptual overview of the model is shown in Figure 4.3.

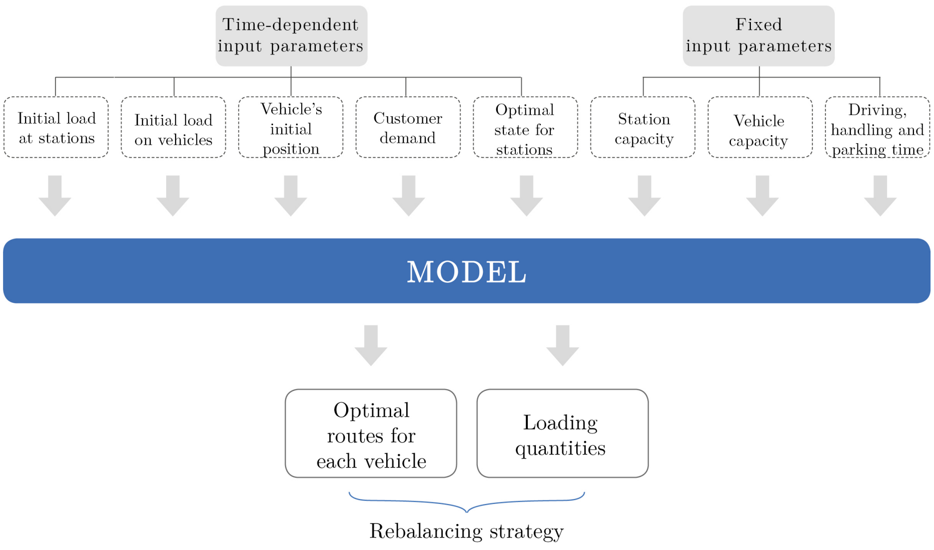


Figure 4.3: Conceptual overview of the input and output of the subproblem.

4.2.1 Example Problem

Figure 4.4 illustrates an example of the DDBRS. The system consists of three stations and one service vehicle. The time horizon $\overline{T}$ is 15 minutes. The initial state, depicted on the left side of the figure, indicates that the time is 3:00, the service vehicle is empty, and stations 1, 2 and 3 contain five, zero and two bicycles, respectively. The red numbers indicate the known demand within the time horizon. A negative net demand indicates more request for bicycles, and a positive net demand indicates more requests for locks.

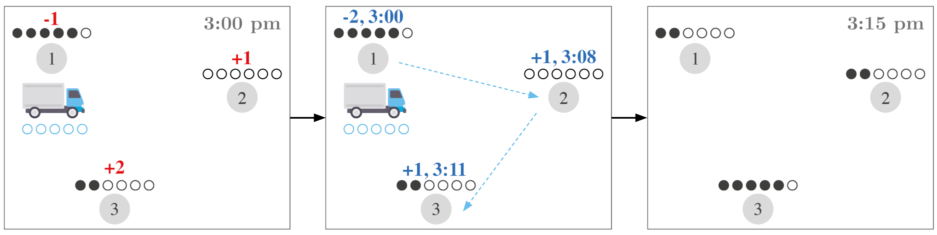


Figure 4.4: The Dynamic Deterministic Bicycle Rebalancing Subproblem. The three figures illustrate the initial state, the service vehicle route, and the final state, respectively.

In this example, one customer wants to rent a bicycle from station 1, one customer wants to return a bicycle to station 2, and two customers want to return their bicycles to station 3. Based on the current state and the known demand, a decision concerning the service vehicle route and load is planned. This plan, as illustrated by the blue symbols in the middle of the figure, consists of picking up two bicycles from station 1, driving to station 2 where one bicycle is delivered, and then drive to station 3 where another bicycle is delivered. The arrival times for the service vehicle are derived based on known transportation and handling time. The right side of the figure shows the resulting situation at 3:15.

Chapter 5

Mathematical Model

In this chapter, a mathematical model for the dynamic deterministic bicycle rebalancing subproblem (DDBRS) is presented. This model is equivalent to the model presented in our project thesis (Gleditsch and Hagen, 2017). As mentioned, a common approach to the dynamic stochastic real-world problem is to approximate it by solving a series of smaller deterministic subproblems. How our model intend to limit shortsightedness, is presented in Section 5.1. The main assumptions for this subproblem are presented in Section 5.2. The notation and constraints used to model the DDBRS are described in detail in Section 5.3 and 5.4, respectively. Lastly, the objective function is formulated in Section 5.5.

5.1 Limit Shortsigthedness

As mentioned in the literature study in Section 3.1, many of the models presented in the studies surveyed do not account for future planning periods, and we, therefore, perceive them as slightly shortsighted. To avoid shortsightedness, two innovative aspects are incorporated in our model. Firstly, we observed in previous studies that the service vehicles would stop before the time horizon if they did not have time to complete an additional trip. To avoid this, we allow the last trip for each service vehicle to exceed the time horizon. This is illustrated in Figure 5.1. By doing this, the idle time of the service vehicle at the end of the period is reduced, and a rebalancing strategy benefiting the future is initiated instead of a shorter disadvantageous trip. The second aspect that may contribute to a long-term focus is the use of optimal state. The optimal state indicates how many bicycles that should be loaded at a station at a particular time to minimize the chance of violations in

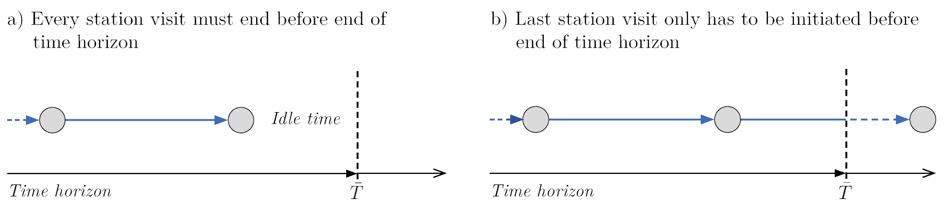

a) Every station visit must end before end of time horizon
Idle time
Time horizon
$\bar{T}$
b) Last station visit only has to be initiated before end of time horizon
Time horizon
$\bar{T}$

Figure 5.1: a) Illustration of how service vehicles may be idle at the end of each time horizon if every station visit must be completed before the end of the time horizon. b) Illustration of how the last station visit are allowed to happen after the time horizon.

the next planning period. We want to incorporate the optimal state by minimizing the difference between the optimal state and the station load at the end of the planning period, i.e. the deviation. By implementing these new aspects into the rebalancing problem, we hope that the rebalancing strategies avoid a myopic focus, and that this can contribute to better solutions in a real-world implementation.

5.2 Assumptions

The assumptions for the DDBRS aim to reduce the complexity of the model, without compromising the realism of the problem. Recall that the subproblem is solved with a shorter time horizon. Whereas a decision in the MDP consists of only one instruction about how many bicycles to load and where to drive next, the DDBRS provides a rebalancing strategy for the entire time horizon. Depending on the length of this time horizon, the service vehicles may be routed for multiple station visits.

As the time horizon is relatively short, the customer demand is assumed known and to follow a constant rate. This means that the net demand is either positive or negative, and indicates how many bicycles that on average either arrive or depart every time unit. This arrival rate does, however, vary with the time of day. Hence, each subproblem is solved with different time-dependent input parameters. The demand increases or decreases the inventory level until the station gets empty or full unless the station is rebalanced by a service vehicle.

We assume that the first station visit for each service vehicle is pre-determined. Also, the driving times from the service vehicles' current positions to their predetermined first station visit are given. This means that we do not know the service vehicles' exact geographical positions, rather, we know where they are headed and their arrival times. The loading quantity at the first station visit is not pre-determined. The service vehicles must finish their ongoing trips before being routed to a new station. To avoid shortsightedness, we allow the last trip for each service vehicle to exceed the time horizon.

The service vehicles can visit multiple stations within the time horizon. However, each station can only be visited a certain number of times. This assumption is included to reduce the dimensionality of the problem. Service vehicles are assumed to drive directly between stations. The parking time at a station is fixed and the handling time is proportional to the number of bicycles being loaded or unloaded. A station cannot be visited by more than one service vehicle at a time. The first service vehicle must finish handling the bicycles before a new service vehicle may arrive.

In this model, all violations are weighted equally. However, violations at different stations and times could be weighted differently, as well as different weighting of congestions and starvations. Additionally, the costs of utilizing the service vehicles, e.g. fuel and labour costs, are neglected. This is because we assume that the service vehicles are working constantly, and not only upon request. We assume that the focus is on rebalancing the system and meeting customer demand, rather than minimizing costs.

5.3 Notation

In this section the sets, indices, parameters, and variables used to model the DDBRS are described. A summary of all notation used is presented in Table 5.1.

The problem consists of a set of stations S, indexed by i and j, and a set of service vehicles V, indexed by v. The problem is solved with a time horizon $\overline{T}$. The service vehicles must finish their ongoing station visits before being routed to a new station. As illustrated in Figure 5.2, $o(v)$ denotes the first station visit for service vehicle v, and $T_v^{o(v)}$ the remaining driving time to this station. All service vehicles must end their routes at an artificial destination d. This destination is common for all the service vehicles, and is included for modeling purposes only. This station is not a real station, hence the driving time to the artificial destination from other stations is zero. The station capacity at d is also zero.

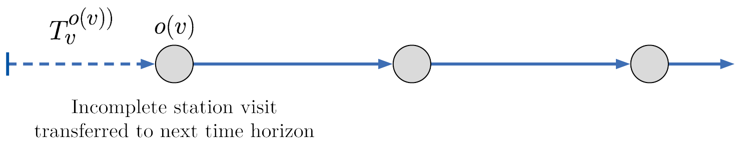


Figure 5.2: If a service vehicle has not completed a station visit, this station visit is transferred to the consecutive time horizon.

Every station has a set of possible visits M, indexed by m and n. A station visit is referred to as (i,m) where i indicates the station id, and m indicates the visit number for that particular station. The maximum number of times a station can be visited by a service vehicle is limited to $\overline{M}$. The first time a station is visited by a service vehicle is labeled by $m = 1$, and the second visit by $m = 2$ and so on. Each station i has a limited capacity, denoted by Q_i^S. Similar to a station, each service vehicle v has a limited number of bicycles that can be transported at the same time, denoted by Q_v^V.

$L_i^{S,o}$ and $L_v^{V,o}$ define the initial load of bicycles at each station i and on each service vehicle v, respectively. The optimal state at station i is denoted O_i. The deviation at a station i is denoted as d_i. This describes how far away the bicycle load at a station is from its optimal state O_i at the time horizon $\overline{T}$. The driving time between station i and station j is denoted by T_{ij}^D. T^P is a fixed time used for parking, and T^H is the unit handling time used for loading and unloading one bicycles.

The model uses the arc-flow variable, x_{imjnv}, which is a binary variable that indicates whether service vehicle v drives directly from station visit (i,m) to station visit (j,n). This formulation makes multiple visits to a station possible, and keeps track of the timing of events. The continuous time variable t_{im} represents the time for which station visit (i,m) starts. The variables q_{imv}^L and q_{imv}^U keep track of how many bicycles that are loaded and unloaded, respectively, to/from station i by service vehicle v. Note that the loading variables are indexed by visit number m, which ties them to a specific visit.

l_{im}^S and l_{imv}^V are the inventory variables. l_{im}^S denotes the number of bicycles at station i just before station visit (i,m). The variable is independent of which service vehicle that serves the station. l_{imv}^V denotes the bicycle load at service vehicle v just after the service vehicle has completed station visit (i,m). The customer demand D_i represents the net demand per time unit at station i. A positive demand indicates demand for locks, and a negative demand indicates demand for bicycles. Stations with positive demand are denoted as pickup stations, and stations with negative demand are denoted as delivery stations. The inventory level at a station and how it is affected by rebalancing operations and customer demand, is illustrated in Figure 5.3. The initial bicycle load at this station is $L_i^{S,o}$. The station faces demand D_i for bicycles, i.e. a negative demand. The demand decreases the inventory level to zero. At time t_{i1} a service vehicle arrives and delivers a loading quantity q_{ivm}^L to the station. Customers continue to demand bicycles, but a service vehicle arrives at the station before any violations occur again.

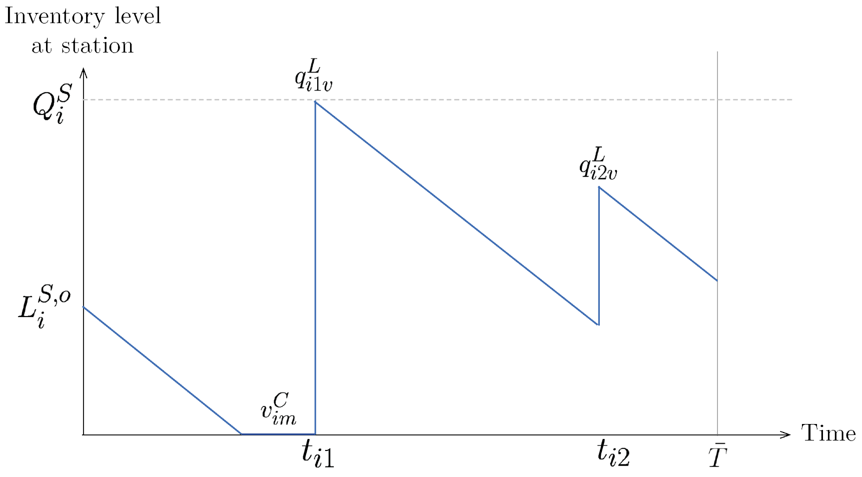


Figure 5.3: The changes in the inventory level at a station due to customer demand and rebalancing conducted by service vehicles.

To begin with, two variables for violations are introduced. v_{im}^C and v_{im}^S represent the accumulated congestions and starvations between station visit (i,m) and the station's previous visit $(i,m-1)$, respectively. In case of no previous trips, these variables represent the violations from the beginning of the time horizon until the station visit (i,m). In Subsection 5.4.2, additional variables and constraints that capture violations in different situations, are presented.

Table 5.1: Notation used for modelling the SBRP

Sets

S	Set of stations
V	Set of service vehicles
M	Set of visits at a station

Indices

i,j	Station $i,j \in S$
v	Service vehicle $v \in V$
m,n	Visit number $m \in M$
$o(v)$	Pre-determined station visit for service vehicle v
d	Artificial destination

Parameters

$\overline{T}$	Time horizon
Q_v^V	Storage capacity at service vehicle v
Q_i^S	Docking capacity at station i
$L_v^{V,o}$	Initial load of bicycles at service vehicle v
$L_i^{S,o}$	Initial load of bicycles at station i
O_i	Optimal state at station i at time $\overline{T}$
$\overline{M}$	Maximum number of visits at a station
T_{ij}^D	Driving time between station i and j
$T_v^{o(v)}$	Driving time for service vehicle v to the starting station $o(v)$
T^P	Service vehicle parking time
T^H	Unit handling time used for picking up or delivering bicycles
D_i	Net customer demand at station i

Variables

x_{imjnv}	1 if service vehicle v drives directly from station visit (i,m) to station visit (j,n), 0 otherwise
t_{im}	Time station visit (i,m) begins
q_{imv}^L	Number of bicycles loaded to station i at visit m by service vehicle v
q_{imv}^U	Number of bicycles unloaded from a station i at visit m by service vehicle v
l_{im}^S	Bicycle load at station i when visit m starts
l_{imv}^V	Bicycle load at service vehicle v right after station visit (i,m)
s_i	Inventory level at station i at time $\overline{T}$
s_v	Inventory level at vehicle v at time $\overline{T}$
v_{im}^C	Congestion at station i between station visit m and its previous visit
v_{im}^S	Starvation at station i between station visit m and its previous visit
d_i	Deviation at station i

5.4 Constraints

In this Section, the constraints that ensure feasible routes, i.e. the routing constraints, vehicle and station loading constraints, time constraints, and the non-negative, binary and integer constraints, are presented. Further, the constraints that ensure the right counting of violations and deviations are presented.

5.4.1 Ensure Feasible Routes

Routing Constraints

The routing constraints (5.1)-(5.5) force the service vehicles to have continuous routes. Each service vehicle has to start in $o(v)$ and end in d. In addition to this, each station visit can only be completed once.

$$\sum_{j\in S}\sum_{n\in M} x_{o(v)1jnv} = 1 \qquad v\in V \tag{5.1}$$

$$\sum_{i\in S}\sum_{m\in M} x_{imd1v} = 1 \qquad v\in V \tag{5.2}$$

$$\sum_{i\in S}\sum_{m\in M} x_{imjnv} - \sum_{i\in S}\sum_{m\in M} x_{jnimv} = 0 \qquad v\in V, j\in\{o(v),d\}, n\in M\backslash\{1\} \tag{5.3}$$

$$\sum_{i\in S}\sum_{m\in M} x_{imjnv} - \sum_{i\in S}\sum_{m\in M} x_{jnimv} = 0 \qquad v\in V, j\in S\backslash\{o(v),d\}, n\in M \tag{5.4}$$

$$\sum_{j\in S}\sum_{n\in M}\sum_{v\in V} x_{imjnv} \le 1 \qquad i\in S\backslash\{d\}, m\in M \tag{5.5}$$

Vehicle Loading Constraints

Service vehicle load constraints (5.6) make sure that the service vehicle load after visiting the first station $o(v)$ is updated. Constraints (5.7) ensure service vehicle load balance between two station visits. Constraints (5.8) and (5.9) make sure no more bicycles are loaded from the service vehicle than available bicycles, and no more bicycles are loaded to the service vehicle than available locks at the service vehicle, respectively. This ensures that the load at the service vehicle never exceeds the service vehicle's capacity.

$$l^V_{o(v)1v} - q^U_{o(v)1v} + q^L_{o(v)1v} = L^{V,o}_v \qquad v \in V \tag{5.6}$$

$$(l^V_{imv} + q^U_{jnv} - q^L_{jnv} - l^V_{jnv})x_{imjnv} = 0 \qquad v \in V, i, j \in S\backslash\{d\}, m, n \in M \tag{5.7}$$

$$q^U_{imv} \leq l^V_{imv} \leq \sum_{j \in S}\sum_{n \in M} Q^V_v \cdot x_{imjnv} \qquad v \in V, i \in S\backslash\{d\}, m \in M \tag{5.8}$$

$$l^V_{imv} \leq \sum_{j \in S}\sum_{n \in M} Q^V_v \cdot x_{imjnv} - q^L_{imv} \qquad v \in V, i \in S\backslash\{d\}, m \in M \tag{5.9}$$

Station Loading Constraints

Constraints (5.10) restrict service vehicles from loading more bicycles to a station than the station's capacity. Constraints (5.11) and (5.12) handle the loading balance and violations at each station. Constraints (5.11) capture violations between the beginning of the time horizon and the first station visit. Constraints (5.12) capture violations between any two station visits.

$$l^S_{im} + \sum_{v \in V}(q^L_{imv} - q^U_{imv}) \leq Q^S_i \qquad i \in S\backslash\{d\}, m \in M \tag{5.10}$$

$$l^S_{i1} - D_i t_{i1} - v^S_{i1} + v^C_{i1} = L^{S,o}_i \qquad i \in S\backslash\{d\} \tag{5.11}$$

$$\begin{aligned}(l^S_{i,m-1} + \sum_{v \in V}(q^L_{i,m-1,v} - q^U_{i,m-1,v}) + D_i(t_{im} - t_{i,m-1})\\ + v^S_{im} - v^C_{im} - l^S_{im})\sum_{j \in S}\sum_{n \in M}\sum_{v \in V} x_{imjnv} = 0 \qquad i \in S, m \in m\backslash\{1\}\end{aligned} \tag{5.12}$$

Time Constraints

The time constraints (5.13) force the time of arrival at station visit (i,m) to be before the time of arrival at station visit (j,n) if the service vehicle drives directly between these two station visits. It also includes the parking, handling and driving time it takes to complete the loading and driving. Constraints (5.14) say that two service vehicles cannot visit the same station at the same time. Constraints (5.15) restrict the time of arrival at the first station $o(v)$ to be after the time it takes to actually get there. Constraints (5.16) ensure that the time of arrival is within the time horizon for all station visits with exception of the last station visit. Constraints (5.17) force the visit time for station visit (i,m) to be zero if no visit is made. These constraints would normally be unnecessary, but as we allow one additional trip after the time horizon, it complicates how the violations are counted, and we need these constraints to capture which station visit that is the last for each station. This is further discussed in Subsection 5.4.2. Lastly, constraints (5.18) say that visit $(m-1)$ has to be done before visit m. These constraints work as symmetry breaking constraints, and help reduce the computational time.

$$(t_{im} + T^H \sum_{v \in V} (q^L_{imv} + q^U_{imv}) + T^P + T^D_{ij} - t_{jn}) \sum_{v \in V} x_{imjnv} \leq 0$$

$$i, j \in S \backslash \{d\}, m, n \in M \tag{5.13}$$

$$t_{im} - (t_{i(m-1)} + T^H \sum_{v \in V} (q^L_{i(m-1)v} + q^U_{i(m-1)v}) + T^P) \sum_{j \in S} \sum_{n \in M} \sum_{v \in V} x_{imjnv} \geq 0$$

$$i \in S, m \in M \backslash \{1\} \tag{5.14}$$

$$t_{o(v)1} \geq T^{o(v)}_v \qquad v \in V \tag{5.15}$$

$$(1 - \sum_{v \in V} x_{imd1v}) \cdot t_{im} \leq \overline{T} \qquad i \in S \backslash \{d\}, m \in M \tag{5.16}$$

$$t_{im}(1 - \sum_{j \in S} \sum_{n \in M} \sum_{v \in V} x_{imjnv}) \leq 0 \qquad i \in S \backslash \{d\}, m \in M \tag{5.17}$$

$$t_{im}(1 - \sum_{j \in S} \sum_{n \in M} \sum_{v \in V} x_{i(m-1)jnv}) \leq 0 \qquad i \in S, m \in M | \{1\} \tag{5.18}$$

Non-negativity, Integer and Binary Constraints

Binary, integer and non-negativity constraints are given by (5.19)-(5.28). The flow variables must be binary, while the loading and unloading variables must be integers. As the expected demand follows a continuous rate, the violation and station load variables are continuous. All variables are non-negative. Constraints (5.7), (5.12) - (5.14), and (5.16) - (5.18) are nonlinear and need to be linearized before implementation. The linearization can be seen in Section A.1 in Appendix A.

$$x_{imjnv} \in \{0,1\} \quad i,j \in S, m,n \in M, v \in V \tag{5.19}$$

$$q^L_{imv}, q^U_{imv} \geq 0, \text{integer} \quad i \in S, m \in M, v \in V \tag{5.20}$$

$$t_{im} \geq 0 \quad i \in S, m \in M \tag{5.21}$$

$$v^C_{im}, v^S_{im} \geq 0 \quad i \in S, m \in M \tag{5.22}$$

$$l^S_{im} \geq 0 \quad i \in S, m \in M \tag{5.23}$$

$$l^V_{imv} \geq 0 \quad i \in S, m \in M, v \in V \tag{5.24}$$

$$s_i \geq 0 \quad i \in S \tag{5.25}$$

$$s_v \geq 0 \quad v \in V \tag{5.26}$$

$$v^C_{im}, v^S_{im} \geq 0 \quad i \in S, m \in M \tag{5.27}$$

$$d_i \geq 0 \quad i \in S \tag{5.28}$$

5.4.2 Violations and Deviations

For each station, we want to capture violations only until the time horizon $\overline{T}$, as illustrated by the solid line in Figure 5.4. If we count violations until the very last station visit for each station, the period where we count violations would vary for each station as they all have different arrival times for their last visit. We want to incentivize the service vehicles to start a new trip even though they will not get there within the time horizon. As the objective includes minimizing violations, the service vehicles would not be motivated to initiate trips that exceed the time horizon if we were to count violations until the very last visit. Thus, we cannot count violations after the time horizon. Instead, we reward these initiated trips. This is discussed further in Section 5.5.

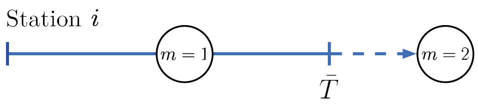


Figure 5.4: The violations are only counted until the time horizon, indicated by the solid line.

For each station we have three different situations: 1) a station does not get any visits, 2) a station gets its last visit within the time horizon, and 3) a station gets its last visit after the time horizon. To separate the violations in these different situations, some additional violation variables are presented. Figures 5.5, 5.6 and 5.7 illustrate these situations and the different violation variables, respectively. As mentioned in Section 5.3, v^C_{im} and v^S_{im} are the congestion and starvation between station visit (i,m) and its previous station visit $(i,m-1)$, respectively. If there are no previous station visits, these variables capture the violations from the beginning of the time horizon until station visit (i,m).

$v^{C,f}_i$	Congestion from last visit (i,m) to $\overline{T}$ if last visit is before $\overline{T}$, or total congestion if no visits
$v^{S,f}_i$	Starvation from last visit (i,m) to $\overline{T}$ if last visit is before $\overline{T}$, or total starvation if no visits
$v^{S,F}_i$	Starvation from $\overline{T}$ to last visit (i,m) if last visit is after $\overline{T}$
$v^{C,F}_i$	Congestion from $\overline{T}$ to last visit (i,m) if last visit is after $\overline{T}$

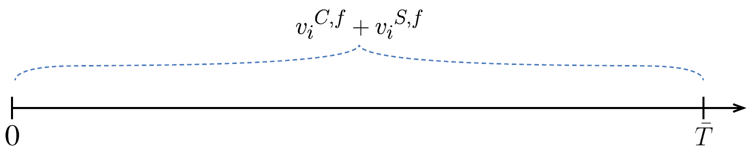


Figure 5.5: Situation 1: Station i does not get any visits

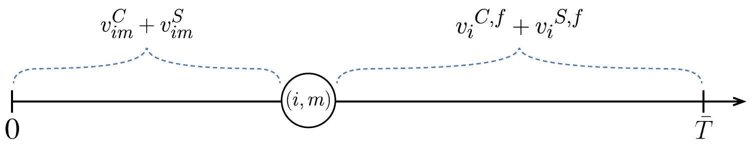


Figure 5.6: Situation 2: Station i gets its last visit within the time horizon

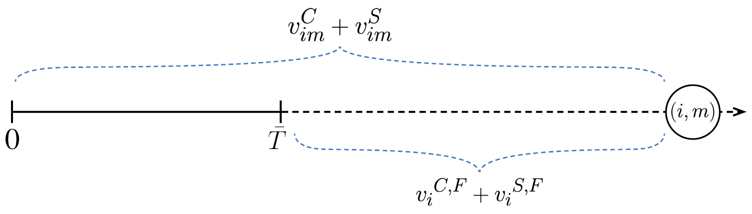


Figure 5.7: Situation 3: Station i gets its last visit after the time horizon

To avoid counting violations after the time horizon, the violations from the time horizon until the last visit $(v_i^{C,F} + v_i^{S,F})$ have to be subtracted from the total violations. The total violations are now calculated as expressed in (5.29).

$$\sum_{i \in S, m \in M} (v_{im}^C + v_{im}^S) + \sum_{i \in S} (v_i^{C,f} + v_i^{S,f}) - \sum_{i \in S} (v_i^{S,F} + v_i^{C,F}) \tag{5.29}$$

In the rest of this section, the constraints that capture these different violation variables are introduced. First, to distinguish between the different situations, the following binary variables are applied. These variables are determined by constraints (5.30)-(5.36).

$\delta_{im}^{\overline{T}}$	1 if station visit (i,m) is after the time horizon $\overline{T}$, 0 otherwise
θ_{im}^{f}	1 if station visit (i,m) is the last visit for station i, 0 otherwise
γi	1 if station i gets at least one visit, 0 otherwise

$$t_{im}(1-\delta_{im}^{\overline{T}}) \leq \overline{T} \qquad i \in S, m \in M \tag{5.30}$$

$$t_{im} \geq \overline{T} \cdot \delta_{im}^{\overline{T}} \qquad i \in S, m \in M \tag{5.31}$$

$$t_{im} \cdot (1-\theta_{im}) \leq t_{i(m+1)} \qquad i \in S, m \in M \setminus \{|M|\} \tag{5.32}$$

$$t_{im} \geq t_{i(m+1)} \theta_{im} \qquad i \in S, m \in M \setminus \{|M|\} \tag{5.33}$$

$$t_{i|M|} \geq \theta_{i|M|} \qquad i \in S \tag{5.34}$$

$$t_{i|M|}(1-\theta_{i|M|}) \leq 0 \qquad i \in S \tag{5.35}$$

$$\gamma_i = \sum_{j \in S} \sum_{n \in M} \sum_{v \in V} x_{i1jnv} \qquad i \in S \tag{5.36}$$

Situation 1: A station does not get any visits within the time horizon. The total violations are $v_i^{C,f} + v_i^{S,f}$, and are shown in Figure 5.5. The station load at $\overline{T}$ is s_i. Constraints (5.37) capture the violations when the station does not get any visits within the time horizon, i.e. $\gamma_i = 0$.

$$(-s_i + L_i^{S,o} + D_i\overline{T} - v_i^{C,f} + v_i^{S,f})(1-\gamma_i) = 0 \qquad i \in S \setminus \{d\} \tag{5.37}$$

Situation 2: A station gets its last visit within the time horizon. The total violations are $\sum_{m \in M}(v_{im}^C + v_{im}^S) + (v_i^{C,f} + v_i^{S,f})$, and are shown in Figure 5.6. Constraints (5.38) capture the violations from the last visit until the time horizon. Note that these restrictions are realized when the station visit (i,m) is both before the time horizon, i.e. $\delta_{im} = 0$, and the last visit for this station, i.e. $\theta_{im} = 1$.

$$(-s_I + l^S_{im} + \sum_{v \in V} (q^L_{imv} - q^U_{imv}) + D_i(\overline{T} - t_{im}) + v^{S,f}_i - v^{C,f}_i)\,(1 - \delta_{im}) \cdot \theta_{im} = 0$$
$$i \in S, m \in M \tag{5.38}$$

Situation 3: A station gets its last visit after the time horizon. Constraints (5.39) capture the violations from the time horizon $\overline{T}$ until the last visit. Note that these constraints are only realized when station visit (i,m) is both after the time horizon, i.e. $\delta_{im} = 1$, and is the last visit for station i, i.e. $\theta_{im} = 1$. Also, note that both of these binary variables are needed as a station can have multiple visits after the time horizon. The service vehicles, however, are restricted to only complete one station visit after the time horizon.

$$(-s_i + (v^S_{im} - v^{S,F}_i) - (v^C_{im} - v^{C,F}_i) + l^S_{im} - D_i(t_{im} - \overline{T}) + v^C_{im} - v^S_{im})\,\delta_{im} \cdot \theta_{im} = 0$$
$$i \in S, m \in M \tag{5.39}$$

As these violations are subtracted from the total violations, the model tries to maximize these violations. Thus, we need constraints that ensure that the violations can be greater than zero only if the station is actually full or empty at the time of the visit. Constraints (5.40) and (5.41) ensure this. $v_{im} - v^{C,F}_i$ and $v_{im} - v^{S,F}_i$ are the violations at the time horizon.

$$(Q^S_i - s_i)(v^C_{im} - v^{C,F}_i) = 0 \quad i \in S, m \in M \tag{5.40}$$
$$s_i(v^S_{im} - v^{S,F}_i) = 0 \quad i \in S, m \in M \tag{5.41}$$

If the last visit is before the time horizon, we want to ensure that the violations after the time horizon $v^{C,F}_i$ and $v^{S,F}_i$ equals zero. Constraints (5.42) and (5.43) ensure this.

$$v^{C,F}_i (1 - \sum_{m \in M} \delta^{\overline{T}}_{im}) = 0 \qquad i \in S \backslash \{d\} \tag{5.42}$$
$$v^{S,F}_i (1 - \sum_{m \in M} \delta^{\overline{T}}_{im}) = 0 \qquad i \in S \backslash \{d\} \tag{5.43}$$

Constraints (5.44) and (5.45) specify the relationship between the violations after the time horizon and the violations between this station visit and the previous station visit. These constraints must be included to ensure that the violations after horizon does not take a value higher than the actual violations.

$$v_i^{C,F} \cdot \theta_{im} \cdot \delta_{im} \leq v_{im}^C \qquad i \in S \backslash \{d\}, m \in M \tag{5.44}$$

$$v_i^{S,F} \cdot \theta_{im} \cdot \delta_{im} \leq v_{im}^S \qquad i \in S \backslash \{d\}, m \in M \tag{5.45}$$

Deviations

We also want to capture the deviation at time $\overline{T}$. The deviation is defined as the difference between the optimal state O_i at time $\overline{T}$ and the actual load s_i at time $\overline{T}$. Constraints (5.46) and (5.47) capture this.

$$d_i \geq O_i - s_i \quad i \in S \tag{5.46}$$

$$d_i \geq s_i - O_i \quad i \in S \tag{5.47}$$

The total deviation is

$$\sum_{i \in S} d_i$$

Constraints (5.30), (5.32), (5.33), (5.35), and (5.37) - (5.45) are nonlinear and need to be linearized before implementation. The linearization can be seen in Section A.2 in Appendix A.

5.5 Objective Function

In the objective function (5.48), we want to minimize the total violations within the time horizon $\overline{T}$, at the same time as we want to minimize the total deviation at time $\overline{T}$. In addition to this, we want to maximize the reward given for initiating trips where the arrival times exceed the time horizon. As this objective function consists of three different objectives, this problem is categorized as a multi-objective optimization problem. To solve this, we have chosen to use the weighted-sum method where each objective is assigned a weight. w^v, w^d and w^r are weights for the total violations, deviations and reward, respectively.

In this section, the objective function is presented, together with the reward function and its associated constraints.

$$\min w^v \Big(\sum_{i \in S, m \in M} (v_{im}^C + v_{im}^S) + \sum_{i \in S} (v_i^{C,f} + v_i^{S,f}) - \sum_{i \in S} (v_i^{S,F} + v_i^{C,F}) \Big) + w^d \Big(\sum_{i \in S} d_i \Big) - w^r \text{Reward} \tag{5.48}$$

The effect of visiting a station, in terms of reduction in violations, is first visible after the stations are visited. Hence, a vehicle is not motivated to initiate a trip to a station after the time horizon if violations are only minimized within the time horizon. Consequently, we need to introduce rewards for these initiated trips. As we do not impose a time constraint on this last visit, a challenge arises when the most optimal station to visit after $\overline{T}$ is undesirably far away. Therefore, we impose a reward for starting a station visit that takes place after $\overline{T}$, and a penalty associated with the driving time to that particular station.

There are different ways to approach the reward function, but in this thesis we have chosen to incentivize the service vehicles to choose a station that has high deviation as we believe this incorporates the rolling horizon aspect in a good way. As mentioned earlier, deviation is derived using the optimal state, and hence accounts for future demand. The service vehicles receive a reward that equals the deviation at the station they choose to visit. This reward is denoted r_i^D and is weighted by w^D. In addition, a penalty t_v^f, equal to the driving time to the chosen station, is subtracted from the reward with weight w^T. The reward function then becomes:

$$w^D \sum_{i \in S} r_i^D - w^T \sum_{v \in V} t_v^f$$

Even though a particular station has high deviation at the time horizon, it does not make sense to drive to a station that has too many bicycles if we already have a full service vehicle. Also, it does not make sense to drive to an empty station if the service vehicle also is empty. Therefore, we want to impose some restrictions on which stations the different service vehicles are allowed to visit as their last visit. Table 5.2 lists the notation used.

Table 5.2: Notation used for modelling the reward function

Sets	
S^-	Delivery stations with negative net demand
S^+	Pickup stations with positive net demand
Parameter	
I	Loading constant implying when the service vehicle is restricted to take certain trips
w^D	Weight for deviation
w^T	Weight for driving time
Variables	
s_v^V	Vehicle load at time $\overline{T}$ for vehicle v
r_i^D	Reward, derived from deviation, for driving to station i after the time horizon
t_v^f	Driving and handling time left for service vehicle v to the last planned station visit at the time horizon $\overline{T}$
σ_v^L	1 if service vehicle load s_v^V is below I, 0 otherwise
σ_v^H	1 if service vehicle load s_v^V is above $Q_v^V - I$

The service vehicle load s_v^V at time $\overline{T}$ is determined by restrictions (5.49) and (5.50). Restrictions (5.51) state that the reward r_i^D for visiting station i must be less than the deviation d_i at station i, whereas restrictions (5.52) states that there can only be a reward if the station actually is visited after the time horizon. If the service vehicle load s_v^V at time $\overline{T}$ is less or equal to the loading constant I, service vehicle v can only visit pickup stations. These stations are determined by the set S^+. Likewise, if the service vehicle load s_v^V at time $\overline{T}$ is greater or equal to the vehicle capacity Q_v^V minus the loading constant I, service vehicle v can only visit delivery stations. These stations are determined by the set S^-. The binary variables σ_v^L and σ_v^H capture this along with restrictions (5.53) - (5.56). If the service vehicle load is in the interval $[I, Q_v^V - I]$, there are no restrictions on where the service vehicle can drive. Restrictions (5.57) determines the remaining driving time from $\overline{T}$ until the final station for service vehicle v.

$$s_v \geq l^V_{imv} - (2 - \delta_{jn} + \delta_{im} - x_{imjnv})O^V_v \quad i,j \in S, m,n \in M, v \in V \tag{5.49}$$

$$s_v \leq l^V_{imv} + (2 - \delta_{jn} + \delta_{im} - x_{imjnv})O^V_v \quad i,j \in S, m,n \in M, v \in V \tag{5.50}$$

$$r^D_i \leq d_i + Q^S_i(1 - \sum_{m \in M} \delta_{im}) \quad i \in S \tag{5.51}$$

$$r^D_i \leq \sum_{m \in M} \delta_{im} Q^S_i \quad i \in S \tag{5.52}$$

$$s^V_v \geq I - I\sigma^L_v \quad v \in V \tag{5.53}$$

$$s^V_v \leq Q^V_v - I + I\sigma^H_v \quad v \in V \tag{5.54}$$

$$x_{imd(v)1v} \leq (1 - \sigma^L_v) + (1 - \delta_{im}) \quad i \in S^- \tag{5.55}$$

$$x_{imd(v)1v} \leq (1 - \sigma^H_v) + (1 - \delta_{im}) \quad i \in S^+ \tag{5.56}$$

$$t^f_v = \sum_{i \in S} \sum_{m \in M} x_{imd(v)1v} \cdot t_{im} - \overline{T} \qquad v \in V \tag{5.57}$$

Constraints (5.57) are nonlinear, and therefore need to be linearized before implementation. A linear version is presented in Section A.3 in Appendix A.

The complete objective function is formulated in (5.58).

$$\min w^v[\sum_{i \in S, m \in M} (v^C_{im} + v^S_{im}) + \sum_{i \in S}(v^{C,f}_i + v^{S,f}_i) - \sum_{i \in S}(v^{S,F}_i + v^{C,F}_i)] + w^d \sum_{i \in S} d_i - w^r[w^D \sum_{i \in S} r^D_i - w^T \sum_{v \in V} t^f_v] \tag{5.58}$$

Chapter 6

Column Generation Heuristic for the DDBRS

As concluded in the literature survey in Chapter 3, it is inefficient to use exact solution methods to solve bicycle rebalancing problems due to the complexity and large solution space. With commercial optimization software, and within reasonable computational time, we are only able to solve the mathematical model presented in Chapter 5 to optimality if there are maximum eight stations and two service vehicles in the system. BSSs today have hundreds of stations. Hence, a heuristic solution method is implemented.

In this thesis, a column generation heuristic is applied to the subproblem with the aim of solving realistic problem sizes within a reasonable computational time. By using a heuristic instead of a commercial optimization solver, we are no longer guaranteed that the solutions found are optimal. However, we believe that adequate solutions can be found, and that the gain of reducing the computational time is more valuable than the loss caused by a reduced solution quality. As mentioned, column generation heuristic applied to BSSs has not been discovered in literature despite its success on vehicle routing problems. As good solutions to the subproblems can be generated by empirical analyses, and as most of the flow variables in the exact solution method will take a value of zero, column generation is assumed to perform well. A general overview of our column generation heuristic is given in Section 6.1. Second, three different variations of the master problem are elaborated in Section 6.2. Further, the initialization algorithm for generating initial columns is described in Section 6.3. Last, the pricing problem is discussed in Section 6.4.

6.1 Overview of Algorithm

Column generation algorithms have been specially designed for solving mathematical programs with a huge number of variables. Unfortunately, this method suffers from slow convergence, something that limits its efficiency and usability (Moungla et al., 2010). Hence, a method that combines exact optimization and heuristics is proposed. Initial columns are generated with a heuristic branching algorithm, whereas the optimal combination of these columns is determined by Xpress in the master problem. Moreover, the pricing problem is solved with a simple heuristic with the aim of finding new and better columns to add to the master problem. Figure 6.1 illustrates the steps in our column generation heuristic.

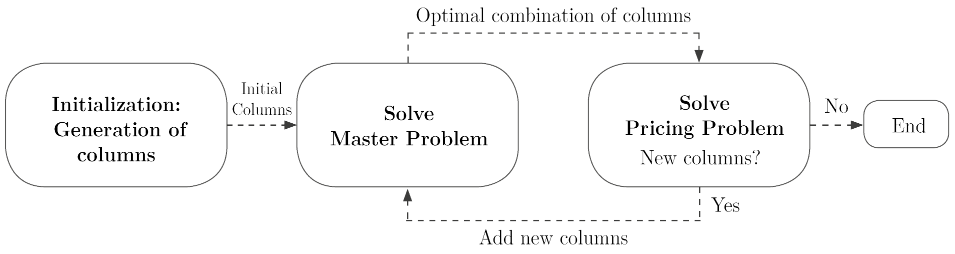


Figure 6.1: The steps in our column generation heuristic

In the initialization, a set of columns is generated for each service vehicle through a heuristic. Each column corresponds to one possible route. The master problem selects the optimal combination of columns by allocating one route to each service vehicle. As the master problem only considers a subset of all possible columns, it is, in fact, a *restricted master problem*. However, we will refer to it as the *master problem* (MP) further in this thesis. The pricing problem heuristic creates a given number of new routes that include critical non-visited stations, and adds these to the master problem. The algorithm iterates between the master problem and the pricing problem for a given number of times. With a pricing problem heuristic, there is no way of knowing how far away the local optimum found in the master problem is from the global optimum. Hence, smart initialization of columns is a crucial step in our column generation heuristic. Better initial columns provide a better first iteration solution, and will accelerate the convergence process. Besides, solving the master problem is the most time-consuming part, especially with large test instances and many columns. Consequently, a considerable amount of effort is put on the initialization of columns.

6.2 Master Problem

The master problem (MP) determines the optimal combination of columns by allocating one route to each service vehicle. Recall that the rebalancing problem has several variables determining, among other things, flow, loading quantity, arrival time, inventory level at service vehicle and station, violations and deviations. Instead of determining these variables within the MP, some of these variables may be predetermined as part of the initialization and rather be input parameters to the MP. A reduction in computational time in the MP is expected, along with a reduction in the solution quality, i.e. a higher objective value. Thus, we expect a trade-off between the computational time and the quality of the solution. To examine how the computational time and the solution quality are influenced by the amount of predetermined information, we present three different versions of the MP.

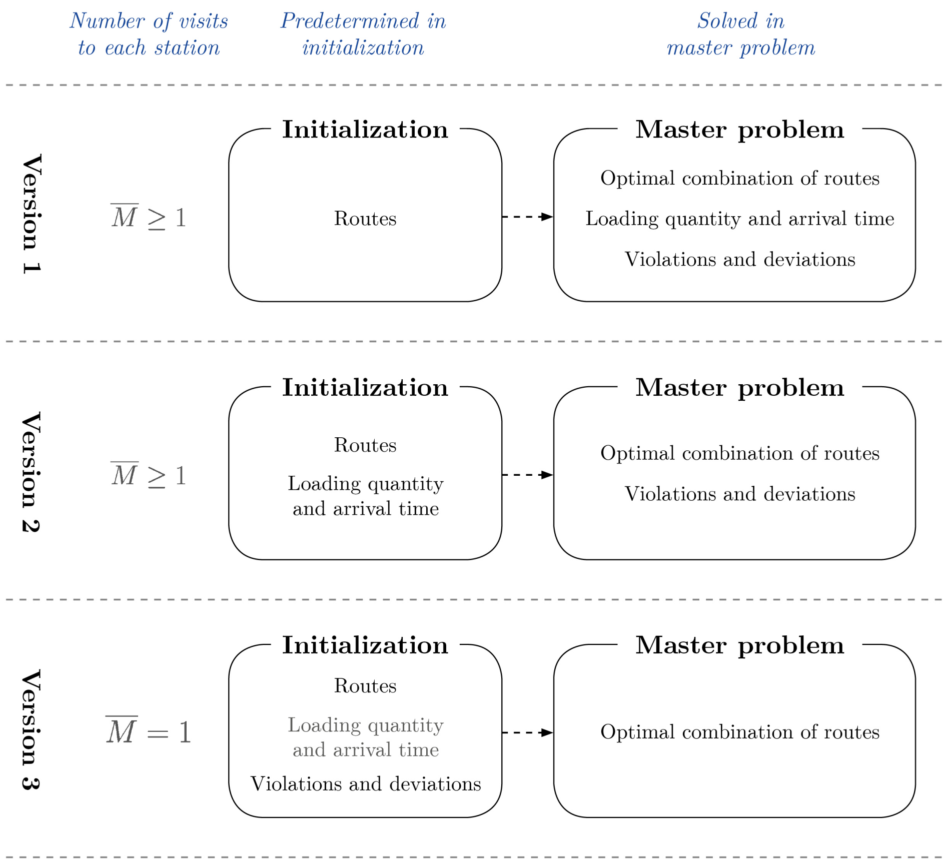

Number of visits to each station
Predetermined in initialization
Solved in master problem
Version 1
$\overline{M} \geq 1$
Initialization
Routes
Master problem
Optimal combination of routes
Loading quantity and arrival time
Violations and deviations
Version 2
$\overline{M} \geq 1$
Initialization
Routes
Loading quantity and arrival time
Master problem
Optimal combination of routes
Violations and deviations
Version 3
$\overline{M} = 1$
Initialization
Routes
Loading quantity and arrival time
Violations and deviations
Master problem
Optimal combination of routes

Figure 6.2: The three different versions of the master problem.

The three versions have an increasing amount of predetermined information and are illustrated in Figure 6.2. In version 1, geographical routes for the service vehicles are created in the initialization process, whereas the loading quantities and the arrival times are determined in the MP. In version 2, also the loading quantities and the arrival times are predetermined in the initialization. However, the master problem has some flexibility to change the given loading quantities. In version 3, the loading quantities and the arrival times are entirely pre-determined by the initialization algorithm. In version 3, there is also added a restriction stating that no more than one visit to each station is allowed, i.e. $\overline{M} = 1$. Both version 1 and 2 inherit all the constraints from the mathematical model presented in Chapter 5. As version 3 restricts each station to have at most one visit, violations and deviations can be predetermined, and the mathematical model of the MP is simplified significantly.

6.2.1 Version 1: Loading Quantity and Arrival Time Determined in MP

In version 1 of the MP, the columns only contain information about the geographical routes. The loading quantities and the arrival times are determined in the MP. Each service vehicle v has a set of columns R_v. The integer parameter A_{ijvr} counts how many times service vehicle v drives directly from station i to station j in route r. The binary variable λ_{vr} takes a value of 1 if route r is allocated to service vehicle v, and zero otherwise. In addition to the notation presented in Section 5.3, the notation in Table 6.1 is used.

Table 6.1: Additional notation used for modelling the MP version 1

Sets	
R_v	Set of possible routes for v
Indices	
r	Route $r \in R_v$
Parameters	
A_{ijvr}	No. of times service vehicle v drives directly from station i to station j if allocated route r
Variables	
λ_{vr}	1 if service vehicle v is allocated route r, 0 otherwise

In this version of the MP, it is possible to have multiple visits to a station, i.e. $\overline{M} \geq 1$, which means that the A_{ijvr} parameters can take a value greater than one. Recall that the mathematical model in Chapter 5 uses the indices m and n to keep track of the station visit number. $m = 1$ refers to the first visit at a station. Since the A_{ijvr} parameters do not track the station visit number, the flow variables x_{imjnv} are still needed to ensure feasible routes. Consequently, the number of variables increases, but the outcome space of the flow variables decreases compared to the original formulation presented in Chapter 5. This complicates the model compared to a situation where flow variables x_{imjnv} are completely replaced by the interior representation $\sum_{r \in R_v} A_{ijvr} \cdot \lambda_{vr}$.

Version 1 of the MP inherits Constraints (5.1)-(5.47) presented in Chapter 5. In addition, Constraints (6.1)-(6.3) are needed. Constraints (6.1) connect the flow variables x_{imjnv} to the interior representation of the flow variables. Constraints (6.2) ensure that each service vehicle is allocated at most one route, and Constraints (6.3) are binary restrictions. The objective function is the same as presented in Section 5.5.

$$\sum_{m \in M} \sum_{n \in M} x_{imjnv} = \sum_{r \in R_v} A_{ijvr} \lambda_{vr} \qquad i, j \in S, v \in V \tag{6.1}$$

$$\sum_{r \in R_v} \lambda_{vr} \leq 1 \qquad v \in V \tag{6.2}$$

$$\lambda_{vr} \in \{0, 1\} \qquad v \in V, r \in R_v \tag{6.3}$$

6.2.2 Version 2: Loading Quantity and Arrival Time Predetermined

In version 2 of the MP, loading quantities and arrival times are also included as input parameters along with the geographical routes. In addition to the notation presented in Section 5.3, and in MP version 1, the notation in Table 6.2 is used.

Table 6.2: Additional notation used for modelling the MP version 2

Parameters	
Q^L_{ivr}	Sum of loading quantities at station i for service vehicle v if allocated route r
Q^U_{ivr}	Sum of unloading quantities at station i for service vehicle v if allocated route r
T_{ivr}	Sum of arrival times at station i for service vehicle v if allocated route r
F	Flexibility parameter on predetermined loading quantity at each station

Q^L_{ivr} and Q^U_{ivr} are the interior representation of the loading and unloading quantities, respectively. As multiple visits to a station is allowed, the loading parameter is the sum of all loading quantities at station i for service vehicle v if allocated route r. T_{ivr} is the arrival times for service vehicle v at station i if allocated route r. Likewise, the parameter is the sum of all individual arrival times at station i for service vehicle v if allocated route r. To demonstrate, if route 1 for service vehicle 1 contains three station visits noted as (station id, loading quantity, arrival time) = (35, 4, 0), (40, -3, 3), (35, 5, 6), the total loading quantity at station 35, vehicle 1, route 1 is $Q^L_{35,1,1} = 4+5 = 9$, and the total arrival time at station 35, vehicle 1, route 1 is $T_{35,1,1} = 0+6 = 6$.

To add flexibility to the model, an additional parameter F is included. This parameter is referred to as the *flexibility parameter*. This allows the MP to choose a loading quantity in the interval $[Q_{ivr} - F, Q_{ivr} + F]$. A higher value of F is expected to improve the quality of the solution while increasing the computational time. The time variable is allowed to change correspondingly. When the flexibility parameter F equals the service vehicle's capacity, the MP has full flexibility of the load, meaning it will equal MP version 1. If F is set to zero, the MP has no flexibility, and, given that $\overline{M} = 1$, this version of the MP equals MP version 3.

As in version 1, the interior representation cannot replace the loading and arrival time variables due to possible multiple visits to a station. Hence, the variables q^L_{imv}, q^U_{imv} and t_{imv} presented in Chapter 5 are still needed. If discarded, infeasible routes could be created in situations where a service vehicle visits the same station multiple times. Constraints (5.1)-(5.47) presented in Chapter 5 and Constraints (6.1)-(6.3) from version 1 are inherited in addition to Constraints (6.4)-(6.9). Constraints (6.4) - (6.7) are the interior representation of the loading and unloading variables, respectively, whereas Constraints (6.8) and (6.9) are the interior representation of the arrival time variable. Recall that T^H is the handling time for one bicycle.

The objective function is the same as presented in Section 5.5. As this is a minimization problem, the solution from version 2 is an upper bound to the solution from version 1, meaning that the objective value from version 2 is equal or worse than the objective value of version 1.

$$\sum_{m\in M} q^L_{imv} \leq \sum_{r\in R_v} (Q^L_{ivr} + F)\lambda_{vr} \qquad i \in S, v \in V \tag{6.4}$$

$$\sum_{m\in M} q^L_{imv} \geq \sum_{r\in R_v} (Q^L_{ivr} - F)\lambda_{vr} \qquad i \in S, v \in V \tag{6.5}$$

$$\sum_{m\in M} q^U_{imv} \leq \sum_{r\in R_v} (Q^U_{ivr} + F)\lambda_{vr} \qquad i \in S, v \in V \tag{6.6}$$

$$\sum_{m\in M} q^U_{imv} \geq \sum_{r\in R_v} (Q^U_{ivr} - F)\lambda_{vr} \qquad i \in S, v \in V \tag{6.7}$$

$$\sum_{m\in M} t_{im} \leq \sum_{v\in V}\sum_{r\in R_v} (T_{ivr} + F \cdot T^H)\lambda_{vr} \qquad i \in S \tag{6.8}$$

$$\sum_{m\in M} t_{im} \geq \sum_{v\in V}\sum_{r\in R_v} (T_{ivr} - F \cdot T^H)\lambda_{vr} \qquad i \in S \tag{6.9}$$

6.2.3 Version 3: Violations and Deviations Predetermined

In contrast to the prior versions, version 3 of the MP restricts each station to have no more than one visit, i.e. $\overline{M} = 1$. Thus, the m index in the formulation is no longer necessary, and the flow, loading, and arrival time variables are replaced by the interior representation. Even though this replacement makes the model a lot simpler, version 3 is reformulated and simplified considerably further by exploiting the fact that each station is visited no more than once.

When $\overline{M} = 1$, the MP determines a combination of routes such that each route consists of unique station visits. Hence, we are now able to predetermine violations and deviations at the visited stations. This was not achievable in the previous versions, as the combination of routes, and where the routes intersected, were unknown. By predetermining violations, deviations, and the rewards, the number of binary variables in the MP decreases considerably, and a reduction in computation time is expected. Unlike the two first versions of the MP where the mathematical model in Chapter 5 was extended, the third version introduces entirely new notation and constraints. New notation is summarized in Table 6.3.

Table 6.3: Additional notation used for modelling the MP version 3

Parameters	
A_{ivr}	1 if service vehicle v visits station i in route r
V_{vr}	Violations prevented if service vehicle v is allocated route r
D_{vr}	Deviation improvement if service vehicle v is allocated route r
R_{vr}	Reward if service vehicle v is allocated route r
V	Total violations in the system if no service vehicles are in operation
D	Total deviation in the system if no service vehicles are in operation
Variables	
λ_{vr}	1 if service vehicle v is allocated route r

A_{ivr} indicates whether a visit to station i is included in route r for service vehicle v. *Violations prevented* V_{vr} indicates how many violations that are prevented if service vehicle v drives route r. *Deviation improvement* D_{vr} is the deviations that occur with no operating service vehicles, minus the deviations that occur when service vehicle v is allocated route r. R_{vr} is the reward that service vehicle v acquire if allocated route r. Recall that the reward is associated with the last station visit in a route. V and D are the total violations and the deviations, respectively, in the system if no service vehicles are in operation. The binary variable λ_{vr} is 1 if service vehicle v is allocated route r, and 0 otherwise.

Due to the new formulation, the objective function in version 3 is slightly changed. However, the objective value is equal for all versions if the chosen geographical routes, loads and arrival times are identical. The objective function is formulated in (6.10).

$$\min \quad w^v(V - \sum_{v\in V}\sum_{r\in R_v} V_{vr}\lambda_{vr}) + w^d(D - \sum_{v\in V}\sum_{r\in R_v} D_{vr}\lambda_{vr}) - w^r(\sum_{v\in V}\sum_{r\in R_v} R_{vr}\lambda_{vr}) \tag{6.10}$$

As we no longer need to determine violations, deviations and rewards in the MP, all the constraints presented earlier are completely replaced by Constraints (6.11) - (6.13). Constraints (6.11) state that each service vehicle can drive at most one route, Constraints (6.12) restrict the stations from having more than one station visit, and Constraints (6.13) are binary restrictions.

$$\sum_{r \in R_v} \lambda_{vr} \leq 1 \qquad v \in V \tag{6.11}$$

$$\sum_{v \in V} \sum_{r \in R_v} A_{ivr} \lambda_{vr} \leq 1 \qquad i \in S \tag{6.12}$$

$$\lambda_{vr} \in \{0,1\} \qquad v \in V, i \in S \tag{6.13}$$

Although the new model is considerably simpler and will reduce the computational time, the solution quality may also reduce. The objective value obtained from master problem version 3 O_{V3} is equal or worse than the solution from version 1 and 2, i.e. $O_{V1} \leq O_{V2} \leq O_{V3}$.

6.3 Initialization of Columns

In this section, the process of generating initial columns, i.e. routes for each service vehicle, is described in detail. As discussed, good initialization is vital for the convergence process and for the quality of the solution. Therefore, empirical analyses and proper problem characteristics are taken into account when initializing columns.

The initial columns are created with a branching algorithm. Each path in the tree represents one geographical route, and each node represents one station visit. Figure 6.3 illustrates an example of a branching tree with four different routes, each of the routes containing three station visits. The first route contains the stations 1, 10 and 2.

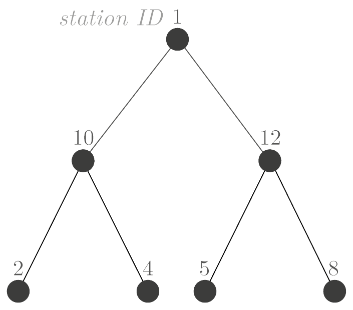


Figure 6.3: A branching tree

6.3.1 Overview of Branching Algorithm

The branching algorithm is summarized by Algorithm 1. This algorithm is run for all the service vehicles. As described in Section 5.2, we assume that the first station visit is given. In Line 1 in the algorithm, a list R is created, containing all routes under construction. Initially, this list contains a single route with only the first predetermined station visit. As long as there are routes in list R, each route is either extended, moved to a list F containing all finished routes, or both. This process is illustrated in Lines 3-15. If the duration of a route is shorter than $\overline{T}$, the route is extended by the branching algorithm. If the duration of a route is longer than $\overline{T}$ minus a constant C, the route is moved to the list F containing all finished routes. This means that if the duration of a route is in the interval $[\overline{T}-C,\overline{T}]$, the route is both extended and moved to the list F. We include routes that are shorter than the time horizon $\overline{T}$ in F to stimulate exploration of the search space.

```
   Data: $S$ := First station visit; $\overline{T}$ := time horizon;
   Result: F: Set of finished routes
1  R: List with routes under construction ← $S$
2  while size of $R > 0$ do
3      for each route $r$ in $R$ do
4          Estimate loading quantities and arrival times
5          if duration of $r < \overline{T}$ then
6              Determine subset $S^R$ of stations that can be added to route $r$
7              Calculate criticality score for each station in subset $S^R$
8              $R$ ← create new routes by extending $r$ and insert into R
9          end
10         if duration of $r > \overline{T} - C$ then
11             insert $r$ into $F$
12         end
13         remove $r$ from $R$
14     end
15 end
```

Algorithm 1: Branching algorithm

Line 6-8 in Algorithm 1 describe the process where route r is extended. Certain problem specific restrictions are enforced on the branching to ensure good initial columns. When the branching algorithm picks the next station visit in a route, it can only choose between a subset S^R of all stations. The process of determining which stations that can be added to route r corresponds to Line 6 in the algorithm. The subset S^R is determined both by clustering and problem specific filtering. These procedures are described further in Sections 6.3.2 and 6.3.3, respectively.

When the subset S^R is determined, each of the stations in the subset is given a *criticality score* indicating the importance of visiting the station when total violations are minimized. This process corresponds to Line 7 in algorithm 1. How the criticality score is calculated is elaborated in detail in Section 6.3.4.

In Line 8 in Algorithm 1, the extension of a route is performed. A branching constant B is defined, expressing the number of branches created from each node. If subset S^R consists of both pickup and delivery stations, the branching algorithm creates up to B new branches with pickup stations, and B new branches with delivery stations. If subset S^R only contains either pickup or delivery station, only up to B new branches are created. Consequently, twice as many branches are created when the service vehicle can visit both pickup and delivery stations. The branching algorithm selects the B stations with the highest criticality score.

Since the decision regarding whether a route should be extended or not is dependent on the duration of the route, the arrival times and the loading quantities must be estimated. This process corresponds to Line 4 in Algorithm 1, and is further described in Section 6.3.5.

6.3.2 Determine Subset S^R: Clustering

Line 6 in Algorithm 1 corresponds to the process of determining a subset S^R of stations that can be added to a route. When columns are created for one service vehicle, the branching algorithm does not consider the columns already created for other service vehicles. This may result in many similar routes for the different service vehicles, and poor exploration of the search space. To increase diversification of columns, clustering is implemented. A service vehicle is only allowed to visit stations that are in the service vehicle's cluster. Thus, all stations in S^R must be in the service vehicle's cluster. Clusters are either static and generated once, or dynamic and created each time new routes are generated.

We propose a clustering model, and several assumptions are made. One cluster is created for each service vehicle. Thus, service vehicle v's cluster is referred to as cluster v. A service vehicle is only allowed to visit stations in its cluster. However, the same station can belong to multiple clusters. When generating clusters, the stations are divided into three sets; stations with low net demand, medium net demand and high net demand. The absolute value of the net demand is considered when dividing the stations into sets. We assume that stations with low net demand do not need to be visited, and are therefore not part of any clusters. The stations with medium net demand are required to belong to one cluster, and the stations with high net demand are required to belong to two clusters. The stations with net demand at the C^Hth percentile belong to the set with high net demand, and the stations with net demand in the C^Lth percentile are labeled as stations with low net demand. The remaining stations are considered to have medium demand.

Further, we assume that it is reasonable to minimize the driving time within a cluster. The driving time within cluster v is defined as the sum of the driving times from service vehicle v's initial station to all the stations in the cluster. Moreover, to ensure an approximately constant number of bicycles within a cluster, the sum of all stations' net demand within each cluster is preferred to be close to zero. Lastly, we want the clusters to contain an approximately equal number of stations.

Figure 6.4 illustrates a BSS that contains 18 stations. C^H is set to 75, and C^L is set to 25. The red nodes indicate stations with high net demand, black nodes medium net demand, and grey nodes low net demand. Station 75 and 88 are the initial stations for service vehicle 1 and 2, respectively. The arrows from the initial stations illustrate the two clusters.

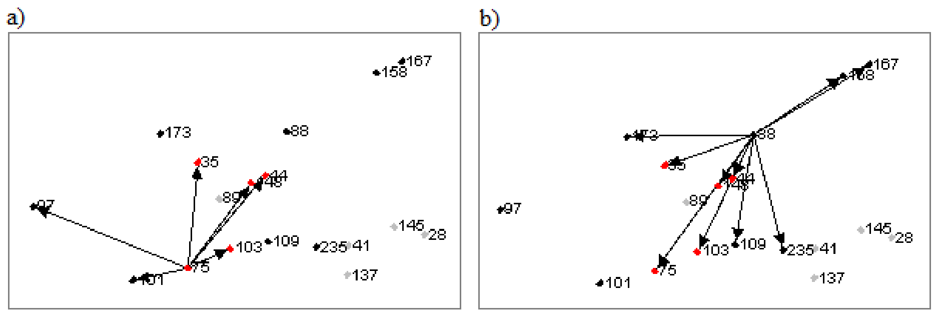


Figure 6.4: a) cluster for service vehicle 1, b) cluster for service vehicle 2

Notation used for modelling the clustering problem is summarized in Table 6.4. The mathematical formulation is presented by the objective function (6.14), and by Constraints (6.15)-(6.20). The model is solved with Xpress.

Table 6.4: Notation for clustering model

Sets	
S	Stations
V	Vehicles
Parameters	
D_i	Net customer demand at station i
C_i	Number of clusters station i should belong to
T_{ij}	Driving time from station i to station j
$o(v)$	Initial station visit for service vehicle v
w^K	Weight total driving time
w^N	Weight total net demand
w^Z	Weight size difference
Variables	
x_{iv}	1 if station i is in cluster v, 0 otherwise
d_v	Absolute value of total net demand in cluster v
n^{max}	Number of stations in largest cluster
n^{min}	Number of stations in smallest cluster

$$min \qquad w^K \sum_{v \in V} \sum_{i \in S} T_{o(v)i} x_{iv} + 4w^N \sum_{v \in V} d_v + 6w^Z (n^{max} - n^{min}) \tag{6.14}$$

$$\sum_{v \in V} x_{iv} = C_i \qquad i \in S \tag{6.15}$$

$$d_v \geq -\sum_{i \in S} D_i x_{iv} \qquad v \in V \tag{6.16}$$

$$d_v \geq \sum_{i \in S} D_i x_{iv} \qquad v \in V \tag{6.17}$$

$$n^{max} \geq \sum_{i \in S} x_{iv} \qquad v \in V \tag{6.18}$$

$$n^{min} \leq \sum_{i \in S} x_{iv} \qquad v \in V \tag{6.19}$$

$$x_{iv} \in \{0,1\} \qquad i \in S, v \in V \tag{6.20}$$

The first term in the objective function (6.14) minimizes the total driving time from the initial station within each cluster, the second term minimizes the absolute value of the total net demand within each cluster, and the third term minimizes the difference between the size of the largest cluster and the size of the smallest cluster. The weights w^K, w^N and w^Z are used to weight the three terms. If there are $|S|$ stations in the system, and $|V|$ service vehicles, we assume that the value of the first term is approximately $\frac{|S|}{|V|} \cdot 6$, the second term $\frac{|S|}{|V|} \cdot 1.5$, and the last term $\frac{|S|}{|V|}$. We want equal weighting of these terms when the weights have the same value. Thus, the second and third term are scaled by the factors 4 and 6, respectively.

Constraints (6.15) ensure that each station belongs to as many clusters as required. Constraints (6.16) and (6.17) capture the absolute value of the total net demand within a cluster. Constraints (6.18) and (6.19) capture the size of the largest and smallest clusters within the system. Binary restrictions of the x_{iv} variables are given by Constraints (6.20).

6.3.3 Determine Subset S^R: Filtering

In addition to clustering, filtering is applied on the subset S^R of stations that can be added to a route. Depending on the existing station visits in a route, we can sometimes predetermine that it is unreasonable to visit either a pick up station or a delivery station next. Thus, these stations can be filtered out of the subset S^R, and better initial columns can be created. The filtering rules applied are established based on empirical analyses from the BSS in Oslo.

If a route so far only consists of one station visit, i.e. the next station added is the second station visit in the route, the filtering rule depends on the service vehicle's initial load, and the station type of the first station visit. If the service vehicle's initial load is lower than a fixed value L^{min}, and the first station visit is to a delivery station, we assume that the service vehicle's load is approximately zero after the first visit. It is meaningless to visit a delivery station if the service vehicle is empty. Hence, the service vehicle is forced to visit a pickup station as its second station visit, and all delivery stations are filtered out of subset S^R. On the contrary, if the initial service vehicle load is higher than L^{high}, and if the first visit is to a pickup station, the service vehicle is forced to visit a delivery station as its second visit, and all pickup stations are filtered out of S^R. Otherwise, both station types can be visited, and none of the stations are filtered out of S^R. Figure 6.5 illustrates the filtering rules applied to the second station visit.

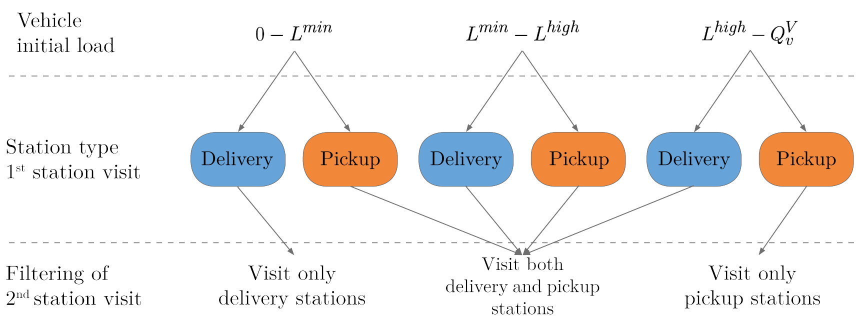

Vehicle initial load
$0 - L^{min}$
$L^{min} - L^{high}$
$L^{high} - Q_v^V$
Station type 1st station visit
Delivery
Pickup
Delivery
Pickup
Delivery
Pickup
Filtering of 2nd station visit
Visit only delivery stations
Visit both delivery and pickup stations
Visit only pickup stations

Figure 6.5: Filtering rules applied on second station visit

If the route so far contains more than two station visits, the filtering rule applied depends on the station type of the last two stations in the route. As the service vehicles have a limited docking capacity, we assume that it is inefficient to visit more than two pickup or two delivery stations consecutively. Observation of operational activity in Oslo supports this assumption. Thus, if the two last station visits are to pickup stations, all pickup stations are filtered out of S^R, and if the two last station visits are to delivery stations, all delivery stations are filtered out of S^R. Figure 6.6 illustrates these restrictions.

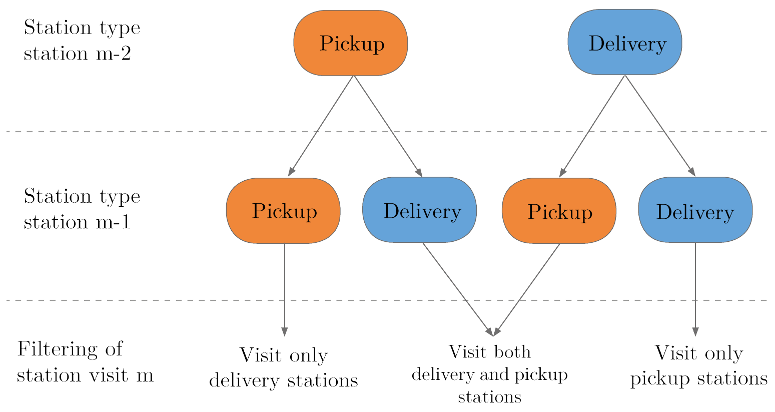

Station type station m-2
Pickup
Delivery
Station type station m-1
Pickup
Delivery
Pickup
Delivery
Filtering of station visit m
Visit only delivery stations
Visit both delivery and pickup stations
Visit only pickup stations

Figure 6.6: Filtering rules applied on station visit m

6.3.4 Calculate Criticality Score for each Station

Before the branching algorithm picks a new station to add to an existing route, each station in subset S^R is given a criticality score that expresses the importance of visiting the station. The arrival time at the last station visit in the existing route is referred to as the *current time, CT*. The criticality score for station i is based on four elements.

1. t_i: Time to violation
2. D_i: Net demand
3. T^D_{Ci}: Driving time
4. d_i: Deviation at $\overline{T}$

Element 1 captures the time until violations start occurring at station i if the station is not visited. If the station is already congested or starved, the time to violation is 0. Lower time to violation makes the station visit more critical. Time to violation t_i is calculated as expressed by Equations (6.21) and (6.22). Q^S_i is the station's capacity, l^{CT}_i is the station's load at CT, and D_i is the station's net demand.

$$\text{Pick up station: } t_i = \frac{Q^S_i - l^{CT}_i}{D_i} \tag{6.21}$$

$$\text{Delivery station: } t_i = \frac{-l^{CT}_i}{D_i} \tag{6.22}$$

Element 2 is net customer demand. A higher net demand makes the station visit more critical. Element 3 is the driving time from the current position C to station i. Geographically closer stations are prioritized. Element 4 is the deviation, i.e. the absolute value of the difference between the station's load at $\overline{T}$ if the station is not visited, and the station's optimal state at $\overline{T}$. The load $l^{\overline{T}}_i$ at station i at $\overline{T}$ if the station is not visited is calculated as expressed by Equations (6.23) and (6.24). l^{CT}_i is station i's load at CT. The deviation d_i is then calculated as expressed by Equation (6.25). Recall that o_i is the optimal state for station i.

$$\text{Pick up station: } l^{\overline{T}}_i = \max\{Q^S_i, l^{CT}_i + D_i(\overline{T} - CT)\} \tag{6.23}$$

$$\text{Delivery station: } l^{\overline{T}}_i = \max\{0, l^{CT}_i + D_i(\overline{T} - CT)\} \tag{6.24}$$

$$d_i = |o_i - l^{\overline{T}}_i| \tag{6.25}$$

The elements are given the weight coefficients w^t, w^n, w^k and w^o, respectively. The total criticality score c_i for station i is expressed in Equation (6.32).

$$c_i = -w^t t_i + w^n D_i - w^k T^D_{Ci} + w^o d_i \tag{6.26}$$

6.3.5 Estimation of Loading Quantity and Arrival Time

Whether a route is extended or not in the branching algorithm depends on the duration of the route. Hence, the route duration is estimated. In addition, the loading quantities and arrival times are included as input to the master problem in version 2 and 3 and must therefore be predetermined.

The arrival time at station j is determined based on the arrival time t_i at station i, a fixed parking time T^P, a handling time T^H proportional to the loading quantity q_i at the previous station i, and the driving time T^D_{ij} between the previous and the current station. All of these parameters are given except the loading quantities. The arrival time t_j at station j is as expressed in Equation (6.27).

$$t_j = t_i + T^P + T^H q_i + T^D_{ij} \tag{6.27}$$

A greedy heuristic is used to estimate the loading quantities at each station visit in a route. If a node in the branching tree is part of several routes, different loading quantities are set for each route. All loading quantities are updated each time a route is extended. The loading quantities are set chronologically for a route, meaning that the load at the first station visit is first determined, then the load at the second station visit, and so on.

As discussed earlier, we assume that it is reasonable to visit one or maximum two pickup or two delivery stations consecutively. Moreover, we assume that the service vehicles strive to utilize their docking capacity. In general, it is not sensible to load the service vehicle full at a pickup station if the next station visit also is to a pickup station. However, this makes sense if the next station visit is to a delivery station. To control these assumptions, a maximum loading quantity Q^{max} is introduced. This parameter restricts the loading quantity at station i from being higher than its value. If the station in question and the succeeding station are both pickup or delivery stations, Q^{max} is set to half the service vehicle's capacity Q^V. Otherwise, Q^{max} is set to the full vehicle capacity. This is expressed in Equations (6.28) and (6.29). Furthermore, the service vehicle load l^V_v right before the visit and the station load l_i^S right before the visit are taken into account. The capacity at station i is noted Q_i^S. The loading algorithm estimates the loading and unloading quantities at each station in a route as expressed in Equations (6.30) and (6.31).

$$\text{Next station same pickup/delivery type: } Q^{max} = \frac{Q^V}{2} \tag{6.28}$$

$$\text{Next station opposite pickup/delivery type: } Q^{max} = Q^V \tag{6.29}$$

$$\text{Pickup stations: } q_i^L = min\{Q^{max}, Q^V - l_v^V, l_i^S\} \tag{6.30}$$

$$\text{Delivery stations: } q_i^U = min\{Q^{max}, l_v^V, Q_i^S - l_i^S\} \tag{6.31}$$

Occasionally, if two delivery stations are visited consecutively, the service vehicle might not be able to unload the remaining bicycles to the second station. Similarly, if the service vehicle visits two pickup stations consecutively, it might have available slots after visiting the second station. The remaining bicycle load/available slots at the service vehicle after the two station visits are noted e. In these situations, the algorithm will perform a *regret*.

The regret function takes the loading algorithm two steps back, and the loading quantities at the two last stations are re-estimated. This time, the Q^{max} parameter is set to half the service vehicle's capacity plus the remaining service vehicle load/slots e. If the loading/unloading quantity at the first station increases as a consequence of this, the arrival time at the second station is delayed. Thus, the station load at this point is different as more customers have picked up/delivered bicycles before the arrival. Therefore, the loading/unloading quantity at the second station is also re-estimated. After regret is executed, none of the stations can receive more bicycles even though $e > 0$. Hence, the regret function is only performed once. Figure 6.7 illustrates the regret function algorithm. The illustration shows the procedure when two delivery stations are visited consecutively. The r parameter limits the regret function to one iteration.

An example is given in Figure 6.8. The service vehicle's capacity is 23. The first station is a pickup station, and the second station is a delivery station. Thus, the maximum loading quantity Q^{max} at the first station is set to the service vehicle's capacity. There are 23 available spots in the service vehicle, and there are 40 bicycles at the station. The loading quantity is set to $min\{Q^{max}, s_v, Q_i - s_i\} = min\{23, 23, 40\} = 23$. Next, there are two consecutive visits to delivery stations, hence $Q^{max} = 12$ at the first of these. The

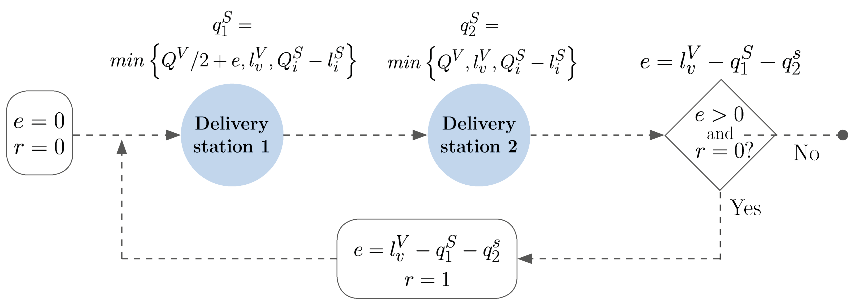

$q_1^S = \min\left\{Q^V/2 + e, l_v^V, Q_i^S - l_i^S\right\}$
$q_2^S = \min\left\{Q^V, l_v^V, Q_i^S - l_i^S\right\}$
$e = l_v^V - q_1^S - q_2^s$
$e = 0$ $r = 0$
Delivery station 1
Delivery station 2
$e > 0$ and $r = 0$?
No
Yes
$e = l_v^V - q_1^S - q_2^s$ $r = 1$

Figure 6.7: If the service vehicle's load is greater than zero after visiting two delivery stations consecutively, the regret function rewinds the loading algorithm two steps, and the loading quantities are re-estimated.

unloading quantity is set to $min\{12, 23, 16\} = 12$, and at the third station the unloading quantity is set to $min\{23, 11, 9\} = 9$ bicycles. As the service vehicle still has a remaining bicycle load $e = 2$, the regret function is executed, and the unloading quantities for the last two stations are re-estimated. Q^{max} at the first delivery station is now set to $12 + e$ and the unloading quantities become $min\{14, 23, 16\} = 14$ and $min\{23, 11, 9\} = 9$.

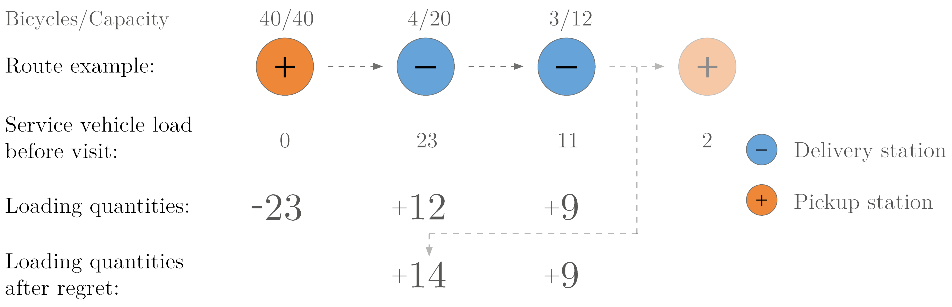


Figure 6.8: Example of how the greedy heuristic estimates loading quantities.

6.4 Pricing Problem

As mentioned, we hypothesize that smart initialization of good columns diminishes the importance of having a complex pricing problem (PP). However, discarding the pricing problem may be unwise as it can help stimulate exploration of the search space and create columns that were omitted in the initialization. Developing a heuristic pricing problem is challenging, as the reduced cost of non-included columns is unavailable. We propose a approach to the pricing problem based on simple mechanisms and assumptions. By considering the output of the master problem, the pricing problem identifies stations that 1) are not part of any of the columns chosen by the master problem, and 2) contribute the most to violations and deviations. The pricing problem will then create new columns where these stations have a greater chance of being visited. The pricing problem algorithm is summarized in Algorithm 2. The notation used is listed in Table 6.5.

Table 6.5: Notation for pricing problem

Parameters	
n^{pp}	Number of PP iterations
B^{pp}	Branching constant in the PP
p^{pp}	Probability constant
w^{pp}	Weight pricing problem score

```
1  F := list with new columns
2  Number of iterations = 0
3  while number of iterations < $n^{pp}$ do
4      Read results from MP
5      $S^{pp}$ := Set of stations not included in MP solution
6      for each station $i \in S^{pp}$ do
7          $s_i^{pp}$ := violations + deviation at station $i$
8          Add $w^{pp}s_i^{pp}$ to criticality score for station $i$
9      end
10     for each service vehicle do
11         F ← Generate new columns through Algorithm 1 with branching constant
             $B^{pp}$, include $w^{pp}s_i^{pp}$ with probability $p^{pp}$
12     end
13     Number of iterations ++
14     Execute MP
15 end
```

Algorithm 2: Pricing Problem algorithm

In Line 7, a *pricing problem score* S_i^{PP} is assigned to each station that is omitted in the optimal combination of columns determined by the MP. The score corresponds to the sum of violations and deviation at the station. In Line 8 the pricing problem score with its corresponding weight w^{pp} is added to the station's criticality score.

In Line 11, the generation of new columns is performed in the same manner as in the initialization, using branching algorithm 1. A separate branching constant $B^{pp} < B$ is defined, expressing how many branches to create. As before, the algorithm will pick the B^{pp} stations with the highest criticality score. The difference is that the criticality score now has a fifth element corresponding to the pricing problem score s_i^{pp}. The updated criticality score is now as expressed in Equation (6.32). However, each time the branching algorithm is calculating the criticality score for a station, the fifth term is only included $p^{pp} \cdot 100$ % of the times. This is to prevent the new columns from having routes containing only stations that were not visited before. We want the pricing problem to act as a neighbourhood search where a few substitutions of an already generated route are tested.

$$c_i = -w^t t_i + w^n n_i - w^d d_i + w^o o_i + w^{pp} s_i^{PP} \tag{6.32}$$

After new columns have been generated for all service vehicles, the MP is re-solved. To summarize, the aim is to generate new columns that include some of the stations that were precluded by the previous MP.

Chapter 7

Implementation

The three master problems presented in Chapter 6 are implemented in Mosel and solved with Xpress. The initialization heuristic and pricing problem have been programmed with Java in the IntelliJ programming development environment. The key input data and test instances used for solving the model are elaborated in Section 7.1 and 7.2, respectively. Our implementation is based on the BSS in Oslo, operated by UIP. UIP has provided us with all data necessary for implementation. Our interpretation and implementation of UIP's current rebalancing method is described in detailed in Section 7.3. Specifications of the computer and software used to solve the rebalancing problem are listed in Table 7.1.

Table 7.1: Details of computer, solver and programming environment used in the computational study

Processor	Intel Core i7-6700 CPU @ 3.40GHz
RAM	32 GB
Operating System	Windows 10 Education 64-bit
Xpress IVE Version	1.24.18 64-bit
Xpress Optimizer Version	31.01.09
Mosel Version	4.6.0
IntelliJ IDEA Version	2017.3

7.1 Key Input Data

There are several fixed input parameters used in our model. The values for each of these parameters are discussed in this section. Parameters not discussed in this chapter, e.g. the time horizon and the maximum number of possible visits to a station, are calibrated through testing in Chapter 9.

7.1.1 Service Vehicles and Bicycles

The number of service vehicles available is considered an input parameter. The mathematical formulation can handle any number of service vehicles and both a homogeneous and heterogeneous fleets. The BSS in Oslo utilizes up to five service vehicles today. Multiple service vehicles make the model more realistic, but also increases the problem complexity. The service vehicles used by UIP are homogeneous and have 23 slots available. Today, UIP has approximately 1,790 bicycles in the system, which correspond to half the number of locks.

7.1.2 Parking, Handling and Driving Time

We were able to observe UIP's daily rebalancing operations late November 2017. The motive was to gather insight on how they make rebalancing decisions today, and to estimate the parking and handling time used for loading bicycles to and from the stations. The main observations were: 1) the handling speed varies between the individual workers, 2) the total handling time depends on how the station is positioned relative to the road and how close you are able to park, and 3) the handling time is not necessarily proportional to the number of bicycles handled. To simplify the model, we assume a linear relationship between the handling time and the number of bicycles handled. Our analysis resulted in a unit handling time of 0.25 minutes per bicycle. The parking time is assumed to be fixed, with a duration of two minutes. As the stations are located all over Oslo city, it is important to consider the driving time between each pair of stations. These driving times are collected by Google Maps' open API using the station coordinates, and requesting the average driving time.

7.1.3 Customer Demand

The customer demand data for each station is provided by UIP. The data set is a compilation of the historical hourly demand for bicycles and locks on weekdays for each station between July 13th 2017 and September 13th 2017. As the historical data only contains information about the customers that actually picked up or delivered a bicycle, the number of violations is unknown. To reflect true demand, the historical data is extrapolated over these periods. For each station, we look at the mean net demand.

7.1.4 Initial State

The initial bicycle loads at the stations and service vehicles are determined randomly. In addition to this, each service vehicle's initial station and remaining time to this particular station is set. In practice, these values would be set to the actual values. When testing, these values are chosen randomly. If the results are simulated, the initial states are updated for each iteration according to the situation in the simulated system.

7.1.5 Optimal State

As mentioned earlier, each station has a defined optimal state for each time period. The optimal value for each station for each hour is provided to us by UIP. The optimal state at a station is defined to be the point where the probability of congestion equals the probability of starvation, and is expressed in Equation (7.1). A detailed elaboration can be found in Espegren and Kristianslund (2016). The equation uses the expected demand and its standard deviation for the next three hours. $L_i^{S,0}$ denotes the initial load at station i for a particular hour, μ_{D_i} denotes the expected net demand for the next three hours at station i, σ_{D_i} denotes the standard deviation at station i for the next three hours, and Q_i^s denotes the capacity at station i. The equation thus sets the optimal state at a station with positive net demand closer to its capacity, and stations with negative net demand closer to zero. However, because of uncertainty, a high standard deviation will shift the optimal state away from the capacity or zero, respectively. As an example, Table 7.2 shows the optimal states, expected net demand, and station capacities for two arbitrary stations with positive and negative net demand, respectively.

$$1-\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\frac{L_i^{S,0}-\mu_{D_i}}{\sigma_{D_i}}} e^{\frac{-t^2}{2}}\,dt=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\frac{L_i^{S,0}-Q_i^s-\mu_{D_i}}{\sigma_{D_i}}} e^{\frac{-t^2}{2}}\,dt \tag{7.1}$$

Table 7.2: Optimal states for two stations with different net demand pattern

Station ID and name	**Net demand 7 - 8 am**	**Violations type**	**Station capacity**	**Optimal state at 7am**
103 Spikersuppa Vest	58.3	Congestion	42	9
44 St. Hanshaugen	-39	Starvation	12	12

7.1.6 Weights in the Objective Function

In the objective function, there are three main weights: 1) the violation weight w^v, 2) the deviation weight w^d, and 3) the reward weight w^r. In addition, there are two weights within the reward function: 1) the weight w^D rewarding initiated trips to stations with high deviation, and 2) the weight w^T penalizing the driving time to the last station visit. In our project thesis (Gleditsch and Hagen, 2017), the optimal values for the weights were determined by simulating different combinations where all weights varied from 0.0 to 1.0, incremented by 0.1. We observed that the model performed best when the weights were set to the values summarized in Table 7.3. These values are used further in this thesis.

Table 7.3: Weights in the objective function

Weight	**Symbol**	**Value**
Violations	w^v	0.6
Deviations	w^d	0.3
Reward function	w^r	0.1
Deviation in reward function	w^D	0.6
Driving time in reward function	w^T	0.4

7.2 Test Instances

While calibrating parameters, four different test instances are used. Test instance 1, 2, 3 and 4 consists of eight, 50, 100, and 158 stations, respectively. To observe different demand patterns, each test is conducted with demand data from two different hours, at 07:00 and at 17:00. A combination of a test instance and an hour is referred to as a *instance/hour combination.* With four test instances and two different hours, there are eight instance/hour combinations. These test instances are used to calibrate the parameters to their best-performing values, and to compare the different versions of the master problem. The test instances are summarized in table 7.4.

Table 7.4: Test instances

Test instance	Nr. of stations	Notation
Test instance 1	8 stations	1_8
Test instance 2	50 stations	2_50
Test instance 3	100 stations	3_100
Test instance 4	158 stations	4_158

The time horizon $\overline{T}$, the number of service vehicles V, and the maximum number of visits $\overline{M}$ are presented as shown in (7.2). An x indicates that the parameter is the one currently tested. Test instance 2_50$(\overline{T} = 20, V = 2\overline{M} = 1)$ indicates that test instance 2 is used, with two service vehicles and with a 20 minute time horizon, where each station can have a maximum of one visit within the time horizon. Further, the solution method used is specified as in (7.3), as either *V1, V2, V3*, *Exact method*, *current strategy*, or *No rebalancing*. (7.3) also specifies the value of the branching constant B and the flexibility parameter F in version 2.

Whether clustering is used or not is specified as true/false as shown in (7.4). If clustering is used, the values indicating the high and low net demand limits, C^H and C^L, are also specified. Pricing problem parameters, if not false, are specified as in (**??**). Number of runs, the branching constant and the probability of adding a non-visited station in the pricing problem are specified. Each combination of parameter values is defined as a *configuration*.

Test instance # ($\overline{T}$ = Time horizon, V = No. of vehicles, $\overline{M}$ = Possible visits) (7.2)

Solution method # (B = Branching constant, F = Flexibility parameter) (7.3)

Clustering = True/False(C^H = high net demand, C^L = low net demand) (7.4)

Pricing Problem = True/False(n^{pp} = No. of runs, B^{pp} = Branching constant (7.5)
p^{pp} = probability constant)

When simulating, a simulation setup has to be set. This includes the start and stop time t for the simulator, the number of scenarios n^S, and the route re-generation point t^{route}. The route re-generation point is denoted *First* if the subproblem is re-solved at the first vehicle arrival, *Second* for second vehicle arrival and *Third* for third vehicle arrival. If the point is a number N, it means that t^{route} is at constant time intervals of N minutes. This is denoted as shown in (7.6). Simulation$(t = 7:00 - 11:00, n^S = 10, t^{route} = First)$ indicates that we simulate over a four hour time period from 6:00 to 10:00, with ten different customer arrivals scenarios, and the subproblem is re-solved when the first service vehicle arrives at a station.

Simulation (t = Start and stop time, n^S = No. of scenarios, t^{route} = Route re-generation point) (7.6)

7.3 UIP's Current Rebalancing Strategy

To be able to quantify how much our model improves the imbalance in the system compared to today's situation, an algorithm for the current rebalancing strategy used by UIP today is modeled. Note, however, that this is our interpretation and implementation of the current strategy, and it is not necessarily entirely accurate. In November 2017, we visited UIP in Oslo to observe their daily operation, and the model implemented is based on the experiences obtained from this trip.

Today, UIP has divided the city centre into 14 zones. Each service vehicle operator is responsible for two zones, one zone with mostly delivery stations and one zone with mostly pickup stations. The allocation of zones changes during the day as the demand pattern throughout the city evolves. An illustration of the zones in the time period between 08:00 and 11:00 is given in Figure 7.1. Pairs of zones are indicated by equal color coding.

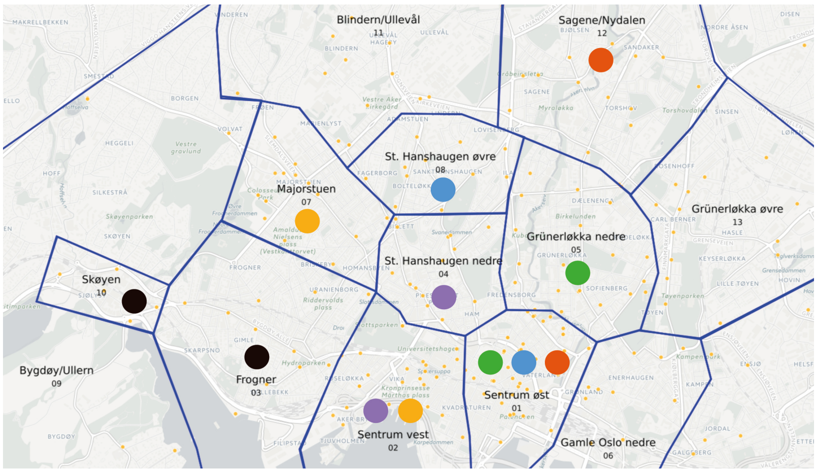


Figure 7.1: Illustration of the division of zones used by UIP in Oslo in the time period 8:00-11:00 when there are five service vehicles in operation. Zones with equal color coding are grouped together.

At the time of our visit, the service vehicle operators did not utilize any analytic program to determine where to drive or how much to load. Instead, they looked at an iPad that showed a live station map with updated information on how the bicycles were distributed around the city. The decisions regarding where to drive and how many bicycles to load were merely based on human experience and gut feeling. They conveyed that their biggest challenge is the massive demand for locks in the inner city center in the morning rush hours.

In our implementation of the current method, we assume that a service vehicle is already positioned at a station when routes are created. This station is referred to as the *current station*. We experienced that the loading quantities corresponded to either half the service vehicle's capacity or the entire vehicle's capacity. First, in our implementation of the current rebalancing method, the load at the current station is determined by the loading algorithm described in Section 6.3.5. The regret function in the loading algorithm is no longer used as we did not experience that they calculated how many bicycles to load at the next station when deciding load at the current station.

Further, the operators explained that they strive to balance several stations slightly, rather than balancing a few stations entirely. We observed that the operators visited at most two delivery stations or two pick up stations consecutively. We experienced that in half of the times, they visited two stations with the same demand type sequentially, and half of the times only one station of a certain demand type was visited before a station with the opposite demand type was visited. We got the feeling that this decision was made randomly and solely based on experience. However, whether a pickup station or delivery station was visited next depended on the service vehicle's load after the current visit.

In our implementation of the current rebalancing method, the rest of the geographical route is determined as illustrated in Figure 7.2. The service vehicle load after the current station visit is referred to as the *vehicle load*. First, if the service vehicle is almost empty, i.e. vehicle load < 5, the service vehicle visits pick up stations. Whether one or two pickup stations are added to the route, is drawn with a 50/50 probability. If the vehicle load is in the interval 5-18, either one pickup station or one delivery station is visited, drawn with a 50/50 probability. Last, if the service vehicle is almost full, i.e. vehicle load > 18, one or two delivery stations are added to the route, drawn with a 50/50 probability.

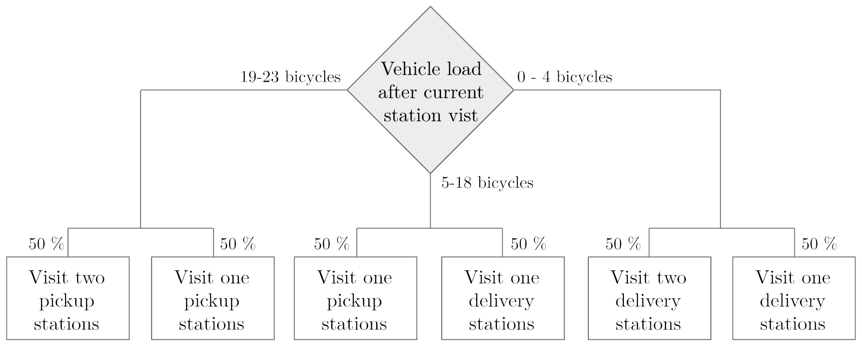

Vehicle load after current station vist
19-23 bicycles
0 - 4 bicycles
5-18 bicycles
50 %
50 %
50 %
50 %
50 %
50 %
Visit two pickup stations
Visit one pickup stations
Visit one pickup stations
Visit one delivery stations
Visit two delivery stations
Visit one delivery stations

Figure 7.2: The algorithm used for creating routes according to UIP's current rebalancing strategy

The next step of the algorithm is to decide exactly which stations to visit. A criticality score, as described in Section 6.3.4, is calculated for each station. The service vehicle operators did not have access to information regarding expected future demand, nor the optimal distribution of bicycles. Further, we experienced that the operators did not consider the driving time when deciding where to drive. However, they had a hypothesis of which stations that were to become full or empty in the near future, and which stations that were the most popular. Thus, the weights within the criticality score are set as shown in Table 7.5. In real-life, operators do not calculate these scores, and we, therefore, assume that they do not always pick the most critical station. Thus, when the next station visit is to be determined, our implementation of the current rebalancing method chooses a station randomly among the five stations with the highest criticality score. Mark that the chosen station must be in the service vehicle's zone.

Table 7.5: Criticality score weights in current rebalancing method

Weight	**Value**
Time to violation, w^v	0.7
Net demand, w^n	0.3
Driving time, w^k	0
Deviation, w^o	0

Chapter 8

Simulation Framework

As the DDBRS, hereby referred to as the subproblem, assumes known demand, real-world uncertainties are not taken into account. Hence, a discrete-event simulation framework is developed to test how the subproblem performs in a realistic setting. In this chapter, we describe in detail how the simulator works. First, a general overview, describing the main idea behind the simulator, is presented in Section 8.1. Second, in Section 8.2, the part of the simulator that generates different scenarios are outlined. In Section 8.3, the iterative process between the simulator and the subproblem is described. This is the process where actual violations are counted. A flow diagram, illustrating the simulation algorithm, is presented. Lastly, in Section 8.4, an example is presented.

8.1 General Overview

Based on probability distributions derived from historical customer demand data, the simulator draws different customer arrivals scenarios. A *customer arrival* is defined as one customer arriving at a station at a specific time with a request for a bicycle or a lock. A *customer arrivals scenario* is defined as one possible outcome of customer arrivals within a certain time period. This means that a customer arrivals scenario contains information about all the customer arrivals within a certain time period. The reason we consider individual customer arrivals and not the aggregated customer demand, is because we need to know the arrival time of each customer. The rebalancing strategies from the subproblem are evaluated by checking how many customer arrivals, in a given scenario, that are violated.

The simulator is programmed in Java. Figure 8.1 illustrates the iterative process between the simulator and the subproblem. The dashed border boxes indicate the information that is passed between the subproblem and the simulator. Recall that each combination of fixed input parameters in the subproblem is defined as one configuration. The main purpose of the simulator is to estimate how well the rebalancing strategies obtained from one configuration performs in a real-world situation, by simulation it over different customer arrivals scenarios.

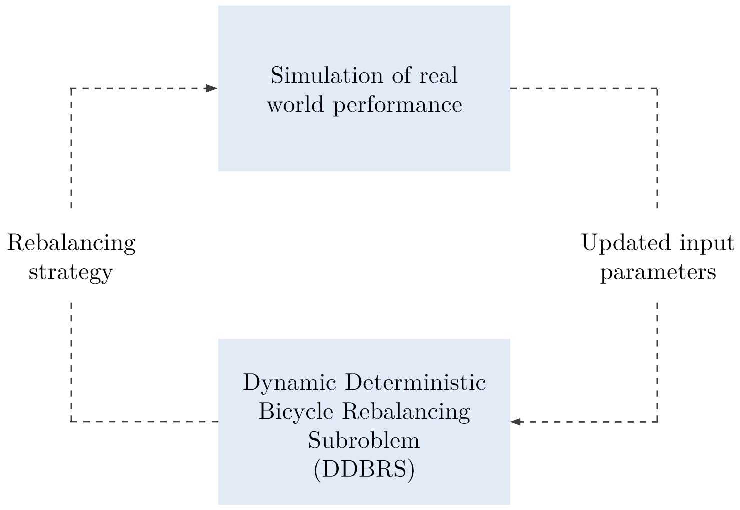

Simulation of real world performance
Rebalancing strategy
Updated input parameters
Dynamic Deterministic Bicycle Rebalancing Subroblem (DDBRS)

Figure 8.1: Iterative process between simulation framework and the DDBRS

When the simulation first starts at t^{start}, the subproblem is run with predefined inputs describing the initial state of the system. The output from the subproblem contains information about the service vehicles' rebalancing strategies, i.e. their routes, arrival times and the loading quantities. As mentioned, the subproblem is solved for a series of shorter time periods. From the output, the simulator determines the next *route re-generation point* t^{route}. This is defined as the time when new routes are generated, i.e. the time the subproblem is re-solved. Until this point, the service vehicles follow the rebalancing strategy suggested by the current subproblem.

The route re-generation point t^{route} is often shorter than the time horizon $\overline{T}$, meaning that the subproblem is re-solved before the end of the time horizon, e.g. every time a service vehicle arrives at a station or after a given time interval. The reason for this is that new exogenous information is revealed, and the subproblem can take better decisions with new updated input parameters. Before routes are re-generated in the subproblem, the simulator goes through the generated customer arrivals in this period, and determines whether each customer experience a violation or not. When the route re-generation point is reached, updated loads, customer demand, and positions are passed from the simulator to the subproblem, and the problem is solved again. A new rebalancing strategy is now provided.

The simulator keeps track of time by having a time variable called *current time*, CT. The current time is initially set to the chosen simulation start time t^{start} This time variable is event-based, meaning that it is updated every time an event occurs, i.e., every time a customer or a service vehicle arrives at a station, or every time routes are re-generated in the subproblem. When the current time equals the predefined simulation stop time t^{stop}, the simulation is complete. Figure 8.2, illustrates how the subproblem is re-solved through the simulation period.

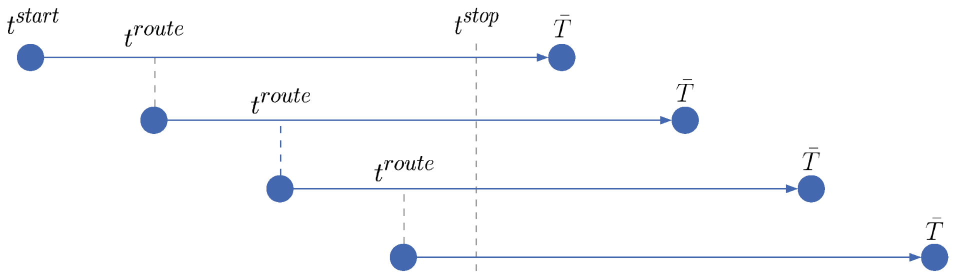


Figure 8.2: Illustration of how the subproblems are re-solved

When the simulation is completed, statistical performance measures are collected. When different configurations are tested over the same scenario, their performances are compared. Note that one scenario only is *one* possible outcome of customer arrivals. To make a reasonable conclusion when comparing different configurations, it is therefore essential that the configurations are tested over several different scenarios.

8.2 Generation of Customer Arrivals Scenarios

Based on historical customer demand data, the mean and standard deviation for the demand for both locks and bicycles are derived for each station and for each hour. The historical demand is assumed to follow a log-normal distribution, i.e., the customer demand for either locks or bicycles cannot be below zero. At times when stations have been empty or full, extrapolation is used to estimate the true demand. Based on this demand data, the simulator is able to draw random customer arrivals for each of the stations in the system. These customer arrivals together form a customer arrivals scenario.

To begin with, the total number of customers that demand a bicycle at each station within the first hour is drawn. This number is drawn from the log-normal distribution for the corresponding hour and station. Further, the arrival times for each of these customers are drawn individually from a uniform distribution, i.e., any time within that hour is equally likely. Similarly, the total number of customers demanding a lock and their arrival times are drawn, but with the probability distribution describing the demand for locks. This procedure is repeated for every hour within the simulation period. To illustrate the random scenario generation, an example is given. In the example, we simulate a customer arrivals scenario from 8:00 to 10:00 for a system with two stations. First, the total number of customers requesting a bicycle in the first hour is drawn. For this time period, the log-normal distribution for station 1 is shown in Figure 8.3. The mean is 5, and the standard deviation is 1. A random number from the distribution is drawn, 3.10 in this example, and is indicated by the red dashed line in the figure. This number is rounded to the closest integer, and the total number of customers requesting a bicycle at station 1 between 8:00-9:00 is set to 3.

Further, the arrival times are drawn individually and independently from a uniform distribution where the earliest possible time is 8:00, and the latest possible time is 9:00. In this example, the arrival times [8:48, 8:03, 8:23] are drawn. Figure 8.4 illustrates the uniform distribution, and the red dashed lines indicate the arrival times drawn by the simulator. The arrival times are saved in a list, illustrated in Table 8.1. Each row in the table represents one customer arrival, including the time, station id, and whether it is a request for bicycle or lock. When the customer requests a bicycle, the quantity is set to - 1, and when the customer requests a bicycle, the quantity is set to + 1.

Table 8.1: Customer arrival list containing the three first customer arrivals drawn by the simulator

Arrival time	**station id**	**Quantity requested**
8:23	1	- 1
8:03	1	- 1
8:48	1	- 1

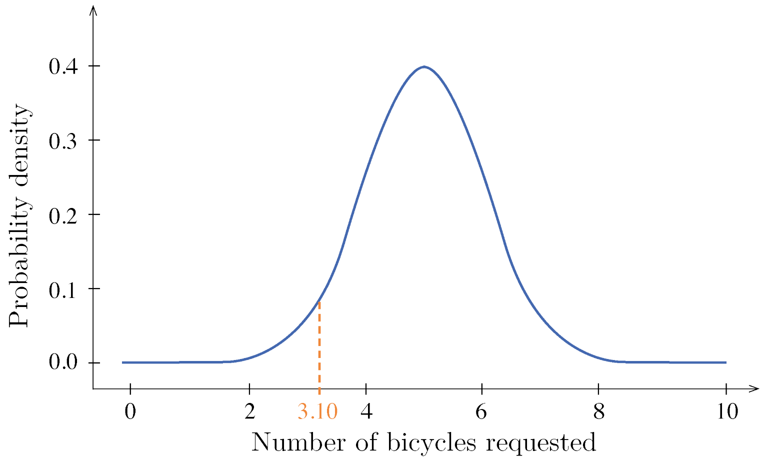


Figure 8.3: Log-normal distribution of the total number of customers requesting a bicycle in the time period 8:00-9:00 for station 1. The red dashed line indicates the random draw.

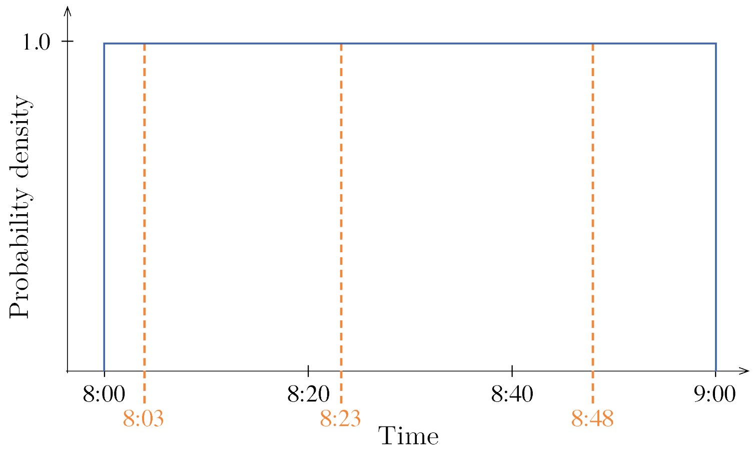


Figure 8.4: Uniform distribution of arrival times from 8:00 to 9:00 for station 1. The red dashed lines indicate the arrival times drawn.

Further, the arrival times for customers requesting a bicycle are simulated. The process is then repeated for the time period 9:00-10:00, with the corresponding probability distribution. All the arrival times are added to the list. The same procedure is carried out for station 2. Table 8.2 illustrates the entire list of customer arrivals between 8:00-10:00. This list is now one possible outcome of the customer arrivals in this time period for these stations, and is denoted a *scenario*.

Table 8.2: Customer arrivals scenario, containing arrival times for station 1 and 2 between 8:00-9:00, drawn by the simulator

Arrival time	**Station ID**	**Quantity requested**
8:03	1	-1
8:19	2	+1
8:23	1	-1
8:48	1	-1
9:11	2	+1
9:20	1	+1
9:31	1	+1

8.3 Simulation of Real-World Performance

To estimate how well a configuration performs, the results from the subproblems are simulated over different scenarios. The flow diagram in Figure 8.5 illustrates the simulation framework. The start and stop time for a scenario is denoted t^{start} and t^{stop}, respectively. Each box represents an action, whereas the arrows represent the order of the execution. In the rest of this section, each of the action boxes is described.

After reading the demand data for each station, the simulator generates a customer arrivals scenario. The current time CT is set to the simulation start time t^{start}. All parameters describing the system at CT is passed from the simulator to the subproblem. Next, the subproblem is solved, and the service vehicles' routes, loading quantities and arrival times are passed to the simulator. The first vehicle arrival is denoted t^{v}, and the first customer arrival is denoted t^{c}. The simulator has to determine the next route re-generation point, denoted t^{route}, This can for example be the next time a service vehicle arrives at a station. Further, the next upcoming event after CT is determined. The next event is the event with the time $min\{t^{stop}, t^{route}, t^{c}, t^{v}\}$. There are four possible first events; 1) the simulation is complete, 2) re-generation of routes in the subproblem, 3) a customer arrival or 4) a service vehicle arrival. Each of these possible first events is listed below with a description of how they are handled. If some of the *t* values have identical values, they are prioritized in the listed order. The simulator collects important statistics about the performance, e.g. the total number of starvations and congestions, and the number of customers.

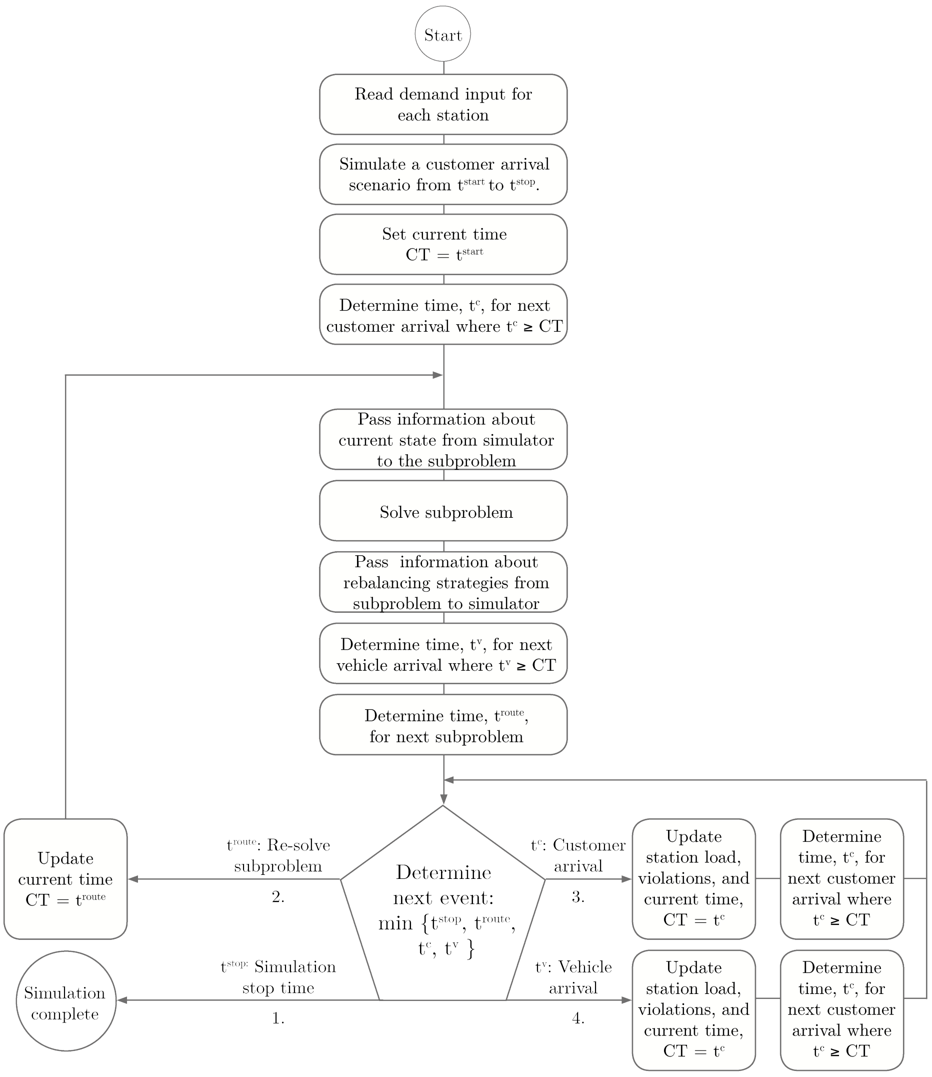

Start
Read demand input for each station
Simulate a customer arrival scenario from t^{start} to t^{stop}.
Set current time CT = t^{start}
Determine time, t^{c}, for next customer arrival where $t^{c} \geq$ CT
Pass information about current state from simulator to the subproblem
Solve subproblem
Pass information about rebalancing strategies from subproblem to simulator
Determine time, t^{v}, for next vehicle arrival where $t^{v} \geq$ CT
Determine time, t^{route}, for next subproblem
Determine next event: min $\{t^{stop}, t^{route}, t^{c}, t^{v}\}$
t^{route}: Re-solve subproblem
2.
Update current time CT = t^{route}
t^{stop}: Simulation stop time
1.
Simulation complete
t^{c}: Customer arrival
3.
Update station load, violations, and current time, CT = t^{c}
Determine time, t^{c}, for next customer arrival where $t^{c} \geq$ CT
t^{v}: Vehicle arrival
4.
Update station load, violations, and current time, CT = t^{c}
Determine time, t^{c}, for next customer arrival where $t^{c} \geq$ CT

Figure 8.5: Flow chart of the Simulation framework

1. t^{stop} is the first event: The simulator has reached its stop time. The total number of violations, and other statistic measures, are collected.

2. t^{route} is the first event: routes are re-generated in the subproblem. The current time is updated to $CT = t^{route}$, and the process loops back to where information about current state is passed to the subproblem.

3. t^c is the first event: A customer arrives at a station either requesting (a) a bicycle, or (b) a lock.

   (a) The customer requests a bicycle: If there are one or more bicycles at the station, the station load is updated to the current load minus one. If the station is empty, the count for number of starvations is incremented by one. The total number of customers is incremented by one either way.

   (b) The customer requests a lock: If there are one or more locks available, the station load is updated to the current load plus one. If the station is full, the count for number of congestions is incremented by one. The total number of customers is incremented by one either way.

   The current time CT is updated to be t^c, and the next customer arrival after CT is now denoted t^c. The process loops back to where the next event is determined.

4. t^v is the first event: A service vehicle arrivals at a station, wanting to either (a) load or (b) unload a certain quantity q of bicycles to or from the station. The current load at the station is denoted l^s, and the current load at the service vehicle is denoted l^v. The capacities at the station and on the service vehicle are denoted Q^s and Q^v, respectively.

   (a) The service vehicle wants to load bicycles to the station: If there are at least q available locks at the station, i.e., $Q^s - l^s \geq q$, and at least q bicycles on the service vehicle, the full amount of bicycles is loaded to the station. If there are not enough available locks, and/or not enough bicycles on the service vehicle, the load equals min $\{Q^s - l^s, l^v\}$. Station and service vehicle loads are updated accordingly.

   (b) The service vehicle wants to unload bicycles from the station. If there are at least q available spots on the service vehicle, i.e., $Q^v - l^v \geq q$, and there are at least q bicycles at the station, the full amount of bicycles are loaded from the station to the service vehicle. If there are not enough spots available on the service vehicles and/or not enough bicycles at the station, the load equals min $\{l^s, Q^v, l^v\}$. Station and service vehicle loads are updated accordingly.

The current time CT is updated to t^v, and the next service vehicle arrival after CT is now denoted t^v. The process loops back to where the next event is determined.

8.4 Example of a Simulation Process

This section illustrates the simulation process by an example. The simulation endures for two hours, and includes two stations. The simulation starts at time $t^{start} = $ 8:00, and stops at time $t^{stop} = $ 10:00. The stations' and the service vehicle's capacity is five bicycles. Routes are re-generated in the subproblem every time a service vehicle arrives at a station. Some events are highlighted and explained, while the entire collection of events are listed in Table 8.2. The rows in Table 8.2 show how the station and vehicle loads, the number of starvations and congestions, the total number of customers, and the number of times routes are re-generated in the subproblem, develop throughout the simulation. The columns show initial load and the times of the events. The vertical lines in the table, indicate route re-generation. The scenario generated is shown in Figure 8.6 a), and corresponds to Table 8.2 in Section 8.3.

Figure 8.6 illustrates the customer and vehicle arrivals on a timeline. The first row, marked as a), shows customer arrivals. Customer arrivals are denoted (Station ID, quantity requested). Row a), b) and c) illustrate the rebalancing strategies obtained the first, second and third time the subproblem is solved, respectively. Vehicle arrivals are denoted (Station ID, quantity loaded to station). The first route re-generation point is set to $t^{route} = $ 8:45. The first customer arrival takes place at $t_c = $ 8:03, and the first vehicle arrival takes place at $t_v = $ 8:00.

The next event after the current time $CT = $ 8:00, is the event happening at time min $\{t^{stop}, t^{route}, t_c, t_v\} = \{$10:00, 8:45, 8:03, 8:00$\} = $ 8:00, and is the vehicle arrival. The service vehicle wants to unload four bicycles from station 2. There are currently four bicycles at the station, and there are four available spots on the service vehicle, so all four bicycles are loaded from the station to the service vehicle. The loads are updated accordingly. The current time is updated to $CT = $ 8:00, and the next vehicle arrival time t_v is set to 8:45.

The next event is determined as min $\{t^{stop}, t^{route}, t_c, t_v\} = \{$10:00, 8:45, 8:03, 8:45$\} =$

8:03, and is a customer arrival. The customer is requesting one bicycle from station 1. As there are no bicycles available, one starvation violation is counted. The station load remains zero. One customer is counted.

The two next events are customer arrivals. The first event, at time 8:19, is a customer returning a bicycle to station 2, and the second event, at time 8:23, is a customer requesting a bicycle at station 1. Two more customer and one more starvation are counted.

When the simulator now determines the next event, $t^{route} = 8{:}45$ is the lowest value greater than CT, meaning that routes are re-generated in the subproblem. Information about current station and vehicle loads, as well as updated driving times, vehicle position, and demand information is passed to the subproblem. Routes are re-generated, and the count for subproblem runs is incremented by one. Figure 8.6 c) illustrates the rebalancing strategy from the subproblem after the route re-generation point. t^{route} is updated to 9:22.

The simulation keeps running until $t^{stop} = 10{:}00$. The remaining events are listed in Table 8.2. When the simulation is terminated, the total number of violations is summed, i.e. $1 + 2 = 3$.

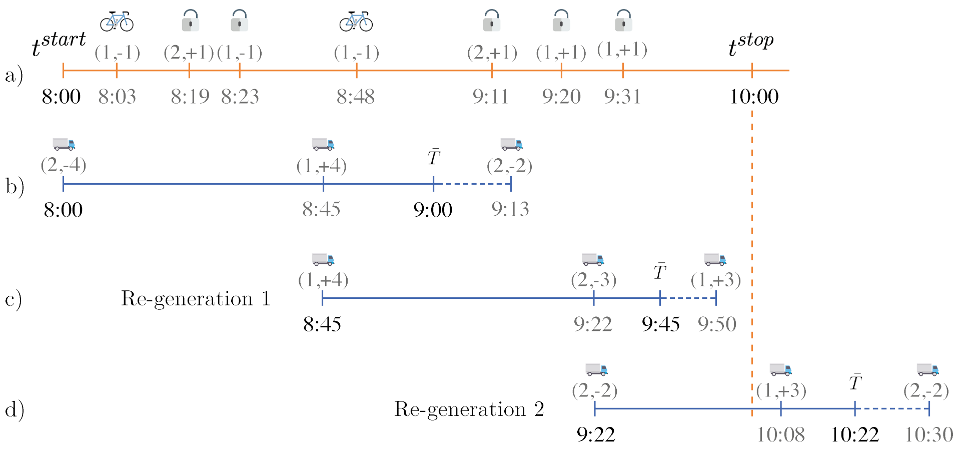


Figure 8.6: a) Customer arrivals in the time period 8:00-10:00, b), c) and d) rebalancing strategies from the first generation of routes, and the first and second route re-generation point.

Table 8.3: Example of simulation process from 8:00 to 9:00 with two stations and one service vehicle

	Initial load	Time for next event										Final result at 10:00
		8:00	**8:03**	**8:19**	**8:23**	**8:45**	**8:48**	**9:11**	**9:20**	**9:22**	**9:31**	
Station 1, l_{s1}	0		0		0	4	3		4		4	4
Station 2, l_{s2}	4	0		1				2		0		0
Service vehicle, l_v	0	4				0				2		2
Starvation count	0		1		2							2
Congestion count	0										1	1
Customer count	0		1	2	3		4	5	6		7	7
Number of subproblem runs	0	1				2				3		3

Chapter 9

Computational study: Parameter Tuning

In this chapter, parameter tuning is conducted for each of the three versions of the master problem. As the same parameters are determined for all three version, a description regarding how the different parameters are tested is presented in Section 9.1. The final configurations for version 1, 2 and 3 of the master problem are summarized in Section 9.2, 9.3, and 9.4, respectively. A detailed description of the parameter tuning is found in Appendix B. Lastly, in Section 9.5, a comparison between all three column generation versions and the exact solution method is conducted.

As the goal is to develop a model applicable for real-world operation, a fast computational time is important. Thus, we define *reasonable computational time* as ten seconds. The *computational time* is defined as the time it takes from the model starts generating initial columns until the master problem has allocated a route to each service vehicle. The reasonable time is set relatively low since the computational time varies due to randomness in the optimization solver and due to a varying number of station visits to determine.

9.1 Elaboration of Parameter Tuning

As the same parameters are tested for all three versions, a general description of how the parameters are tested is presented in this section. Version specific configurations are presented in their respective subsections.

9.1.1 Weights in the Criticality Score

First, the weights in the criticality score are determined. The importance of visiting a station i is expressed by the time before violations start occurring t_i, net demand per minute D_i, driving time T^D_{Ci}, and the deviation d_i. These terms are weighted with the coefficients w^t, w^n, w^k and w^o, respectively, and the total criticality score is given by Equation (9.1).

$$c_i = -w^t t_i + w^n D_i - w^k T^D_{Ci} + w^o d_i \tag{9.1}$$

To find the optimal values for the weights in the criticality score, different combinations of weight values are tested on each test instances. All the weights are varied from 0.0 to 1.0, incremented by 0.1. Also, the four weights must sum up to 1.0. This gives a total of 286 weight combinations. The goal is to make the branching algorithm pick the best stations first when adding new station visits to a route. This is to ensure that good routes are created despite a low branching constant. Thus, the branching constant is set relatively low in this test. As new branches in the initialization are created independently for each service vehicle, only two service vehicles are used in this test, i.e. $V = 2$.

Both the lowest and the highest objective values are presented for each instance. This is to observe the size of the gap, and thus, how much the results are affected by the values of the weights. Also, the percentage of the weight combinations that gives the lowest objective value is stated. If many different combinations result in the same objective value, it suggests that good routes are created regardless of the values of the weights. Moreover, trends within each test instance are observed, and a weight combination is set for each version. As real-world instances contain a lot of stations, results obtained from larger test instances are prioritized.

9.1.2 Branching Constant and Number of Possible Visits

The branching constant B indicates how many new branches the branching algorithm creates. Thus, a higher B means that more initial columns are generated. The number of possible visits $\overline{M}$ states the maximum number of visits each station is allowed to have. An increase in either B or $\overline{M}$ increases the solution space in the master problem, and an improved objective value is expected. However, the computational time is expected to increase drastically. Different combinations of these parameters are tested to evaluate the trade-off between computational time and the quality of the solution. While testing B and

$\overline{M}$, all other parameters are kept constant. As the number of service vehicles is one of the main drivers of the computational time, only two service vehicles are used in this test. $\overline{M}$ is only tested with the values 1 and 2, as a higher number is inappropriate with a short time horizon. The testing interval for B varies for each version. A conclusion is made, stating whether it is necessary with two possible visits, and the best-performing branching constant.

9.1.3 Parameter Tuning for Clustering Problem

By applying clusters, service vehicles are restricted to only visit stations that are in their respective clusters. As a result, the sets of columns have less overlap. Less overlapping columns might result in more feasible route combinations in the master problem, as each station can only have a maximum of $\overline{M}$ visits. Thus, better solutions are potentially found without having to increase the branching constant B. However, clustering can also prevent some good columns from being created as the service vehicles are restricted from visiting certain stations. Hence, the objective value can either increase or decrease when clustering is applied. Test instance 1 is not included in the clustering tests as we believe clustering on such small instances is unnecessary. The clusters are created in a separate model referred to as the *clustering model*. A set of parameters in the clustering model has to be set, and include the weights within the objective function, and the parameters that separate the stations' net demand into low, medium and high. We assume that the performance of these parameters is independent of which version of the master problem that is used as the computational time is unaffected by these parameters. As version 2 and 3 of the master problem are modifications of version 1, only version 1 is used in the testing of the clustering parameters. The results from these tests are used for all three versions.

9.1.4 Effect of Introducing Clustering

How clustering affects the objective function is observed for all three versions. Each instance is solved with and without clustering, and a conclusion regarding whether clustering should be applied or not is conducted by comparing the objective values.

9.1.5 Parameter Tuning for Pricing Problem

The impact of iterative adding of new columns, i.e. including the pricing problem, is tested. As the pricing problem adds new columns to the already initialized columns, the solution space increases, and the objective value can either remain unchanged or improve.

However, re-solving the master problem repeatedly increases the computational time, and we are again faced with a trade-off between the quality of the solution and the computational time. The parameters affecting the quality of the pricing problem are first tested. These parameters are the pricing problem weight within the criticality score w^{pp}, and the probability p^{pp} of including the pricing problem weight. We assume that the performance of these parameters is independent of which version of the master problem that is used as the computational time is not affected by these parameters. As version 2 and 3 are modifications of version 1, only version 1 is used in the testing of these parameters. The results from these tests are used for all three versions. As there is randomness involved in the pricing problem, all tests are run five times each. The results presented are the average values of the respective five runs.

9.1.6 Effect of Introducing Pricing Problem

How the pricing problem affects the objective value and the computational time is observed for all three versions of the master problem. The number of times the pricing problem is run n^{pp} and the branching constant B^{pp} in the pricing problem are first tested for each version. Including the pricing problem might result in an unreasonable computational time if the highest possible branching constant B within reasonable time, is used. Thus, the pricing problem is also tested with a lower B. By doing this, we can observe whether it is best to have a lower B value and include the pricing problem, or a higher B value and exclude the pricing problem. Different combinations of B, n^{pp} and B^{pp} are tested, and the values are determined based on the best result obtained within a reasonable time.

9.1.7 Number of Service Vehicles

Along with the branching constant and the number of possible visits, the computational time is expected to be extremely dependent on the number of service vehicles in the system. Thus, the changes in computational time are observed when the number of service vehicles increases. If the computational time increases considerably with an increasing number of service vehicles, it is desirable to see the impact of a dynamic branching constant, i.e. a branching constant that changes as a function of the number of service vehicles. The number of service vehicles V is varied from two to five. For each value of V, the highest possible B is observed, i.e. the highest B that keep the computational time reasonable.

9.2 Parameter Tuning for MP Version 1

In version 1 of the master problem, the loading quantities and the arrival times are determined in the master problem, along with the best combination of geographical routes. Detailed description of parameter tuning for version 1 is presented in Section B.1 in Appendix B. The final configuration used for version 1 is summarized in Table 9.1, and the configuration used for clustering and the pricing problem common for all versions is presented in Table 9.2.

Table 9.1: Configurations for MP version 1

Input parameter	**Symbol**	**Value**
Crit. score weight, time to violation	w^t	0.1
Crit. score weight, net demand	w^n	0.7
Crit. score weight, driving time	w^k	0.0
Crit. score weight, deviation	w^o	0.2
Number of possible visits	$\overline{M}$	1
Clustering		True
Pricing problem		False
Branching constant	B when $V = 2$	7
	B when $V = 3$	4
	B when $V = 4$	3
	B when $V = 5$	2

Table 9.2: Configurations for Clustering and Pricing Problem common for all versions

Weight	**Symbol**	**Value**
Clustering parameters		
Total driving time	w^K	0.5
Net demand	w^N	0.1
Size difference	w^Z	0.4
High net demand	C^H	70
Low net demand	C^L	30
Pricing problem parameters		
Pricing problem score weight	w^{pp}	4
Probability of including pricing problem score	p^{pp}	0.4

When tuning the parameters for version 1, several discoveries were made. For the criticality score, the model performs best when the time to violation weight w^t is low, the net demand weight w^n and the deviation weight w^o are high, and the driving time weight w^k is zero. The objective value improves when the branching constant and the number of possible visits increases as the search space in the master problem widens. Although a better solution is found when $\overline{M} = 2$ compared to when $\overline{M} = 1$ when the same branching constant is used, we can increase the branching constant when $\overline{M} = 1$ and find even better solutions. Hence, allowing multiple visits is concluded to reduce the quality of the solution.

Clustering is concluded to enhance the performance of version 1 as the objective value improves or remain unchanged. The pricing problem, on the other hand, is concluded to be counterproductive as the branching constant has to be decreased. The fact that a higher branching constant during the initialization of columns is better than including the pricing problem and using a lower branching constant demonstrates that the initialization algorithm performs well. The primary drivers of the computational time are the branching constant, number of possible visits and number of service vehicles. When introducing more service vehicles, the branching constant must be decreased.

9.3 Parameter Tuning for MP Version 2

In version 2, the loading quantities and arrival times are predetermined in the initialization, but the master problem has a given flexibility to adjust these values. To begin with, the flexibility parameter is set to its median value of 12 before it is tested and its best-performing value is obtained. Detailed description of parameter tuning for version 2 is presented in Section B.2 in Appendix B. Final configurations used for version 2 are summarized in Table 9.3.

When tuning the parameters for version 2, several discoveries were made. For the criticality score, the best solutions are found when the time to violation and driving time are weighted between 0.0 or 0.1. The distribution between net demand and deviation does not seem to be significant. Allowing multiple visits to a station results in a drastic increase in computational time and only a marginal improvement in solution. Hence, multiple visits is concluded to be unnecessary. A higher branching constant yields better solutions but increases computational time. The constant is set to its best-performing value obtained within a reasonable time. When more service vehicles are added, the branching constant must be further reduced.

Table 9.3: Configurations for MP version 2

Input parameter	**Symbol**	**Value**
Crit. score weight, time to violation	w^t	0.1
Crit. score weight, net demand	w^n	0.8
Crit. score weight, driving time	w^k	0.0
Crit. score weight, deviation	w^o	0.1
Number of possible visits	$\overline{M}$	1
Flexibility parameter	F	18
Clustering		True
Pricing problem		False
Branching constant	B when $V = 2$	7
	B when $V = 3$	5
	B when $V = 4$	3
	B when $V = 5$	2

The flexibility parameter F in version 2 allows for some degree of slack in the predetermined loading quantities and arrival times. However, there is a marginal difference between the results obtained with various flexibility parameters. All results produced when $F = 18$ are obtained within a reasonable time. Based on this, $F = 18$ is concluded as the best value. Clustering is concluded as enhancing for version 2, as the objective value is either improved or unchanged. The pricing problem, on the other hand, is concluded as counterproductive as the branching constant has to be decreased.

9.4 Parameter Tuning for MP Version 3

In version 3, the loading quantities and arrival times are entirely predetermined in the initialization. Oppose to the other versions, the stations are restricted to no more than one visit, i.e. $\overline{M} = 1$, and the optimal route combination chosen by the master problem consists of unique station visits. Hence, violations and deviations are also predetermined, and the mathematical model is significantly simplified. The computational time is expected to reduce considerably. Detailed description of parameter tuning for version 3 is presented in Section B.3 in Appendix B. Final configurations used for version 3 are summarized in Table 9.4.

Table 9.4: Configurations for MP version 3

Input parameter	**Symbol**	**Value**
Crit. score weight, time to violation	w^t	0.1
Crit. score weight, net demand	w^n	0.5
Crit. score weight, driving time	w^k	0.0
Crit. score weight, deviation	w^o	0.4
Number of possible visits	$\overline{M}$	1
Clustering		False
Pricing problem		True
Number of pricing problem iterations	n^{pp}	2
Branching constant in pricing problem	B^{pp}	15
Branching constant	B	20

When tuning the parameters for version 3, several discoveries are made. For the criticality score, the lowest objective value is found when the time to violation and driving time are weighted less, and the net demand and deviation are weighted highest. This strengthens the hypothesis stating that accounting for future demand is necessary. An increasing branching constant has minor effects on the objective value. This may indicate that the optimal solution is found. In version 3, the branching constant is set relatively high compared to the other versions as the computational time is drastically decreased. With $B = 20$, the branching algorithm branches to 20 pickup stations and 20 delivery stations from each node, which we assume are sufficient on these test instances. The branching constant does not need to be reduced when more service vehicles are added to the system.

Clustering is concluded as counterproductive as the objective value is worse in 33% of the instance/hour combinations. The reason for this may be that the branching constant is so high in version 3 that the master problem has no problem finding non-overlapping geographical routes, and applying clusters only prevents good columns from being created. As opposed to the other versions, including the pricing problem is not at the expense of the branching constant B, as the computational time is reasonable. The objective value is slightly improved, hence, including the pricing problem in version 3 is concluded to be appropriate.

9.5 Comparison of Column Generation Heuristics and Exact Solution Method

In this section, the different versions of the column generation heuristics are compared against each other. The heuristics are also compared to the results obtained when the model is solved with an exact solution method through a commercial solver. This is conducted to see how close the heuristic results are to the optimal values, and whether the heuristics are able to outperform the exact solution method when larger test instances are used. The comparison aims to establish a hypothesis of the best-performing version of the master problem.

The versions of the master problem and the exact solution method are compared for all eight instance/hour combination, and the objective values are presented. The exact solution method is only able to solve the smallest test instance 1_8 within a reasonable time. For the other test instances, we test whether a solution is found within 200 seconds, and list this as *(objective value found / gap between best solution found and upper bound)*. If no solution found within 200 seconds, it is listed as *No solution.* Note that this is not within our definition of reasonable time, but the values are listed for comparison purposes.

The comparison is conducted for two and five service vehicles to observe whether different versions should be used for different number of vehicles. Table 9.5 compares the objective values when there are two service vehicles in the system. The lowest objective value for each test instance is marked with green.

Version 1 and 2 are able to find the optimal solution obtained by the exact solution method for instance 1_8. As for version 3, the loading quantities are entirely determined by the heuristic, and the route chosen by the exact method is no longer optimal as the loading quantities are different. Hence, the objective value is slightly changed for version 3. For test instance 2_50, the exact solution method is only able to obtain a solution with an approximate 90% gap. The solutions found with the column generation heuristics are better, and the exact solution method is concluded as insufficient on the larger test instances.

Table 9.5: Comparison of column generation heuristics and exact solution method, $V = 2$

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)

		Exact	**V1**	**V2**	**V3**
Instance	**Hour**	Obj. value	Obj. value	Obj. value	Obj. value
1_8					
	07:00	55.67	55.67	55.67	56.34
	17:00	18.01	18.01	18.01	27.15
2_50					
	07:00	(220.76 / 86.14%)	213.31	213.31	214.74
	17:00	(169.95 / 89.09%)	158.81	158.81	167.83
3_100					
	07:00	No solution	389.36	389.36	391.03
	17:00	No solution	239.25	239.25	240.11
4_158					
	07:00	No solution	536.90	536.90	535.56
	17:00	No solution	362.77	362.77	369.39

As shown, version 1 and 2 performs best in all instance/hour combinations. Our hypothesis is that version 1 and 2 perform better for a smaller number of service vehicles, and that version 3 outperforms version 1 and 2 when more service vehicles are used. As the computational time is equal for version 1 and 2, and as version 1 produce at least as good solutions as version 2, version 1 is concluded as the best column generation heuristic when two service vehicles are in operation. Table 9.6 compares the objective values when five service vehicles are in operation.

With five service vehicles, the column generation heuristics are not able to find the optimal solution obtained by the exact solution method on instance 1_8. With five service vehicles, the branching constants in version 1 and 2 must be set relatively low to ensure that the problem can be solved within a reasonable time. However, a low branching constant is shown not to be sufficient as the optimal routes no longer are found. With the smallest test instance there are few non-overlapping columns, and we observe that most of the service vehicles only visit their predetermined initial station. For instance 3, the optimal route might be among the initial columns, but the master problem chooses differently due to other loading quantities. For the larger test instances, the exact solution method is not able to obtain a solution, and the column generation heuristics outperform the exact solution method.

Table 9.6: Comparison of column generation heuristics and exact solution method, $V = 5$

Test instance x ($\overline{T} = 20$, $V = 5$, $\overline{M} = 1$)

		Exact	**V1**	**V2**	**V3**
Instance	**Hour**	Obj. value	Obj. value	Obj. value	Obj. value
1_8					
	07:00	39.01	44.75	44.52	50.93
	17:00	11.68	12.35	12.35	20.66
2_50					
	07:00	No solution	195.97	196.50	193.38
	17:00	(153.13 / 85.48%)	141.98	142.0	144.41
3_100					
	07:00	No solution	372.14	372.14	370.97
	17:00	No solution	224.95	224.95	224.08
4_158					
	07:00	No solution	521.47	521.50	515.48
	17:00	No solution	341.61	342.22	345.45

Among the versions, version 3 yields the best results most frequently when more service vehicles are in operation. Note that the branching constants are different for each version of the master problem. If the branching constants were equal for all three version, version 1 would yield the best results. However, for version 3, we can use a higher branching constant, and thus generate more and better columns than in version 1 and 2. All things considered, version 3 is concluded to perform best when there are five service vehicles in operation.

Figure 9.1 illustrates the objective values obtained with the different versions of the master problem for a varying number of service vehicles for test instance 3_100 at 07:00. The figure illustrates that version 1 and 2 perform best when there are fewer service vehicles in the system, and that version 3 outperforms the other versions when there are more than three service vehicles in the system.

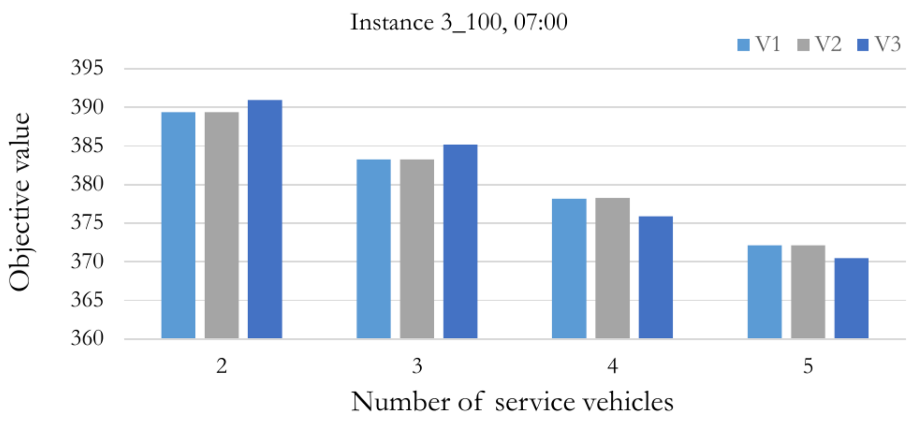


Figure 9.1: Comparison of the different column generation heuristics

Chapter 10

Computational Study: Simulation

In this chapter, the rebalancing strategies from the column generation heuristics are simulated with the framework described in Chapter 8. By simulating the results with different customer arrivals scenarios, we can observe a more realistic image of the performance. First, in Section 10.1, we discuss how the simulation results are evaluated. Second, in Section 10.2, simulations with the different column generation heuristics are conducted and the results are compared. One of the versions is selected to be the best-performing version, and final parameter tuning is conducted. At last, operational insights are presented in Section 10.3.

10.1 Evaluation Setup

In this section, we discuss how the results from simulation are evaluated. First, we describe which evaluation metric that is used. Second, we discuss how two different configurations are compared through a statistical t-test.

10.1.1 Evaluation Metric

As described in Chapter 8, the simulation framework generates random customer arrivals scenarios, where each customer arrivals scenario is defined as one possible outcome of customer arrivals within a certain time period. We will refer to this as a *scenario* further in the thesis. To evaluate the results, we look at the expected total violations, i.e. the average number of violations that occur in all scenarios. We refer to this as *total violations*. Recall that a violation is defined as an unsatisfied customer request. A *request* is defined as a customer wanting either a lock or a bicycle. Violations are further divided into starvations and congestions. A starvation occurs when a customer arrives at an empty station with intentions of getting a bicycle, and a congestion violation occurs when a customer arrives at a full station with intentions of delivering a bicycle. In the model developed in Chapter 5, these events are equally weighted. As it is desired to satisfy all customer requests, less total violations are preferred. The total violations is given in Equation (10.1).

$$\text{Total violations} = \frac{1}{\text{No. of scenarios}} \cdot \sum_{s \in \text{scenario}} \text{Total violations}_s \tag{10.1}$$

Different configurations of the heuristics are simulated from 7 am to 11 am over ten different scenarios. As mentioned earlier, we look at each station separately when customer arrivals are drawn. This will make the total number of bicycles in the system vary throughout the day. This variation is more prominent when the simulation is run over a longer period. To keep the number of bicycles in the system to an approximately constant level, a limited time period of four hours is chosen.

To increase the precision and reduce the variance of the estimate, each configuration is tested with ten different scenarios. Additionally, the variance is reduced by using common random numbers, i.e. the same random ten scenarios are used for each test. This results in a positive covariance between each observation, hence reducing the variance of the simulation results. As a point of reference, Table 10.1 presents the results obtained when no rebalancing takes place, i.e. the number of service vehicles in operation is zero.

Table 10.1: Results obtained with no rebalancing

Test instance 4_158, 07:00 **Solution method** No rebalancing **Simulation** ($t = 7:00 - 11:00$, $S = 10$)	
Number of customer requests	8475.5
Number of violations	1916.5
Number of congestions	570.2
Number of starvations	1346.3

10.1.2 Statistical T-test

The simulation results only provide an estimate as only ten different scenarios are simulated. To compare two different model configurations, a statistical *t-test* is conducted. Given a sample from a population, a t-test tests if the mean of the population could reasonably be a particular value (Walpole et al., 2007). We will use the t-test to check how likely it is that two configurations perform equally well. Our null hypothesis states that the two configurations tested perform equally well, whereas the alternative hypothesis states that one of the two configurations performs better than the other.

As we want to compare two different configurations, X and Y, we do a pairwise comparison of the results obtained from each of the ten scenarios. This means that for each of the ten scenarios, we set $Z_s = X_s - Y_s$, where X_s and Y_s are the total violations from the two different configurations with customer scenario s. Z_s is then the difference between the two results. We will now have ten Z-values. By conducting a t-test, we can express how likely it is, through a p-value, that the true difference Z is zero, meaning that the two configurations perform equally good. More precisely, if the null hypothesis is true, the p-value tells us the probability of observing a more extreme test statistic in the direction of the alternative hypothesis (The Pennsylvania State University, 2018). Conventionally, if $p < 0.05$ then the mean is unlikely to be zero whereas $p > 0.05$ provides no such evidence (Teetor, 2011).

The p-value is derived as shown in Equation (10.2). $\bar{z}$ is the mean of the sample, s_D is the standard deviation of the sample, and n is the number of elements in the sample. To calculate the p-value, we have used the statistical program R with the Z_s values as input.

$$p = P(|Z| > \frac{\bar{z} - 0}{s_D/\sqrt{n}}) \tag{10.2}$$

Table 10.2 illustrates an example where configuration X and Y are tested over five different scenarios. We want to check whether there are reasons to believe that one of the configurations performs better than the other. The Z_s values in the right column are passed to the statistical program R where the p-value is calculated. $p = P(|Z| > \frac{\bar{z}-0}{s_D/\sqrt{n}}) = 0.011$, meaning that it is only 1.1% probability that the two configurations perform equally good. As this value is lower than 0.05, we conclude that X performs better than Y.

Table 10.2: Example t-test calculation

	Total violations		
Scenario s	X_s	Y_s	$Z_s = X_s - Y_s$
1	15	17	-2
2	13	16	-3
3	16	19	-3
4	15	16	-1
5	14	15	-1

10.2 Parameter Tuning with Simulation

In this section, the rebalancing strategies obtained from the deterministic subproblems are simulated in a rolling horizon framework. The rebalancing strategies from version 1 and 3 are compared, and whether clustering and the pricing problem should be included is re-tested. Additionally, varying route re-generation points and time horizons are examined, and their best-performing values are obtained.

10.2.1 Comparison of Heuristics

Our hypothesis from the parameter tuning in Chapter 9 states that version 1 performs best when there are three or fewer service vehicles in the system, whereas version 3 performs best when there are more than three service vehicles in the system. To confirm or disprove this hypothesis, the results from both version 1 and version 3 are simulated with two and five service vehicles. The best results are marked with green.

Table 10.3: Comparison of heuristics

Test instance 4_158 ($\overline{T} = 20$, $V = x$)
Solution method x
Simulation ($t = 7:00 - 11:00$, $S = 10$, $t^{route} = F$)

	Version 1	**Version 3**
Solution method	Total violations	Total violations
V = 2	1562.4	🟩 1547.3
V = 5	1316.5	🟩 1129

As shown in Table 10.3, version 3 yield fewer total violations when two service vehicles are used. This invalidates the hypothesis that version 1 performs best when there are few service vehicles in operation. As for five service vehicles, version 3 performs the best, and the hypothesis is confirmed. By conducting a t-test where the results obtained with two service vehicles, and then the results obtained with five service vehicles are compared, we get p-values of 0.035 and $2 \cdot 10^{-6}$, respectively. Thus, we can statistically confirm that version 3 performs better regardless of the number of service vehicles.

As shown in Section 9.5, the objective value obtained from solving a subproblem with version 1 is lower than when it is solved with version 3 when two service vehicles are in operation. However, the simulated results state that version 3 performs best. This highlights the fact that the subproblem is just a proxy of the stochastic problem, and that the optimal solution in one subproblem might not be optimal in a dynamic setting.

By examining the rebalancing strategies generated with version 1 and version 3, we observe that within the four hour time period each service vehicles visits 22.1 and 25.5 stations, respectively, whereas the average loading quantities are 11.9 and 11.4, respectively. The average loading quantity is approximately the same regardless of which version that is used. However, it looks like version 1 generates better geographical routes as more station visits are completed within the time period. Thus, our hypothesis is that version 3 performs better because it the higher branching constant results in better geographical routes.

10.2.2 Clustering and Pricing Problem

In Chapter 9, clustering was concluded to be counterproductive, whereas the pricing problem was concluded to be enhancing for version 3. In this section, the effect of clustering and of including the pricing problem is re-tested by simulating the results in a dynamic setting. In addition, dynamic clustering is tested. Dynamic clustering means that new clusters are created each time new routes are generated, as opposed to regular clustering where the clusters are static and only created once. Table 10.4 compares the results when clustering, dynamic clustering, and the pricing problem are included and excluded. The best and worst result is marked with a green and red color, respectively.

We observe that regular clustering performs better when the pricing problem is excluded, while dynamic clustering performs better when the pricing problem is included. However, the changes are marginal, and presumably only affected by randomness in the pricing problem. Based on these results we are not able to conclude whether dynamic clustering is better than regular clustering. However, the difference between the results obtained with and without clustering is substantial, revealing that clustering is counterproductive. By conducting a t-test where the results obtained with and without clustering are compared, we confirm, with 99.9% confidence, that the model performs better without clustering. Our hypothesis is that if the branching constant in the initialization process was lower, clustering would improve the results. However, with a high branching constant, the model generates so many routes that clustering actually decreases the search space in the initialization process.

Table 10.4: Effect of introducing clustering and pricing problem

Test instance 4_158 ($\overline{T} = 20, V = 5$)
Solution method V3
Clustering = True / False($C^H = 70, C^L = 30$)
Pricing problem = True / False($n^{pp} = 2, B^{pp} = 15, p^{pp} = 0.4$)
Simulation ($t = 7:00 - 11:00$, $S = 10$, $t^{route} = F$)

	Clustering = False	**Clustering = True**	
		Dynamic = False	**Dynamic = True**
	Total violations	Total violations	Total violations
Pricing Problem = False	1131	1258.3	1269.1
Pricing Problem = True	1129	1285.9	1261.4

The fewest violations occur when clustering is excluded, and the pricing problem included. Note that this is the same conclusion as obtained in the parameter tuning chapter. Note that the model is not highly dependent on the pricing problem as the improvement is marginal. By conducting a t-test where the results obtained with and without the pricing problem are compared, we observe that there is an 87% probability that the two configurations perform equally well. Even though we do not have strong enough evidence to claim that the pricing problem improves the quality of the solutions, we choose to include it as the expected number of violations is lower, and as the computational time is reasonably low. Thus, the clustering and pricing problem configurations stay unchanged for further testing.

10.2.3 Examination of Adjustments in Service Vehicle Routes

The subproblem assumes deterministic demand. However, the real world demand is stochastic, and the demand patterns might deviate from its expected value. The *route re-generation point* t^{route} is defined as the time in which new service vehicle routes are generated by the algorithm. Until t^{route}, the service vehicles follow the rebalancing strategy suggested by the previous subproblem. At t^{route}, new exogenous information is revealed, and service vehicle routes are re-generated.

We want to examine the changes in the service vehicle routes when routes are re-generated. Figure 10.1, illustrates the routes generated for service vehicle 1 when test instance 4_158 at 07:00 is used with five service vehicles, and with a re-generation point corresponding to the first vehicle arrival. Each row represents one route. The circles indicate the stations and their respective ids, the number to the right of the circles indicate the loading quantities, and the numbers below indicate the arrival times.

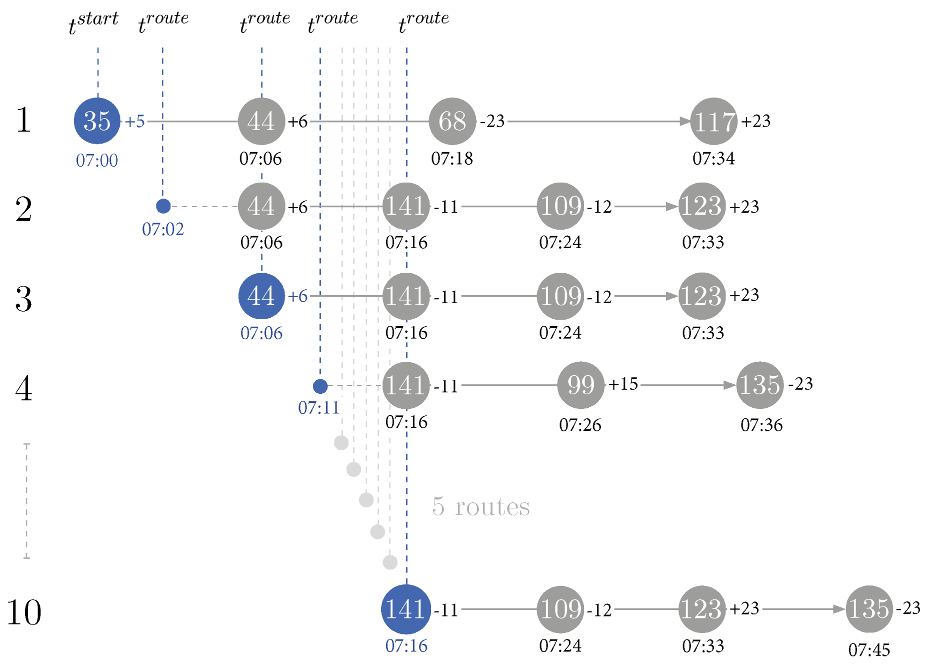


Figure 10.1: Routes generated for service vehicle 1: instance 4_158 at 07:00 with five service vehicles.

Row 1 indicates the first route generated for service vehicle 1. The service vehicle visits station 35 and loads five bicycles to the station. The next planned station visit is to station 44 at 7:10. However, another service vehicle arrives at another station at 7:02, and new routes are generated for all service vehicles. The new route for service vehicle 1 is illustrated in Row 2. Note that the first station visit is predetermined, and thus, not changed. The loading quantities, on the contrary, are not predetermined, but we observe that it remains unchanged at the first station. The rest of the route is changed, which implies that the actual customer demand might have differed from its expected value.

At 7:10, service vehicle 1 arrives at station 44, and new routes are generated. The loading quantity at station 44 is unchanged, and six bicycles are loaded to the station. Next station visit is planned to station 141 at 7:19. The routes generated at 7:02 (Row 2) and 7:10 (Row 3) are identical, indicating that actual customer demand probably was as expected. Between 7:10 and 7:19, six other service vehicle arrivals happen before service vehicle 1 arrives at station 141, and six new routes are generated for all service vehicles. These routes correspond to Row 5-9, hidden in the figure. At 7:19, service vehicle 1 arrives at station 141, and eleven bicycles are unloaded from the station.

This figure illustrates that the same stations often are picked when new routes are generated. Also, we observe that the loading quantity at the first station remains unchanged. Hence, re-generating routes every time a service vehicle arrives at a station may be unnecessary. This is further tested in Subsection 10.2.4. It also demonstrates that the branching algorithm works well, and that the routes in the master problem are picked wisely.

Figure 10.2 illustrates the routes generated for service vehicle 1. The routes shown are generated each time service vehicle 1 arrives at a new station, i.e. routes generated when other service vehicles arrive at a station are excluded. The time period is from 7:00 am to 08:00 am. Recall that the first station visit in each route is predetermined. These station visits are marked as blue. The key output variables from each subproblem are the loading quantity at the first predetermined station, and the second station visit, as these are the instructions that are realized. These output variables are marked with a red dotted box.

Even though only the first station visit is predetermined, we observe in this example that three out of the seven re-generated routes also keep the second station visit unchanged, i.e.

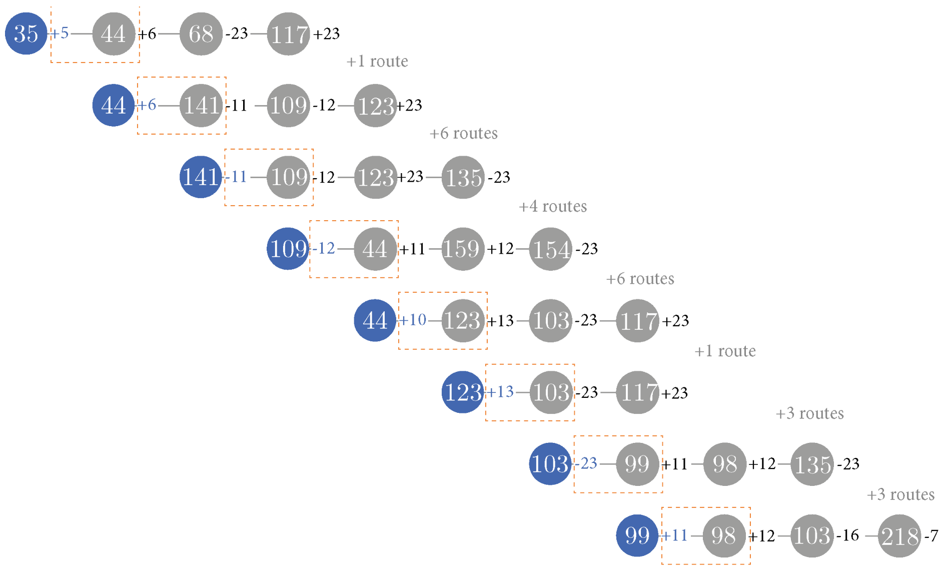


Figure 10.2: Station visits completed by service vehicle 1 between 7:00 and 8:00: instance 4_158 at 07:00 with five service vehicles and a time horizon of 20 minutes.

the third station visit from the previous route is not changed in the new route generated. In addition, the loading quantity at the first station remains unchanged six out of seven times, i.e. 86%. This further strengthens the hypothesis stating that it might be unnecessary to re-generate new routes every time a new service vehicle arrives at a station.

10.2.4 Route Re-generation Point and Time Horizon

The time horizon $\overline{T}$ is the length of the planning horizon within the subproblem, and is longer than the route re-generation point t^{route}. By generating new routes before $\overline{T}$, the system is allowed to adjust the already suggested routes. Table 10.5 summarizes the findings from running the simulator with three different route re-generation points t^{route}; 1) the first vehicle arrival, 2) the third vehicle arrival, and 3) with a constant time intervals of ten minutes. These are referred to as *First*, *Third* and *10 min* in the table. The length of the time horizon $\overline{T}$ is varied from 10 to 30 minutes, incremented by 10.

As the route re-generation point does not affect the computational time, the time is only listed once for each value of $\overline{T}$. Our evaluation of the best and worst result is marked with green and red, respectively.

Table 10.5: Route re-generation point and Time horizon

Test instance 4_158 ($\overline{T} = x, V = 5$)

Solution method V3

Simulation ($t = 7:00 - 11:00$, $S = 10$, $t^{route} = x$)

Time horizon $\overline{T}$	**Re-gen. point** t_{route}	**Total violations**	**Computational time** sec.
10 min			
	First	1261.0	
	Third	1299.3	0.28
	10 min	1370.2	
20 min			
	First	1129.0	
	Third	1154.6	3.0
	10 min	1225.2	
30 min			
	First	1104.2	
	Third	1096.3	11.1
	10 min	1122.0	

We recognize that the total number of violations tend to increase when the route re-generation point is delayed. This is reasonable as the opportunity to adjust the suggested rebalancing strategies is reduced. We also observe that the model tend to perform better with a longer time horizon $\overline{T}$. Although the optimal state and reward in the objective function account for future demand, it is challenging to bias the subproblem to make good long-term decisions. The amount of future demand considered increases with a longer time horizon. Hence, it is not surprising that the model performs better in a rolling horizon framework when a longer time horizon is used.

The best result obtained within a reasonable time is marked with green, and is obtained with a time horizon $\overline{T} = 20$ minutes, and with a route re-generation point t^{route} = First. It is reasonable to believe that a time horizon of 10 minutes is insufficient, especially when routes are re-generated less frequently. This is confirmed in the table, as this combination yields the worst result. As for $\overline{T} = 30$, the computational time is beyond what is defined as reasonable and is therefore concluded as impractical. Note, however, that a time horizon of 30 minutes yields the best results, and the trade-off between the computational time and the quality of the solution can be re-evaluated. The best result, marked with grey, regardless of computational time, is obtained with a time horizon $\overline{T} = 30$ and a route re-generation point t^{route} = Third, and yields a 3% reduction in total violations compared to when $\overline{T} = 20$, and t^{route} = First.

10.2.5 Summary of Parameter Tuning conducted with Simulation

By simulating the rebalancing strategies from version 1 and version 3 in a rolling horizon framework, we conclude that version 3 performs best regardless of the number of service vehicles in the system. This invalidates the hypothesis from Chapter **??**, which states that version 1 performs best when there are less than four service vehicles in the system. Further, we conclude that including the pricing problem in version 3 improves the results marginally, whereas clustering is shown to be counterproductive.

Table 10.6: Summary of Simulation

Parameter	**Value**
Best-performing version	Version 3
Clustering	False
Dynamic Clustering	False
Pricing Problem	True
Route Re-generation Point	First vehicle arrival
Time Horizon $\overline{T}$	20 minutes

By examining the adjustments in the rebalancing strategies made before and after a route re-generation point, we observe that the same stations often are picked and that the loading quantity at the first station visit almost always remains unchanged. This may indicate that re-generating routes at every vehicle arrival is redundant. However, varying route re-generation points and time horizons are tested, and a time horizon of 20 minutes, and a route re-generation point corresponding to every vehicle arrival is concluded to give the best results within reasonable computational time. The final configurations for the column generation heuristic are summarized in Appendix C.

10.3 Operational Insights

Our primary focus in this thesis is the rebalancing strategies on the operational level. However, it is interesting also to observe how the system is affected by different strategic decisions. Relevant decisions at the strategic level include how many service vehicles and bicycles to have in the system, the size of the service vehicles, how starvations and congestions are prioritized, and whether geo-fencing should be used. Note that the observations made in this section are merely intended as a basis for strategic decision-making for UIP. Our column generation heuristic is compared to the rebalancing method currently used by UIP to see if our heuristic potentially can improve the performance of the system. The current strategic decisions made by UIP are summarized in Table 10.7. These decisions are varied, and the effects are examined.

Table 10.7: UIP's current strategic decisions

Parameter	**Setting**
Number of service vehicles	5
Number of slots per vehicle	23
Number of stations	158
Number of locks	3580
Number of bicycles	1790
Geo-fencing	False

10.3.1 Comparison with Current Rebalancing Method

Rebalancing a BSS is a necessary operation to satisfy customer demand. However, rebalancing is costly as it requires operators and service vehicles. As of today, UIP does not utilize any analytic program to determine routes for the service vehicles. Each service vehicle is assigned two specific zones every morning. Today, they have 14 zones, where each service vehicle in operation is responsible for one zone with mostly delivery stations and one zone with mostly pickup stations. The rebalancing decisions are solely based on experience and gut feeling.

Table 10.8 compares the performance of our heuristic to UIP's current rebalancing method, and the percentage improvement is observed. Note that the current rebalancing method is based on our interpretation and implementation of how UIP conducts rebalancing today, and is not necessarily entirely accurate. The results are also compared to a situation with no rebalancing, i.e. no service vehicles are in operation. The total number of customer requests within the simulated time period is listed in the last table row as a point of reference.

Table 10.8: Column generation heuristic compared to current rebalancing method

Test instance 4_158 ($\overline{T} = 20$, $V = 5$)
Solution method V3 / Current / No rebalancing
Simulation ($t = 7:00 - 11:00$, $S = 10$, $t^{route} = F$)

	Total violations	**% Improvement compared to current method**
CG Heuristic	1129.0	31
Current method	1482.0	-
No rebalancing	1916.5	-
Total number of requests	8475.5	-

Compared to the current rebalancing method, our column generation heuristic reduces the total violations with 31%. This means that 353 more customer requests are expected to be satisfied over a period of four hours. By conducting a t-test, we can confirm with a confidence of $\sim$ 100%, that version 3 of the master problem performs better than the current rebalancing strategies. Hence, the generation of smart rebalancing strategies as opposed to taking decisions based on gut feeling is advantageous.

A reason for this substantial improvement may be that the current method utilizes zones, while testing conducted in this thesis has concluded that clustering is counterproductive. We believe that dividing the stations into zones is a reasonable choice when the decisions are taken solely by the operators as this narrows down the options each service vehicle has, and decisions are easier made. However, with data-driven decisions from our column generation heuristic, clusters only reduce the search space, and good routes are omitted.

10.3.2 The Value of Service Vehicles

Increasing the number of service vehicles will result in a higher level of satisfied customer requests. However, managing the service vehicles is costly for UIP, and a trade-off between the cost of operating additional service vehicles and the gain of satisfying more customer requests should be considered.

Optimal Number of Service Vehicles

The marginal effect of one additional service vehicle is observed by simulating with a varying number of service vehicles. As UIP should do assessments regarding costs, the intention of this analysis is not to conclude with an optimal number of service vehicles, but rather present data as a basis for decision-making. Table 10.9 presents the total violations and the marginal reduction in violations when one service vehicle is added.

Table 10.9: Value of Service Vehicles

Test instance 4_158 ($\overline{T} = 20, V = x$)

Solution method V3

Simulation ($t = 7:00 - 11:00$, $S = 10$, $t^{route} = F$)

No. of vehicles	Total violations	Marginal violation reduction
V = 2	1547.3	-
V = 3	1400.4	146.9
V = 4	1240.7	159.7
V = 5	1129.0	111.7
V = 6	1018.3	110.7
V = 7	951.9	66.4
V = 8	855.2	96.7
V = 12	682.5	43.2

Additional service vehicles should be added as long as the marginal cost of operating one more service vehicle is less than the gain of satisfying the marginal improvement in satisfied customer request. As expected, the total number of violations decreases when the number of service vehicles increases. This is reasonable, as the service vehicles strive to improve the balance in the system. More interesting is the observation of a diminishing marginal reduction of violations. As the most critical stations are always prioritized, a higher number of service vehicles allow visits also to less critical stations, and the marginal reduction of violations, therefore, decreases with more service vehicles. Figure 10.3 illustrates the trend of a decreasing marginal reduction in total violations as the number of service vehicles increases.

One might imagine that 12 service vehicles would yield a lot fewer than 607.9 violations as 368 bicycles can be transported simultaneously. The reason for this may be that the initial bicycle load at the stations is randomly distributed when simulating, as opposed to in real-life where the initial load at the stations depend on the rebalancing job already conducted. This means that if 12 service vehicles are in operation, the initial load is most likely a lot closer to the stations' optimal states than when the loads are picked randomly.

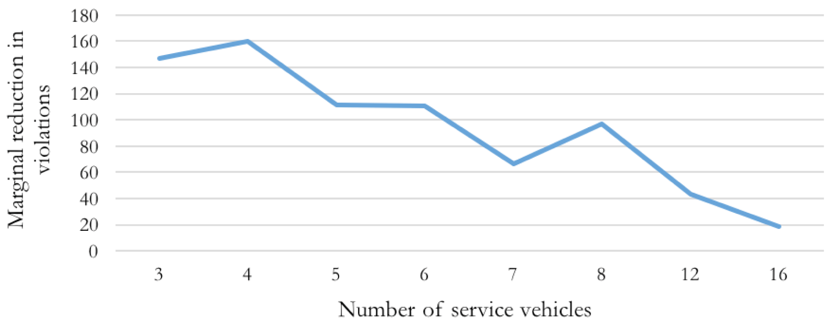


Figure 10.3: Marginal reduction in violations as a function of number of service vehicles

Varying Number of Service Vehicles with Time of Day

Further, we hypothesize that the number of service vehicles should vary throughout the day. In addition to minimizing the total violations, we assume that UIP strives to achieve a constant level of violations throughout the day to increase predictability for the customers. By adjusting the number of service vehicles, UIP can control the number of violations. Also, if labor is a scarce resource, UIP should prioritize to utilize the service vehicles when they are most needed. Figure 10.4 illustrates the expected demand for bicycles for each hour during the day. We see that the demand increases from 7:00-13:00, and then decreases again. Thus, the demand is highest between 12:00 and 15:00.

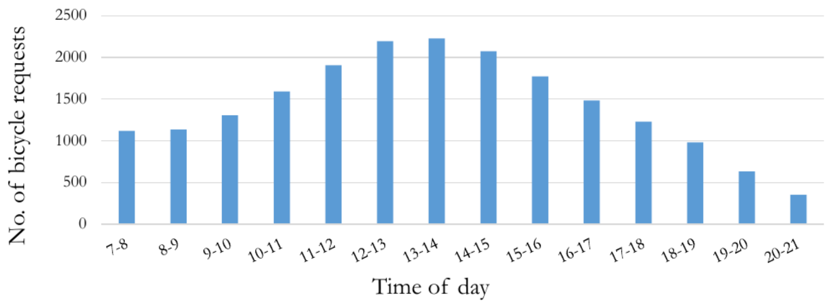


Figure 10.4: Requests for bicycles at different hours in the day

Table 10.10 summarizes the results when the number of service vehicles is varied from two to eight, and the simulation start time is varied between 8:00 and 17:00, incremented by three hours. Note that the simulation duration is adjusted to one hour, so the number of total violations is substantially decreased compared to a four hour simulation period.

Table 10.10: A varying number of service vehicles for different times of the day

Test instance 4_158 ($\overline{T} = 20$, $V = x$)
Solution method V3
Simulation ($t = x$, $S = 10$, $t^{route} = F$)

No. of vehicles	**t = 08:00-09:00**	**t = 11:00-12:00**	**t = 14:00-15:00**	**t = 17:00-18:00**
	Total violations			
V = 2	210.9	330.6	437.5	178.1
V = 3	203.8	318.7	420.1	171.6
V = 4	185.1	308.2	413.2	158.5
V = 5	171.8	297.5	399.5	149.1
V = 6	164.0	287.5	383.6	154.1
V = 7	158.1	283.4	372.8	154.2
V = 8	141.4	265.4	355.0	145.3

As expected, the number of total violations decreases when the number of service vehicles increases. We observe that the number of violations correlates with the number of customers. As the demand varies throughout the day, increasing the number of service vehicles in the period with the highest customer demand seems reasonable.

Effect of introducing larger service vehicles

Further, the effect of replacing the standard-size service vehicles with larger service vehicles is assessed. We assume that a large service vehicle has a capacity of 60 bicycles, as opposed to the standard-size service vehicle with 23 slots. Moreover, we assume that the fixed parking time and the unit handling time are increased from 2 to 6 minutes and 0.25 to 0.4 minutes per bicycles, respectively, when the large service vehicles are used. This is based on conversations with the service vehicle operators in UIP. The number of large service vehicles is varied from 2 to 5, and Table 10.11 outlines the results obtained.

Table 10.11: Value of one large service vehicle

Test instance 4_158 ($\overline{T} = 20, V = x$)

Solution method V3

Simulation ($t = 7:00 - 11:00$, $S = 10$, $t^{route} = F$)

Type	**No. of service vehicles**	**Capacity per vehicle**	**Total capacity**	**Total violations**
Standard				
	5	23	115	1129.0
Large				
	2	60	120	1686.7
	3	60	180	1575.4
	4	60	240	1495.5
	5	60	300	1428.9
	9	60	540	1114.5

Replacing five standard-size service vehicles with two large service vehicles yields the same total service vehicle capacity, but shows not to be efficient in terms of avoiding violations. On the contrary, operating two large service vehicles may be cheaper than operating five standard-size service vehicles, assuming only one operator at each service vehicle. Five service vehicles have to be replaced by a minimum of nine large service vehicles to satisfy an approximately equal amount of customer requests. Hence, despite the extended service vehicle capacity, the increased handling and parking time make the large service vehicles less efficient. Note, however, that the branching algorithm in the initialization is unchanged when the model is solved with the large service vehicles. Thus, the assumptions made regarding route generation and loading quantities may no longer be suitable.

10.3.3 Different Prioritization of Starvations and Congestions

In this thesis, we have assumed equal prioritization of avoiding starvations and congestions. However, a congested station may result in customers leaving their rented bicycles unlocked, which is very costly for UIP if stolen. Also, UIP suspects a greater amount of frustrations from customers that experience congestions than from customers experiencing starvations. Hence, UIP has expressed that avoiding congestions is prioritized. In the objective function in the subproblem, total violations are weighted with the weight w^v. The objective function in the subproblem is now re-formulated so that the total amount of congestions and starvations are weighed separately with the weights w^C and w^S, respectively. As discussen in Section 7.1.6, the optimal value of w^v is set to 0.6. In the re-formulated objective function, this is equivalent to setting both w^C and w^S to 0.6. Different priorities of starvations and congestions are examined in Table 10.12.

Table 10.12: Different weighting of starvations and congestions

Test instance 4_158 ($\overline{T} = 20, V = 5$)

Solution method V3

Simulation ($t = 7:00 - 11:00$, $S = 10$, $t^{route} = F$)

w^C	w^S	Total violations	Congestions	Starvations
0.6	0.6	1129.0	200.4	928.6
0.7	0.5	1125.4	199.6	925.8
0.8	0.4	1123.5	192.4	931.1
0.9	0.3	1133.4	191.4	942.0
1.0	0.2	1162.9	193.5	969.4

As shown in Table 10.12, different weightings of starvations and congestions have minor implications on the number of congestions. However, the number of starvations and the total number of violations, tend to increase when the congestions are weighted higher. Thus, weighting the congestions higher with the aim of reducing the congestions, is concluded to be counterproductive. Consequently, equal prioritization is maintained. An interesting observation is, however, that 82% of the total violations is due to starvations, i.e. no available bicycles, when congestions and starvations are equally weighted. We believe that the reason for this is that UIP intentionally has set the number of bicycles in the system low enough to achieves this distribution.

10.3.4 The Value of Bicycles

The impact caused by varying the number of bicycles is examined. Figure 10.5 presents the results obtained from simulation when the total number of bicycles is varied between 1000 and 3500, incremented by 500. Today, they have 1790 bicycles, and a total of 3580 locks in the system, i.e. the total number of bicycles is approximately half the number of locks. The results obtained with 1790 is marked with a red dotted line. Note that as the number of bicycles increases, the total number of locks remains the same, and the bicycle/lock ratio increases.

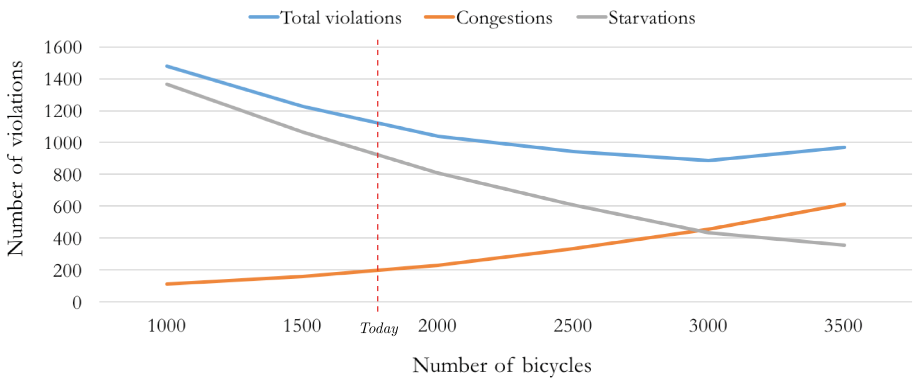


Figure 10.5: Violations as a function of number of bicycles in the system

The current number of bicycles, i.e. 1790 bicycles, does not yield the lowest total violations, but is most likely chosen to reduce the number of congestions, and thus, the number of possibly stolen bicycles. Figure 10.5 shows that an increasing amount of bicycles decreases the number starvations while increasing the number of congestions. Further, we observe that the total number of violations first decreases, and then increases when the total number of bicycles increases. The minimum number of violations is obtained with approximately 3000 bicycles in the system, i.e. a bicycle/lock ratio of 0.84. An interesting discovery is that the total violations are at its minimum when the total starvations equal the total congestions.

By increasing the number of bicycles in the system from 1790 to 3000, the total number of violations is reduced by 21%. This is at the expense of 254 additional congestions. This is equivalent to a 126% increase in the number of congestions.

10.3.5 The Value of Geo-fencing

With emerging technology, concepts like geo-fencing are enabled. Geo-fencing allows a station to be overflowed by bicycles beyond its fixed capacity. A geo-fenced area is an area around a station in which bicycles can be locked. As more bicycles now can be parked, geo-fencing can mitigate the issue of congestion at stations, i.e. when there are no available locks. Note, that the geo-fencing is not a replacement of the physical stations. Starved stations, i.e. no available bicycles, are not affected by geo-fencing.

Increasing Station Capacity with Geo-fencing

It is reasonable to set an upper limit on the total capacity within the geo-fence. Figure 10.6 presents the results when the effect of geo-fencing is simulated by increasing each stations' capacity by multiplying their current capacities with a multiplier of 1, 1.5, 2, 3, and 4.

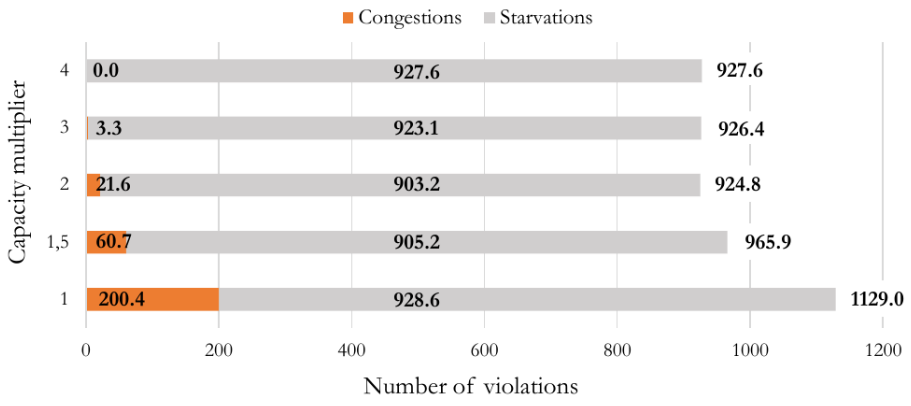


Figure 10.6: The Value of Geo-fencing and different capacity multipliers

As illustrated in Figure 10.6, the number of total violations decreases when geo-fencing is enabled. The minimum number of total violations is obtained when each stations' capacity is doubled, i.e. a station capacity multiplier of 2 is applied. By doubling the station capacities, an 18% reduction in total violations is obtained. Not surprisingly, the number of congestions converges to zero as the station capacities increase. The number of starvations first decreases, and then increase again.

Avoiding excessively loaded stations when geo-fencing is enabled is favorable as there are people and other surroundings to consider. It is impractical if hundreds of bicycles are parked at a station in the center of Oslo, as this may block traffic and prevent people from passing. Hence, the station with the highest load throughout the simulation period is observed to get an image of how geo-fencing would affect the city environment. When geo-fencing is enabled with an upper limit of 4x the station capacity, the most loaded station is station 126, Spikersuppa Vest, with 81 bicycles. This is in fact only 33 bicycles above the capacity. Whether 81 bicycles at one station is manageable, has to be assessed by UIP.

Optimal number of bicycles when geo-fencing is applied

In Subsection 10.3.4, an increasing number of bicycles in the system was tested. Increasing the number of bicycles led to an increasing amount of congestions as the total number of locks remained unchanged. However, increasing the number of bicycles may be convenient if geo-fencing is enabled. Figure 10.7 illustrates the results obtained when the total number of bicycles is varied from 1000 to 7000, incremented by 1000. The upper limit on the geo-fence is set to 2x the station capacity.

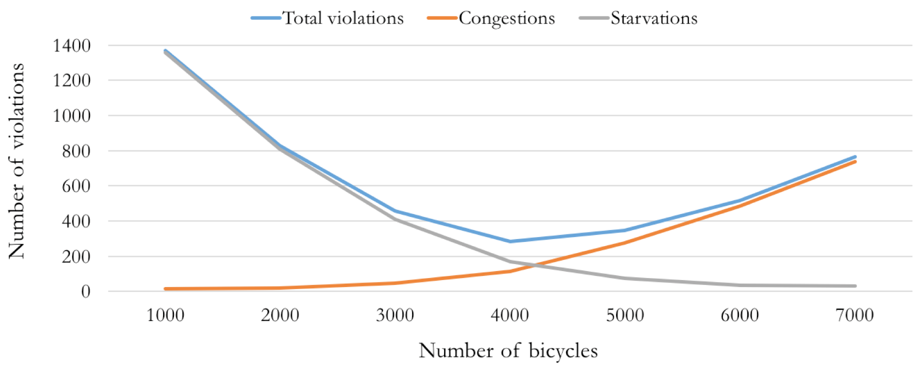


Figure 10.7: Violations as a function of number of bicycles in the system when geo-fencing is enabled

We observe from Figure 10.7 that with approximately 4000 bicycles in the system, the fewest number of violations are experienced. If congestions are prioritized, a lower total number of bicycles could be considered at the expense of more starvations. To conclude, enabling geo-fencing allows for more bicycles in the system, and the number of total violations is decreased to approximately 280. Compared to the current rebalancing strategy used by UIP today, this is equivalent to an 81% decrease in the total number of violations.

Similarly to the above analysis, the most loaded station is observed to get an image of how geo-fencing would affect the city environment. With 4000 bicycles, the station that experiences the highest load during the ten different scenarios, is station 126, Spikersuppa Vest, with 96 bicycles. Whether 96 bicycles is a reasonable amount of bicycles locked at a station has to be assessed by UIP. To summarize, increasing the number of bicycles when geo-fencing is enabled satisfies more customers, but goes at the expense of an excessive amount of bicycles at the most popular stations.

Effect of introducing large service vehicle when geo-fencing is applied

With geo-fencing and more bicycles in the system, it might be more appropriate to utilize large service vehicles. Table 10.13 presents the results when the number of large service vehicles is varied between 2 and 5, the geo-fence factor is set to 2x the capacity, and there are 4000 bicycles in the system. Note that the fixed parking time and the variable handling time are again increased to 6 minutes and 0.4 minutes per bicycle handled, respectively.

Table 10.13: Geo-fencing and Large Service Vehicles

Test instance 4_158 ($\overline{T} = 20$, $V = x$)
Solution method V3
Simulation ($t = 7:00 - 11:00$, $S = 10$, $t^{route} = F$)

No. of large service vehicles	Total violations	Congestions	Starvations
V = 2	728.2	303.0	425.2
V = 3	660.4	269.5	390.9
V = 4	584.8	239.1	345.7
V = 5	540.6	211.2	329.4

As observed in Table 10.13, the total violations occurring when there are five large service vehicles in the system is nowhere near the total violations when five standard size service vehicles are used. Hence, replacing the standard sized service vehicles with large service vehicles appear to be inefficient. This is reasonable as a large service vehicle uses more time on parking and handling bicycles than a standard size service vehicle and therefore reaches out to fewer stations.

10.3.6 Summary of Operational Insights

Through analyses conducted at the operational level, we learned that compared to UIP's current rebalancing method, our column generation heuristic decreases the number of total violations in the system with as much as 31%, leaving the number of service vehicles, the number of bicycles and other settings unchanged. This is a significant improvement which substantiates that rebalancing strategies should be data-driven as opposed to being based on gut-feeling and human experience.

Various factors regarding the BSS were analyzed, e.g. the number of service vehicles and bicycles in the system, the effect of large service vehicles, different prioritization of starvations and congestions, and the effect of allowing geo-fencing. As for the number of service vehicles, we observed that both the total violations that occur and the marginal reduction in violations, reduce when additional service vehicles are added. This means that with more service vehicles, the contribution of each service vehicle diminishes. The break-even point between the marginal costs of operating an additional service vehicle and the gain of satisfying additional customers has to be evaluated by UIP. Further, we observed that the number of service vehicles in operation should be increased mid-day when the demand is highest.

UIP has expressed that they often prioritize congested stations, i.e. full stations, as a stolen bicycle is costly. However, we observed that different prioritization of starvations and congestions in the objective function has approximately no effect on the distribution between starvations and congestions. On the contrary, we observed that adjusting the number of bicycles in the system has a major effect on the distribution between starvations and congestions. By increasing the number of bicycles, the number of starvations and congestions decreases and increases, respectively. To minimize the total number of violations, the number of bicycles should be increased from 1790, i.e. today's level, to approximately 3000 bicycles. This results in a 21% reduction in the number of violations. However, this increases the number of congestions, and the trade-off should be evaluated. Moreover, we observed that, with the number of bicycles that are in the system today, 82% of the violations stem from starvations.

Combining geo-fencing and more bicycles appear to be the best solution when the aim is to decrease violations. By increasing the number of bicycles to 4000, and by doubling each station's capacity through geofencing, the number of total violations reduces with approximately 75% and the total congestions reduces with 44%. Compared to UIP's current rebalancing method, this corresponds to an 81% reduction in total violations. The effect of replacing today's service vehicles with larger trucks was examined, and concluded to be inefficient regardless of geofencing and the number of bicycles.

Chapter 11

Concluding Remarks

In this chapter, the concluding remarks are drawn in Section 11.1, and future research opportunities are presented in Section 11.2.

11.1 Conclusion

In this thesis, we presented a column generation heuristic for solving the dynamic rebalancing problem for a Bike Sharing System (BSS). A BSS offers short-term bicycle rental to its users, but due to the one-directional flow of travel, the system becomes imbalanced and customers requesting the service are left unsatisfied. The BSS in Oslo, operated by Urban Infrastructure Partners (UIP), is used as a sample case in this thesis. They utilize service vehicles to restore balance. The key objective of the dynamic rebalancing problem is to generate efficient routes and loading quantities for the service vehicles with the aim of minimizing violated customer demand. The real-world problem is complex as it is both dynamic and stochastic. A simplification is made, and the real-world problem is approximated through a set of smaller subproblems with known customer demand, denoted as the Dynamic Deterministic Bicycle Rebalancing Subproblem (DDBRS).

A heuristic solution approach is necessary to solve the DDBRS of realistic size. As only a small portion of the variables in the DDBRS will take a value greater than zero, a column generation heuristic is proposed. The heuristic includes a heuristic initialization procedure, a master problem solved to optimality, and a heuristic pricing problem. The initialization process consists of a branching algorithm that generates a set of columns for each service vehicle. The master problem further determines the optimal combination of columns by allocating one column to each service vehicle. The mathematical formulation of the master problem differs from others in literature as we interpret a long-term focus by accounting for future demand. We propose three different versions of the master problem, where the amount of predetermined information in the initialized columns differs. In version 1, the loading quantities are determined in the master problem; in version 2, the loading quantities are predetermined in the initialization and re-evaluated in the master problem; and in version 3, the loading quantities are entirely predetermined. A heuristic pricing problem is developed with the goal of generating new and better columns.

The column generation heuristics is compared to each other and to an exact solution algorithm. For the large test instances, the exact solution method is unable to obtain a solution, and the column generation heuristics excel. Further, version 3 of the master problem, where loading quantities are predetermined, is shown to outperform the others as it is able to handle many service vehicles and a more extensive search space. We also observed that the optimal solution from one subproblem was not necessarily the best solution in a dynamic setting. This highlights the fact that the subproblem only is a proxy of the stochastic problem. The computational time when solving the DDBRS with 158 stations and five service vehicles is approximately 3 seconds. As the master problem is capable of handling a large subset of columns, a substantial number of columns are generated in the initialization process, and the pricing problem showed only to improve the results marginally. Additionally, clustering of stations is proposed, but is shown to be counterproductive when version 3 of our column generation heuristic is used.

As the DDBRS does not account for real-world uncertainties, the results are simulated using an implemented discrete-event simulator. A configuration is evaluated by observing the expected total violations obtained from ten randomly drawn demand scenarios generated based on historical data. After a certain time, new exogenous information is revealed, and routes are re-generated by the algorithm. By examining the routes re-generated for a service vehicle, it is observed that the branching algorithm performs well. Compared to UIP's current rebalancing method, our column generation heuristic reduces the expected total violations by 31%. Hence, the generation of smart and data-driven rebalancing strategies as opposed to making decisions based on gut feeling is advantageous.

The main findings from analyses conducted at a strategic level were that increasing the number of bicycles in the system combined with the concept of geo-fencing appear to be the best opportunity when the aim is to decrease violations. By increasing the number of bicycles to 4000, and by doubling each station's capacity through geo-fencing, the total violations reduces further with approximately 75% and the total congestions reduces with 44%. In addition, more service vehicles should be added as long as the marginal cost of operating one more service vehicle is less than the gain of satisfying the marginal improvement in satisfied customer requests. As expected, the total number of violations decreases when the number of service vehicles increases.

To conclude, solving the DDBRS with a column generation heuristic produces high-quality solutions within a reasonable computational time. Our heuristic increases the number of satisfied customer requests substantially, and can thus be a significant contribution to BSSs. In fact, by implementing our column generation heuristic, applying geo-fencing, and increasing the number of bicycles in the system, the total violations in Oslo can decrease with as much as 81% compared to today's current rebalancing strategies.

11.2 Further Research Opportunities

This section highlights research opportunities that can contribute to further improvements of the BSS. How the realism of customer interactions can be improved, further development of the pricing problem, the implementation of user incentivizing, and demand forecasting are discussed.

11.2.1 Improving Realism of Customer Interactions

The simulation framework developed in this thesis separates the generation of requests for bicycles and locks when generating demand scenarios. In the real-world, requests for bicycles and locks are dependent as a customer renting a bicycle also must return it. Also, if the customer experiences starvation, and chooses not to find a bicycle at another station, the customer does not have a bicycle to return. A future research opportunity is to develop a simulation framework that generates customer trips where the request for a lock is dependent on whether the customer gets a bicycle or not instead of customer requests. Implementing this will further improve the realism of the simulated results.

Further, one can assume that if a customer experiences a starvation/congestion, she will look for an available bicycle/lock at a nearby station, and the violation might be omitted. This assumption is probably rational as many stations are placed close to each other. This assumption can be incorporated in the simulation framework, but also in the initialization. Even though a station is starved or congested, it might not be a problem if another station nearby has available bicycles/locks, and the station can thus be given a lower criticality score.

11.2.2 Further Development of the Pricing Problem

The pricing problem developed in our column generation heuristic is based on simple mechanisms. We identify critical stations omitted in the chosen routes that may contribute to a better solution if included. The improvements obtained with this implementation of the pricing problem were marginal. Hence, a future research opportunity is to develop a smarter pricing problem algorithm.

11.2.3 User Incentivizing

An interesting topic regarding BSSs is to engage the users themselves to rebalance the system and equalize the one-directional flow of travels. Singla et al. (2015) discuss that utilizing service vehicles goes against the green concept of BSS, and present a mechanism that incentivizes the customers to pick up and return bicycles at preferred stations. In Paris, Velib' runs a static incentive schema that offers users 15 extra free minutes each time they return a bicycle to an elevated station. Fricker and Gast show that even simple incentives, such as suggesting that users return to the least loaded station among two stations, improve the imbalance by an exponential factor. There is much potential in intelligent and self-sustainable systems, and lots of interesting research topics can be related to this.

11.2.4 Demand Forecasting

The rebalancing strategies determined by our model are highly dependant on the net demand and optimal state derived from the demand data obtained from UIP. Hence, any miscalculations or incorrect assumptions leads to inefficient and unsuitable rebalancing strategies. Developing a smarter and complete demand forecast is therefore beneficial. The main challenge is to estimate demand in the periods when the stations are either full or empty, as true demand is not captured. An opportunity is to utilize smart technology and machine learning in demand forecasting. In addition, the demand forecast should consider weather, events and holidays that affect the demand patterns.

Appendix A

Linearization

A lot of the constraints from Section 5.4 are nonlinear. To solve a problem using linear solving tools, we need to make these constraints linear before we can implement them. Big-M formulations are used, where the Ms are set to values that are large enough to include all solutions at the same time as they are as tight as possible. Section A.1 presents linear versions of all the nonlinear constraints from Section 5.4.1. Section A.2 presents linear versions of all the nonlinear constraints from Section 5.4.2. And, lastly Section A.3 presents linear versions of the nonlinear constraints from Section 5.5.

A.1 Ensure Feasible Routes

Constraints (5.7), (5.12) - (5.14),and (5.16) - (5.18) are nonlinear and need to be linearized before implementation. Big-M formulations are used for linearization. The M values are specified so that they are of sufficient size, but also as tight as possible.

Constraints (5.7):

$$(l^V_{imv} + q^U_{jnv} - q^L_{jnv} - l^V_{jnv})x_{imjnv} = 0 \quad v \in V, i \in S\backslash\{d\}, j \in S, m,n \in M$$

Constraints (5.7) ensure that the service vehicle load is in balance before and after a visit. Big-M is set to the upper capacity inventory level for service vehicle v. The linear version is presented in Constraints (A.1) and (A.2).

$$l^V_{imv} + q^U_{jnv} - q^L_{jnv} - l^V_{jnv} - M(1 - x_{imjnv}) \leq 0 \quad v \in V, i \in S \backslash \{d\}, j \in S, m, n \in M \tag{A.1}$$

$$l^V_{imv} + q^U_{jnv} - q^L_{jnv} - l^V_{jnv} + M(1 - x_{imjnv}) \geq 0 \quad v \in V, i \in S \backslash \{d\}, j \in S, m, n \in M \tag{A.2}$$

where

$$M = max\{l^V_{imv} + q^U_{jnv} - q^L_{jnv} - l^V_{jnv}\} = Q^V_v$$

Constraints (5.12):

$$(l^S_{i,m-1} + \sum_{v \in V}(q^L_{i,m-1,v} - q^U_{i,m-1,v}) + D_i(t_{im} - t_{i,m-1}) + v^S_{im} - v^C_{im} - l^S_{im})$$
$$\sum_{j \in S}\sum_{n \in M}\sum_{v \in V} x_{imjnv} = 0 i \in S, m \in m \backslash \{1\}$$

Constraints (5.12) capture violations between each stations visit. The linear version is presented by Constraints (A.3) and (A.4). Equation (A.5) shows the terms that have to be maximized in the big M-formulation. Equation (A.6) states that the maximum quantity loaded to the station equals the minimum of the stations capacity and the service vehicle capacity. The last station visit within the time horizon can theoretically happen right before the time runs out, before it get the chance to unload or load any bikes. Since the station visit t_{im} can happen after the time horizon, $max\{t_{im}\}$ equals the length of the time horizon, plus the maximum driving time to the station and the maximum handling and parking time at the previous station. This is shown in Equation (A.7). The maximum violation is shown in Equation (A.8), and equals the demand times the maximum value for t_{im}. Equation (A.9) states the maximum load at the station, and equals the station capacity.

$$\begin{aligned} &l^S_{i,m-1} + \sum_v (q^L_{i,m-1,v} - q^U_{i,m-1,v}) + D_i(t_{im} - t_{i,m-1}) + v^S_{im} - v^C_{im} - l^S_{im} \\ &\quad + M(1 - \sum_{j \in S}\sum_{n \in M}\sum_{v \in V} x_{imjnv}) \geq 0 \qquad i \in S, m \in m \backslash \{1\} \end{aligned} \tag{A.3}$$

$$\begin{aligned} &l^S_{i,m-1} + \sum_v (q^L_{i,m-1,v} - q^U_{i,m-1,v}) + D_i(t_{im} - t_{i,m-1}) + v^S_{im} - v^C_{im} - l^S_{im} \\ &\quad - M(1 - \sum_{j \in S}\sum_{n \in M}\sum_{v \in V} x_{imjnv}) \leq 0 \qquad i \in S, m \in m \backslash \{1\} \end{aligned} \tag{A.4}$$

$$M = max\{q^U_{im} - D_i \cdot t_{im} + v^C_{im} + l^S_{im}\} \tag{A.5}$$

where

$$max\{q_{im}^{U}\} = min\{max_{v\in V} Q_v^V, Q_i^S\} \tag{A.6}$$

$$max\{t_{im}\} = \overline{T} + T^H(min\{max_{v\in V} Q_v^V, max_{k\in S} Q_k^S\}) + T^P + max_{k\in S} T_{ki}^D \tag{A.7}$$

$$max\{v_{im}^{C}\} = max\{D_i \cdot t_{im}\} \tag{A.8}$$

$$max\{l_{im}^{s}\} = Q_i^S \tag{A.9}$$

Constraints (5.13):

$$(t_{im} + T^H(q_{imv}^L + q_{imv}^U) + T^P + T_{ij}^D - t_{jn})x_{imjnv} \leq 0 \quad v \in V, i, j \in S\backslash\{d\}, m, n \in M$$

Constraints (5.13) restrict the time between station visits. Constraints (A.10) is a linear version of Constraints (5.13). Equation (A.11) shows the terms that have to be maximized in big-M. max t_{im} is as described above, and is shown is Equation (A.12). The maximum quantity that is unloaded or loaded at station visit (im) is shown in Equation (A.13). As shown in Equation (A.14), the driving time from station i to the next station j will be zero as the next station will be the artificial destination when t_{im} is maximized. (A.15) The maximum value of $-t_{jn}$ equals zero.

$$\begin{aligned} t_{im} + T^H \sum_{v\in V}(q_{imv}^L + q_{imv}^U) + T^P + T_{ij}^D - t_{jn} - M(1 - \sum_{v\in V} x_{imjnv}) \leq 0 \\ i \in S, j \in S\backslash\{d\}, m, n \in M \end{aligned} \tag{A.10}$$

$$M = max\{t_{im} + T^H \sum_{v\in V}(q_{imv}^L + q_{imv}^U) + T^P + T_{ij}^D - t_{jn}\} \tag{A.11}$$

where:

$$max\{t_{im}\} = \overline{T} + T^H(min\{max_{w\in V} Q_w^V, max_{k\in S} Q_k^S\}) + T^P + max_{k\in S} T_{ki}^D \tag{A.12}$$

$$max\{\sum_{v\in V}(q_{imv}^L + q_{imv}^U\}) = min\{max_{w\in V} Q_w^V, Q_i^S\} \tag{A.13}$$

$$max\{T_{ij}^D\} = 0 \tag{A.14}$$

$$max\{-t_{jn}\} = 0 \tag{A.15}$$

Constraints (5.14):

$$t_{im} - (t_{i(m-1)} + T^H \sum_{v \in V} (q^L_{i(m-1)v} + q^U_{i(m-1)v}) + T^P) \sum_{j \in S} \sum_{n \in M} \sum_{v \in V} x_{imjnv} \geq 0$$
$$i \in S, m \in M \backslash \{1\}$$

Constraints (5.14) say that if visit (i,m) is planned, then this visit has to start after visit $(i,m-1)$ is done with handling. Constraints (A.16) is a linear version of this constraint. Equation (A.17) shows the terms that have to be maximized in big-M. The terms that have to be maximized in M are specified in Equations (A.18) and (A.19), and follow the same logic as described above.

$$t_{im} - (t_{i(m-1)} + T^H \sum_{v \in V} (q^L_{i(m-1)v} + q^U_{i(m-1)v}) + T^P) + M(1 - \sum_{j \in S} \sum_{n \in M} \sum_{v \in V} x_{imjnv}) \geq 0 \quad i \in S, m \in M \backslash \{1\} \tag{A.16}$$

$$M = max\{t_{i(m-1)} + T^H \sum_{v \in V} (q^L_{i(m-1)v} + q^U_{i(m-1)v}) + T^P\} \tag{A.17}$$

where:

$$max\{t_{i(m-1)}\} = \overline{T} + T^H (min\{max_{w \in V} Q^V_w, max_{k \in S} Q^S_k\}) + T^P + max_{k \in S} T^D_{ki} \tag{A.18}$$

$$max\{\sum_{v \in V} (q^L_{i(m-1)v} + q^U_{i(m-1)v})\} = min\{max_{v \in V} Q^V_v, Q^S_i\} \tag{A.19}$$

Constraints (5.16):

$$(1 - \sum_{v \in V} x_{imd(v)1v}) \cdot t_{im} \leq \overline{T} \qquad i \in S \backslash \{d\}, m \in M$$

Constraints (5.16) restrict the visit time at a station to be before the time horizon as long as it is not the last station the vehicle visits. Constraints (A.20) show the reformulation using big-M. M is set to the maximum value $t_{im} + \overline{T}$ can take, shown in Equation (A.21). Equation (A.22) specifies the variable term in big-M.

$$t_{im} - \overline{T} - M \sum_{v \in V} x_{imd(v)1v} \leq 0 \quad i \in S \backslash \{d\}, m \in M \tag{A.20}$$

$$M = max\{t_{im} - \overline{T}\} \tag{A.21}$$

$$max\{t_{im}\} = \overline{T} + T^H (min\{max_{w \in V} Q^V_w, max_{k \in S} Q^S_k\}) + T^P + max_{k \in S} T^D_{ki} \tag{A.22}$$

Constraints (5.17):

$$t_{im}(1-\sum_{j\in S}\sum_{n\in M}\sum_{v\in V}x_{imjnv}) \leq 0 \quad i\in S\backslash\{d\}, m\in M$$

Constraints (5.17) say that the visit time must be zero if no visits remade. Constraints (A.23) is a linear version.

$$t_{im} \leq M\sum_{j\in S}\sum_{n\in M}\sum_{v\in V}x_{imjnv} \quad i\in S\backslash\{d\}, m\in M \tag{A.23}$$

where

$$M = max\{t_{im}\} = \overline{T} + T^H(min\{max_{w\in V}Q^V_w, max_{k\in S}Q^S_k\}) + T^P + max_{k\in S}T^D_{ki}$$

Constraints (5.18):

$$t_{im}(1-\sum_{j\in S}\sum_{n\in M}\sum_{v\in V}x_{i(m-1)jnv}) \leq 0 \quad i\in S, m\in M|\{1\}$$

Constraints (5.18) say that visit $(m-1)$ has to be done before visit m. Constraints (A.24) are linear versions of these constraints.

$$t_{im} \leq M\sum_{j\in S}\sum_{n\in M}\sum_{v\in V}x_{i(m-1)jnv} \quad i\in S, m\in M|\{1\} \tag{A.24}$$

where

$$M = max\{t_{im}\} = \overline{T} + T^H(min\{max_{w\in V}Q^V_w, max_{k\in S}Q^S_k\}) + T^P + max_{k\in S}T^D_{ki}$$

A.2 Violations and deviations

Constraints (5.30), (5.32), (5.33), (5.35) and (5.37) - (5.45) are nonlinear and need to be linearized before implementation. Big-M formulations are used for linearization. The M values are specified, and are of sufficient size, and as tight as possible.

Constraints (5.30), (5.32), (5.33), and (5.35):

$$\begin{aligned} t_{im}(1-\delta^{\overline{T}}_{im}) &\leq \overline{T} && i \in S, m \in M \\ t_{im} \cdot (1-\theta_{im}) &\leq t_{i(m+1)} && i \in S, m \in M \backslash \{|M|\} \\ t_{im} &\geq t_{i(m+1)}\theta_{im} && i \in S, m \in M \backslash \{|M|\} \\ t_{i|M|}(1-\theta_{i|M|}) &\leq 0 && i \in S \end{aligned}$$

Recall the binary variables:

$\delta^{\overline{T}}_{im}$	1 if visit (i,m) is after the time horizon $\overline{T}$, 0 otherwise
θ^{f}_{im}	1 if visit (i,m) is the last visit for station i, 0 otherwise
γi	1 if station i gets at least one visit, 0 otherwise

These constraints can be linearized with Constraints (A.25), (A.26), (A.27), and (A.28), respectively.

$$t_{im} \leq \overline{T} + M\delta^{\overline{T}}_{im} \qquad i \in S, m \in M \tag{A.25}$$

$$t_{im} \leq t_{i(m+1)} + M\theta_{im} \qquad i \in S, m \in M \backslash \{|M|\} \tag{A.26}$$

$$t_{im} \geq t_{i(m+1)} - M(1-\theta_{im}) \qquad i \in S, m \in M \backslash \{|M|\} \tag{A.27}$$

$$t_{i|M|} \leq M\theta_{i|M|} \qquad i \in S \tag{A.28}$$

where

$$M = max\{t_{im}\} = \overline{T} + T^{H}(min\{max_{w \in V} Q^{V}_{w}, max_{k \in S} Q^{S}_{k}\}) + T^{P} + max_{k \in S} T^{D}_{ki}$$

Constraints (5.37):

$$(-s_i + L^{S,o}_i + D_i\overline{T} - v^{C,f}_i + v^{S,f}_i)(1-\gamma_i) = 0 \qquad i \in S \backslash \{d\}$$

Constraints (5.37) capture the violations from the beginning of the time horizon until the end of the time horizon for stations that do not get any visit, i.e.γ_i = 0. These constraints can be linearized by Constraints (A.29) and (A.30).

$$s_i \leq L^{S,o}_i + D_i\overline{T} - v^{C,f}_i + v^{S,f}_i + M\gamma_i \qquad i \in S \backslash \{d\} \tag{A.29}$$

$$s_i \geq L^{S,o}_i + D_i\overline{T} - v^{C,f}_i + v^{S,f}_i - M\gamma_i \qquad i \in S \backslash \{d\} \tag{A.30}$$

where

$$M = Q_i^S$$

Constraints (5.38):

$$(-s_I + l_{im}^S + \sum_{v \in V}(q_{imv}^L - q_{imv}^U) + D_i(\overline{T} - t_{im}) + v_i^{S,f} - v_i^{C,f})\,(1-\delta_{im}) \cdot \theta_{im} = 0$$

$$i \in S, m \in M$$

Constraints (5.38) capture the violations from the last visit until the time horizon from stations that gets its last visit before the time horizon, i.e. $\delta_{im} = 0$ and $\theta_{im} = 1$. Constraints (A.31) and (A.32) are linearizations of these constraints.

$$s_i \geq l_{im}^S + \sum_{v \in V}(q_{imv}^L - q_{imv}^U) + D_i(\overline{T} - t_{im}) + v_i^{S,f} - v_i^{C,f} - M(\delta_{im} + 1 - \theta_{im}) i \in S, m \in M \tag{A.31}$$

$$s_i \leq l_{im}^S + \sum_{v \in V}(q_{imv}^L - q_{imv}^U) + D_i(\overline{T} - t_{im}) + v_i^{S,f} - v_i^{C,f} + M(\delta_{im} + 1 - \theta_{im}) i \in S, m \in M \tag{A.32}$$

where

$$M = Q_i^S$$

Constraints (5.39):

$$(-s_i + (v_{im}^S - v_i^{S,F}) - (v_{im}^C - v_i^{C,F}) + l_{im}^S - D_i(t_{im} - \overline{T}) + v_{im}^C - v_{im}^S)\,\delta_{im} \cdot \theta_{im} = 0$$

$$i \in S, m \in M$$

Constraints (5.39) capture the violations from the time horizon $\overline{T}$ until the last visit for stations that gets their last visit after the time horizon, i.e. $\delta_{im} = 1$ and $\theta_{im} = 1$. Constraints (A.33) and (A.34) are linearizations of these constraints.

$$s_i \geq (v_{im}^S - v_i^{S,F}) - (v_{im}^C - v_i^{C,F}) + l_{im}^S - D_i(t_{im} - \overline{T}) + v_{im}^C - v_{im}^S - M(2 - \delta_{im} - \theta_{im}) i \in S, m \in M \tag{A.33}$$

$$s_i \leq (v_{im}^S - v_i^{S,F}) - (v_{im}^C - v_i^{C,F}) + l_{im}^S - D_i(t_{im} - \overline{T}) + v_{im}^C - v_{im}^S + M(2 - \delta_{im} - \theta_{im}) i \in S, m \in M \tag{A.34}$$

where

$$M = max\{Q_i^S + |D_i| t_{im} + d_i(t_{im} - \overline{T})\}$$

$$max\{t_{im}\} = \overline{T} + T^H(min\{max_{w \in V} Q_w^V, max_{k \in S} Q_k^S\}) + T^P + max_{k \in S} T_{ki}^D$$

Constraints (5.40) and (5.41):

$$\begin{aligned}(Q_i^S - s_i)(v_{im}^C - v_i^{C,F}) = 0 \quad & i \in S, m \in M \\ s_i(v_{im}^S - v_i^{S,F}) = 0 \quad & i \in S, m \in M\end{aligned}$$

Constraints (5.40) and (5.41) ensure that the violations can be greater than zero only if the station is actually full or empty at the time of visit.To make these constraints linear we introduce the following binary variables and constraints:

$\delta_i^{C,H}$	1 if station i is full at the time horizon, 0 otherwise
$\delta_i^{S,H}$	1 if station i is empty at the time horizon, 0 otherwise

Constraints (5.40) and (5.41) can be linearized by Constraints (A.35)-(A.38).

$$s_i + Q_i^S \delta_i^{S,H} \leq Q_i^S \qquad i \in S \backslash \{d\} \tag{A.35}$$

$$(v_{im}^S - v_i^{S,F}) - M(\delta_i^{S,H} - \theta_{im} - \delta_{im}^{\overline{T}} + 2) \leq 0 \qquad i \in S \backslash \{d\}, m \in M \tag{A.36}$$

$$s_i - Q_i^S \delta_i^{C,H} \geq 0 \qquad i \in S \backslash \{d\} \tag{A.37}$$

$$(v_{im}^C - v_i^{C,F}) - M(\delta_i^{C,H} - \theta_{im} - \delta_{im}^{\overline{T}} + 2) \leq 0 \qquad i \in S \backslash \{d\}, m \in M \tag{A.38}$$

where

$$M = |D_i|\overline{T}$$

Constraints (5.42) and (5.43):

$$\begin{aligned}v_i^{C,F}(1 - \sum_{m \in M} \delta_{im}^{\overline{T}}) = 0 \qquad & i \in S \backslash \{d\} \\ v_i^{S,F}(1 - \sum_{m \in M} \delta_{im}^{\overline{T}}) = 0 \qquad & i \in S \backslash \{d\}\end{aligned}$$

Constraints (5.42) and (5.43) ensure that the violations after the time horizon equal zero if the last visit is before the time horizon. Constraints (A.39) and (A.40) represent linear versions of these.

$$v_i^{C,F} \leq M \sum_{m \in M} \delta_{im}^{\overline{T}} \qquad i \in S \backslash \{d\} \tag{A.39}$$

$$v_i^{S,F} \leq M \sum_{m \in M} \delta_{im}^{\overline{T}} \qquad i \in S \backslash \{d\} \tag{A.40}$$

where

$$M = max\{|D_i|(t_{im} - \overline{T})\}$$
$$max\{t_{im}\} = \overline{T} + T^H(min\{max_{w \in V} Q_w^V, max_{k \in S} Q_k^S\}) + T^P + max_{k \in S} T_{ki}^D$$

Constraints (5.44) and (5.45):

$$v_i^{C,F} \cdot \theta_{im} \cdot \delta_{im} \leq v_{im}^C \qquad i \in S \backslash \{d\}, m \in M$$
$$v_i^{S,F} \cdot \theta_{im} \cdot \delta_{im} \leq v_{im}^S \qquad i \in S \backslash \{d\}, m \in M$$

Constraints (5.44) and (5.45) specify the relationship between the violations after the time horizon and the violations between this station visit and the previous station visit. Constraints (A.41) and (A.42) represent linear versions of these.

$$v_i^{C,F} - M(2 - \theta_{im} - \delta_{im}) \leq v_{im}^C \qquad i \in S \backslash \{d\}, m \in M \tag{A.41}$$

$$v_i^{S,F} - M(2 - \theta_{im} - \delta_{im}) \leq v_{im}^S \qquad i \in S \backslash \{d\}, m \in M \tag{A.42}$$

where

$$M = max\{|D_i|(t_{im} - \overline{T})\}$$
$$max\{t_{im}\} = \overline{T} + T^H(min\{max_{w \in V} Q_w^V, max_{k \in S} Q_k^S\}) + T^P + max_{k \in S} T_{ki}^D$$

A.3 Objective Function

Constraints 5.57:

$$t_v^f = \sum_{i \in S} \sum_{m \in M} x_{imd(v)1v} \cdot t_{im} - \overline{T} \qquad v \in V$$

Constraints (5.57) track the remaining driving time to the last station visit beyond the time horizon, and Constraints (A.43) and (A.44) show the reformulation using big-M. M is set to the maximum value $t_v^f - t_{im} + \overline{T}$ can take, shown in Equation (A.45). Equations (A.46) and (A.47) specify the terms in big-M.

$$t_v^f - t_{im} + \overline{T} - M(1 - x_{imd(v)1v}) \leq 0 \quad i \in S \backslash \{d\}, m \in M, v \in V \tag{A.43}$$

$$t_v^f - t_{im} + \overline{T} + M(1 - x_{imd(v)1v}) \geq 0 \quad i \in S \backslash \{d\}, m \in M, v \in V \tag{A.44}$$

where

$$M = max\{t_v^f - t_{im} + \overline{T}\} \tag{A.45}$$

$$max\{t_v^f\} = T^P + T^H \cdot \min\{Q_v^V, max_{j \in S} Q_j^S\} + max_{j,k \in S} T_{jk}^D \tag{A.46}$$

$$max\{-t_{im}\} = 0 \tag{A.47}$$

Appendix B

Parameter Tuning

B.1 Parameter Tuning for MP Version 1

In version 1 of the master problem, the loading quantities and the visit times are determined in the master problem, along with the best combination of geographical routes.

Weights in the Criticality Score

To determine the criticality score for version 1 of the master problem, a branching constant *B* of 3 is used. Table B.1 summarizes important findings from the test. The third and fourth columns, marked as *Lowest obj. val.* and *Highest obj.val.*, indicate the lowest and highest objective values found out of the 286 weight combinations, respectively. The fifth column states the percentage of the weight combinations that results in the lowest objective value.

Table B.1: Testing of criticality score weights, V1

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)

Solution method V1 ($B = 3$)

Clustering = False

Pricing problem = False

Instance	**Hour**	Lowest obj.val.	Highest obj.val.	% with lowest obj.val
1_8				
	07:00	55.67	56.13	82.1
	17:00	18.05	18.90	93.4
2_50				
	07:00	207.49	219.03	16.1
	17:00	154.91	167.02	0.3
3_100				
	07:00	381.08	394.37	15.4
	17:00	230.72	235.12	1.1
4_158				
	07:00	514.32	535.76	0.3
	17:00	343.60	360.64	0.7

For test instance 1_8, almost all weight combinations result in the same objective value. We can conclude that with such small test instances, the lowest objective value is easily found regardless of the values of the weights in the criticality score. For test instance 2_50 and 3_100, we observe that several weight combinations find the lowest objective value. Common factors are that the driving time weight w^k is low and the remaining weights all have a value higher than zero.

The results with test instance 4_158, show a clear trend that the model performs best when the time to violation weight w^t is low, and when the net demand weight w^n and the deviation weight w^o are high. The top twenty results are obtained when w^t is 0.0 or 0.1. The best weight combinations for instance 4_158 at 07:00 and 17:00 are $(w^t, w^n, w^d, w^o) = (0.0, 0.7, 0.0, 0.3)$, and $(0.1, 0.6, 0.1, 0.2)$, respectively. Based on the trends discovered, the weight combination is set as $(w^t, w^n, w^k, w^o) = (0.1, 0.7, 0.0, 0.2)$. The chosen values are used for further testing for version 1.

Branching Constant and Number of Possible Visits

The branching constant is varied from $B = 3$ to $B = 9$, incremented by 2, while the maximum number of visits $\overline{M}$ is set to 1. Table B.2 shows the results.

Table B.2: Testing of branching constant when $\overline{M} = 1$, V1

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V1 ($B = x$)
Clustering = False
Pricing problem = False

		B = 3	**B = 5**	**B = 7**	**B = 9**
Instance	**Hour**	obj.val. / time	obj.val. / time	obj.val. / time	obj.val. / time
1_8					
	07:00	55.67 / 0.92	55.67 / 1.13	55.67 / 1.14	55.67 / 1.07
	17:00	18.01 / 0.57	18.01 / 0.67	18.01 / 0.68	18.01 / 0.63
2_50					
	07:00	213.31 / 0.95	213.31 / 2.18	213.31 / 5.76	213.31 / 20.47
	17:00	160.59 / 0.91	158.81 / 1.79	158.81 / 3.68	158.81 / 10.82
3_100					
	07:00	395.52 / 1.10	395.30 / 2.49	389.95 / 4.83	389.95 / 15.35
	17:00	243.61 / 1.44	239.25 / 2.60	239.25 / 4.99	239.25 / 12.76
4_158					
	07:00	540.70 / 2.14	540.68 / 3.27	540.38 / 5.64	538.24 / 22.60
	17:00	370.54 / 1.98	365.97 / 3.11	364.97 / 4.47	362.61 / 11.20

We observe that the objective value improves and the computational time increases as the branching constant increases. As reasonable time is defined to be ten seconds, a branching constant of $B = 9$ seems to be too high. Thus, we conclude that a branching constant of 7 produces the best solutions within a reasonable time.

To ensure a reasonable computational time when $\overline{M}$ increases, lower branching constants are tested while the number of possible visits M is set to 2. The branching constant is varied from $B = 3$ to $B = 7$, incremented by 2. Table B.3 presents the results.

Table B.3: Testing of branching constant when $\overline{M} = 2$, V1

Test instance x ($\overline{T} = 20$, $V = 2$ $\overline{M} = 2$)

Solution method V1 ($B = x$)

Clustering = False

Pricing problem = False

		B = 3	**B = 5**	**B = 7**
Instance	**Hour**	obj.val. / time	obj.val. / time	obj.val. / time
1_8				
	07:00	55.67 / 4.02	55.67 / 4.75	55.67 / 4.77
	17:00	18.01 / 1.23	18.01 / 1.35	18.01 / 1.48
2_50				
	07:00	213.31 / 2.189	213.31 / 19.23	213.31 / 130.05
	17:00	160.59 / 2.55	158.81 / 16.67	158.81 / 71.00
3_100				
	07:00	395.52 / 3.16	395.30 / 13.41	389.95 / 100.45
	17:00	243.61 / 4.13	239.25 / 21.07	239.15 / 76.72
4_158				
	07:00	540.70 / 6.18	540.69 / 17.09	540.38 / 61.94
	17:00	370.04 / 5.99	365.97 / 12.73	364.96 / 65.64

As shown in Table B.3, the computational time increases considerably when multiple visits to a station is allowed, while the objective values improve marginally or remain unchanged compared to when $\overline{M} = 1$. A branching constant of 3 must be used when $\overline{M} = 2$ to maintain reasonable computational time.

Figure B.1 illustrates the objective value for instance 4_158 at 17:00. When $B = 3$ and $\overline{M} = 2$, there is a 0.3% improvement in the objective value compared to when $\overline{M} = 1$. When $B = 5$ or $B = 7$, even smaller improvements occur. The computational time increases drastically when $\overline{M} = 2$. When $B = 9$ and $\overline{M} = 2$, the model provides no solution within 200 seconds, whereas a 0.6 % reduction is obtained when $\overline{M} = 1$. Within reasonable time, the model result in an objective value of 370.04 when $\overline{M} = 2$, and an objective value of 364.97 when $\overline{M} = 1$. This demonstrates that although a better solution is found when $\overline{M} = 2$ compared to when $\overline{M} = 1$ when the same branching constant is used, we can increase the branching constant when $\overline{M} = 1$ and find even better solutions. Hence, allowing multiple visits is concluded to reduce the quality of the model.

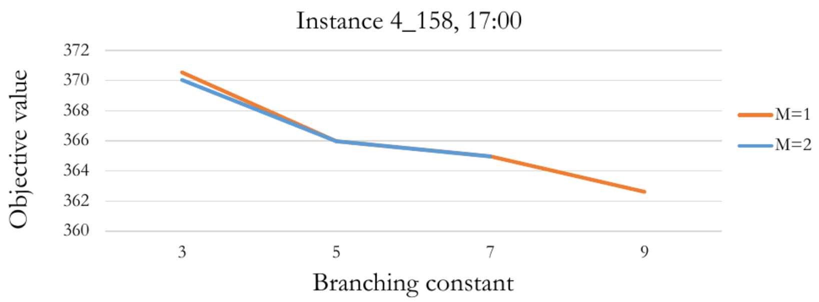


Figure B.1: Instance 4_158, 17:00: Objective value as a function of branching constant B and the number of possible visits $\overline{M}$, V1

Parameter Tuning for Clustering Problem

Different combinations of the clustering weights w^K, w^N and w^Z are tested. All the weights are varied from 0.0 to 1.0, incremented by 0.1. Also, they must sum up to 1.0. This produces a total of 66 weight combinations for each test instance. The lowest and the highest objective values are observed for each instance. The percentage of combinations that gives the lowest objective value is stated, as this conveys the impact the weights have on the quality of the clusters. Table B.4 presents the results.

Table B.4: Testing of weights in clustering problem

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V1 ($B = 7$)
Clustering = True ($C^H = 90$, $C^L = 25$)
Pricing problem = False

Instance	**Hour**	Lowest obj.val.	Highest obj.val.	% with lowest obj.val.
2_50				
	07:00	213.94	215.44	2.4
	17:00	158.81	164.76	15.5
3_100				
	07:00	389.36	394.57	84.4
	17:00	239.32	239.55	84.4
4_158				
	07:00	536.90	537.14	34.4
	17:00	364.81	365.02	40.4

For all instances, we observe that the solution is better when the net demand weight w^N is between 0.0 and 0.2. For instance 3_100 and 4_158, we observe that the clustering performs better when the driving time w^K is weighted the highest. The gap between the lowest and highest objective value is relatively small for all instances, which suggests that the impact of the clustering weights is minor. Based on the observed trends, the clustering weights are set to $(w^K, w^N, w^Z) = (0.5, 0.1, 0.4)$.

Whether a station has low, medium or high net demand, i.e. belongs to zero, one or two clusters, is determined based on the demand parameters C^H and C^L. C^H is varied from 0 to 50, and C^H is varied from 50 to 100, both incremented by 10. The result from the testing is presented in Table B.5.

Table B.5: Testing of high and low demand parameters, C^H and C^L, in clustering problem

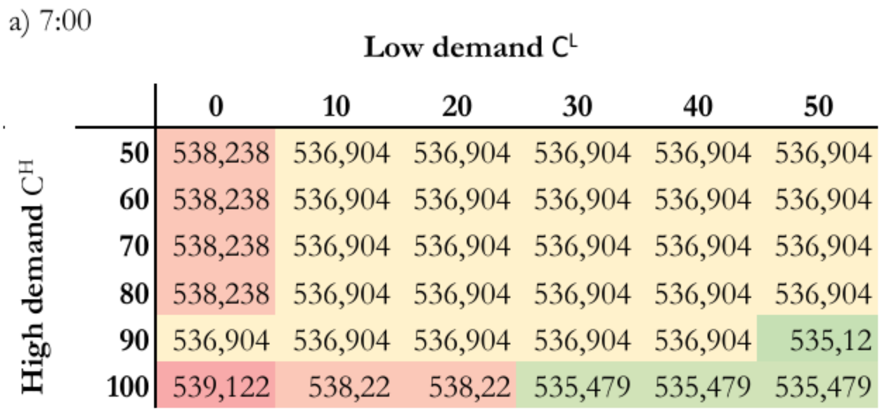
Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V1 ($B = 7$)
Clustering = True($C^H = x$, $C^L = x$)
Pricing problem = False

Instance	Hour	Lowest obj.val.	Highest obj.val.	% with lowest obj.val.
2_50				
	07:00	213.31	214.32	66.7
	17:00	158.81	159.1	86.1
3_100				
	07:00	389.07	393.07	52.8
	17:00	236.38	239.55	8.3
4_158				
	07:00	535.12	539.12	2.8
	17:00	362.62	366.74	16.7

For instance 2_50, the model performs best when C^H is lower than 80. The percentages of combinations with the lowest objective value are relatively high, meaning many different combinations perform well. For instance 3_100, the objective values are lowest when C^L is lower than 40, and when C^H is lower than 90.

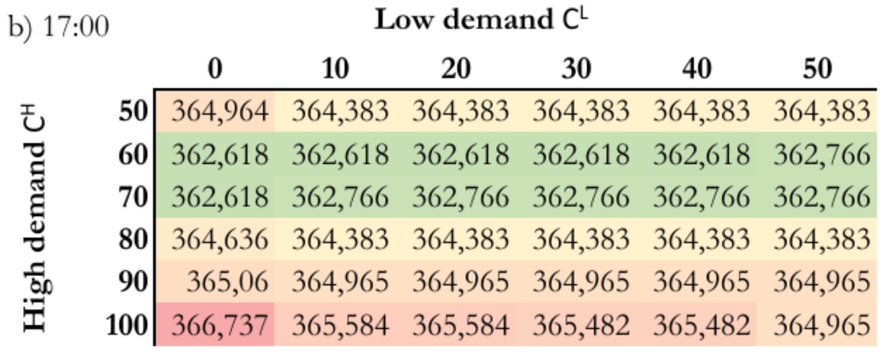
Figure B.2 illustrates the objective values for instance 4_158 when the demand parameters are varied. a) presents the results from 7:00, and b) from 17:00. The green and red colors indicate lower and higher objective values, respectively. We observe that the green areas do not overlap, meaning that different combinations perform well on different instances. However, both have worse performance when C^L is very low, and C^H very high.

Based on all the observed trends, the C^H and C^L parameters are set to 70 and 30, respectively. Note that these values are used in the rest of this paper, and are not tested again for version 2 and 3.



a) 7:00

High demand C^H	Low demand C^L					
	0	10	20	30	40	50
50	538,238	536,904	536,904	536,904	536,904	536,904
60	538,238	536,904	536,904	536,904	536,904	536,904
70	538,238	536,904	536,904	536,904	536,904	536,904
80	538,238	536,904	536,904	536,904	536,904	536,904
90	536,904	536,904	536,904	536,904	536,904	535,12
100	539,122	538,22	538,22	535,479	535,479	535,479



b) 17:00

High demand C^H	Low demand C^L					
	0	10	20	30	40	50
50	364,964	364,383	364,383	364,383	364,383	364,383
60	362,618	362,618	362,618	362,618	362,618	362,766
70	362,618	362,766	362,766	362,766	362,766	362,766
80	364,636	364,383	364,383	364,383	364,383	364,383
90	365,06	364,965	364,965	364,965	364,965	364,965
100	366,737	365,584	365,584	365,482	365,482	364,965

Figure B.2: Instance 4_158 7:00 and 17:00: Objective value as a function of clustering weights

Effect of Introducing Clustering

Table B.6 compares the objective values for all instance/hour combinations when clustering is included and excluded. The objective value obtained with clustering is marked as red, green or with nothing depending on whether it produces a higher, lower, or unchanged value, respectively.

Table B.6: Testing of clustering, V1

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V1 ($B = 7$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = False

		Clustering = False	**Clustering = True**
Instance	**Hour**	Objective value	Objective value
2_50			
	07:00	213.31	213.31
	17:00	158.81	158.81
3_100			
	07:00	389.95	389.36
	17:00	239.25	239.25
4_158			
	07:00	540.38	536.90
	17:00	364.96	362.77

For three of the six instance/hour combinations, the results remain unchanged. For instance 4_158 at both hours, and for instance 3_100 at 07:00, the objective value is improved, marked with green in the table. As the objective value remains unchanged or improves, and, most importantly, do not worsen in any of the test instances, we conclude that clustering should be included when version 1 is used.

Parameter Tuning for Pricing Problems

First, the pricing problem score weight w^{pp} within the criticality score is tested. The weight is varied from 3 to 7, incremented by 1. The weight is set relatively high compared to the other weights in the criticality score as this increases the change of creating routes with stations that were not visited before.

The different weight combinations are tested on all instance/hour combinations, with a time horizon $\overline{T}$ of 20 minutes, a maximum number of visits to each station $\overline{M}$ of 1, and with two service vehicles. As the other pricing problem parameters are not yet determined, these are set based on our hypothesis: the pricing problem is run once for each test, the pricing problem branching constant is 3, and the probability of including the pricing problem score in the criticality score is set to fifty percentage, i.e. $n^{pp} = 1, B^{pp} = 3, p^{pp} = 0.5$.

Table B.7 presents the results. When the different weights w^{pp} produces different average objective values for a test instance, the best result is marked with green, and the worst with red.

Table B.7: Testing of pricing problem score weight

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V1 ($B = 7$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = True($n^{pp} = 1, B^{pp} = 3, p^{pp} = 0.5$)

		$w^{pp} = 3$	$w^{pp} = 4$	$w^{pp} = 5$	$w^{pp} = 6$	$w^{pp} = 7$
Instance	**Hour**	obj. val.	obj. val.	obj. val.	obj. val.	obj. val.
1_8						
	07:00	55.67	55.67	55.67	55.67	55.67
	17:00	18.01	18.01	18.01	18.01	18.01
2_50						
	07:00	213.31	213.31	213.31	213.31	213.31
	17:00	158.81	158.81	158.81	158.81	158.81
3_100						
	07:00	389.36	389.36	389.36	389.36	389.36
	17:00	238.05	237.60	237.60	237.81	237.69
4_158						
	07:00	536.90	536.13	536.35	536.91	536.55
	17:00	362.77	362.77	362.77	362.77	362.77

Table B.7 shows that the objective value is dependent on the weight value only for two of the eight instance/hour combinations. For instance 3_100, 17:00, the problem performs best when the value is 4 or 5, and for instance 4_158, 7:00, the pricing problem performs best when the value is 4. Thus, the weight value is set to 4.

Further, different values of the probability p^{pp} of including the pricing problem score in the the criticality score is tested. The probability parameter is varied from 0.2 to 1.0, incremented by 0.2. Table B.8 presents the results. Similarly, when the different probabilities p^{pp} produces different average objective values for a test instance, the best result is marked with green, and the worst with red.

Table B.8: Testing of probability of including pricing problem score

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V1 ($B = 7$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = True($n^{pp} = 1, B^{pp} = 3, p^{pp} = x$

		$p^{pp} = 0.2$	$p^{pp} = 0.4$	$p^{pp} = 0.6$	$p^{pp} = 0.8$	$p^{pp} = 1.0$
Instance	**Hour**	obj. val.	obj. val.	obj. val.	obj. val.	obj. val.
1_8						
	07:00	55.67	55.67	55.67	55.67	55.67
	17:00	18.01	18.01	18.01	18.01	18.01
2_50						
	07:00	213.31	213.31	213.31	213.31	213.31
	17:00	158.81	158.81	158.81	158.81	158.81
3_100						
	07:00	389.36	389.36	389.36	389.36	389.36
	17:00	238.77	237.79	236.90	237.69	237.45
4_158						
	07:00	536.19	536.53	536.90	536.90	536.90
	17:00	362.77	362.77	362.77	362.776	362.77

From Table B.8, we observe that the results are dependent on the probability value only for test instance 3_100 at 17:00 and instance 4_158 at 07:00. The pricing problem performs best when $p^{pp} = 0.6$ for instance 3_100 at 17:00, and when $p^{pp} = 0.2$ for instance 4_158 at 07:00. However, the pricing problem is chosen as $p^{pp} = 0.4$, since this works well for both test instances. Note that these values are used also for version 2 and 3.

Effect of Introducing Pricing Problem

The number of pricing problem runs n^{pp} is varied from 1 to 3, and the branching constant in the pricing problem B^{pp} is varied from 2 to 4. In theory, the solution should improve when n^{pp} or B^{pp} increases. However, due to randomness in the pricing problem this does not always occur.

Without the pricing problem, and with a branching constant of 7, the computational times are just slightly within reasonable time. However, with the pricing problem and $B = 7$, the computational times become unreasonable. Hence, this test is conducted with a lower branching constant of 5. When conducting the test, we observe that instance 3_100 at 17:00, and instance 4_158 at 7:00 and 17:00 are the only instances where the objective values change as a function of B^{pp} and n^{pp}. Table B.9 presents the results obtained when testing on those three instance/hour combinations.

Table B.9: Testing of pricing problem parameters, V1

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V1 ($B = 5$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = True($n^{pp} = x, B^{pp} = x, p^{pp} = 0.4$)

			$n^{pp} = 1$	$n^{pp} = 2$	$n^{pp} = 3$
Instance	**Hour**	B^{pp}	obj. val. / time	obj. val. / time	obj. val. / time
3_100					
	17:00	2	237.74 / 6.48	237.26 / 10.07	237.74 / 15.37
		3	238.68 / 6.96	237.26 / 11.03	237.03 / 17.36
		4	238.29 / 8.54	237.87 / 14.41	236.67 / 21.13
4_158					
	7:00	2	538.74 / 9.29	537.77 / 15.33	537.81 / 20.93
		3	537.36 / 9.93	536.38 / 20.03	535.01 / 28.51
		4	536.01 / 12.28	535.25 / 29.60	534.21 / 41.94
	17:00	2	364.38 / 7.67	364.38 / 12.98	363.84 / 17.36
		3	363.93 / 9.33	364.38 / 16.74	364.24 / 25.31
		4	364.38 / 10.89	364.04 / 22.04	364.08 / 37.08

As observed in Table B.9, the model produces best results within reasonable time when $n^{pp} = 1$ and $B^{pp} = 3$. This is marked with green in the table.

Introducing the pricing problem is at the expense of a higher branching constant B in the initialization as it must be reduced to 5 to keep the computational time reasonable. To conclude whether the pricing problem should be applied on not, Table B.10 compares the objective values obtained when $B = 7$ and with no pricing problem, to the objective values obtained with $B = 5$ and with the pricing problem. The objective value obtained with the pricing problem is marked as red, green or with nothing depending on whether it produces a higher, lower, or unchanged value, respectively.

Table B.10: Effect of introducing pricing problem, V1

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V1 ($B = 7/5$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = True($n^{pp} = 1, B^{pp} = 3, p^{pp} = 0.4$)

		B = 7	**B = 5**
		Pricing problem = False	**Pricing problem = True**
Instance	**Hour**	Objective value	Objective value
1_8			
	07:00	55.67	55.67
	17:00	18.08	18.08
2_50			
	07:00	213.31	213.31
	17:00	158.81	158.81
3_100			
	07:00	389.36	389.36
	17:00	239.25	238.68
4_158			
	07:00	536.90	537.36
	17:00	362.77	363.93

The objective value gets better for instance 3_100 at 17:00, and worse for instance 4_158, both at 7:00 and 17:00, when the pricing problem is included. The fact that a higher branching constant during the initialization of columns is better than including the pricing problem and using a lower branching constant, demonstrates that the initialization algorithm performs well. As larger instances are prioritized, including the pricing problem in version 1 is concluded to be counterproductive.

Number of vehicles

The model may be solvable with a higher number of service vehicles. However, the branching constant must be adjusted to keep the computational time reasonable. Table B.11 shows the highest possible branching constant that can be applied with an increasing number of service vehicles.

Table B.11: Highest possible branching constant B for different numbers of vehicles V, V1

Test instance x ($\overline{T} = 20, V = x, \overline{M} = 1$)
Solution method V1 ($B = x$)
Clustering = True($C^H = 70, C^L = 30$)
Pricing problem = False

		V=3	**V=4**	**V=5**
Instance	**Hour**	Highest possible B	Highest possible B	Highest possible B
1_8				
	07:00	7	7	7
	17:00	7	7	7
2_50				
	07:00	6	4	3
	17:00	5	4	3
3_100				
	07:00	6	3	2
	17:00	5	3	2
4_158				
	07:00	5	3	2
	17:00	4	3	2

As observed in Table B.11, if extra service vehicles are added, the branching constant must be decreased to keep the computational time reasonable. The highest possible branching constants when 3, 4, and 5 service vehicles are included are 4, 3 and 2, respectively.

B.2 Parameter Tuning for MP Version 2

In version 2, the loading quantities and visit times are predetermined in the initialization, but the master problem has a given flexibility to adjust these values later. To start with the flexibility is set to its median value of 12, before it is tested and its best-performing value is obtained.

Weights in the Criticality Score

A branching constant B of 3 is used, and Table B.12 summarizes important findings from the test. The third and fourth columns, marked as *Lowest obj. val.* and *Highest obj.val.*, indicate the lowest and highest objective values found out of the 286 weight combinations, respectively. The fifth column states the percentage of the weight combinations that results in the lowest objective value.

Table B.12: Testing of criticality score weights, V2

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V2 ($B = 3$, $F = 12$)
Clustering = False
Pricing problem = False

Instance	**Hour**	Lowest obj.val.	Highest obj.val.	% with lowest obj.val.
1_8				
	07:00	55.67	56.13	82.3
	17:00	18.01	18.90	92.1
2_50				
	07:00	213.31	223.84	15.9
	17:00	159.20	174.44	0.4
3_100				
	07:00	395.40	403.67	4.8
	17:00	239.05	252.32	1.8
4_158				
	07:00	539.39	549.27	0.4
	17:00	365.59	379.86	18.3

For instance 1_8, the values of the weights have minor significance as the model is able to detect the solution with the lowest objective value for almost all weight combinations. For test instance 2_50, the best solution is found when the driving time is not weighted. For instance 3_100, multiple weight combinations yield the lowest objective value. Generally, when the best solution is found, net demand and deviation are weighted highest, while the driving distance is given minor weight. For instance 4_158 at 07:00, the lowest objective value is only found once with the weight combination $(w^t, w^n, w^k, w^o) = (0.0, 0.9, 0.0, 0.1)$. The values of the weights have less impact at 17:00, but generally, the best solutions are found when the time to violation and driving time are weighted between 0.0 or 0.1. The distribution between net demand and deviation does not seem to be significant. Based on the trends discovered from the analyses, the weight combination is set as $(w^t, w^n, w^k, w^o) = (0.1, 0.8, 0.0, 0.1)$. The chosen values for the weight coefficients are used for further testing of version 2.

Branching Constant and Number of Possible Visits

When each station is only allowed one visit, different branching constants yield the results presented in Table B.13. The branching constant is varied from $B = 3$ to $B = 9$, incremented by 2.

Table B.13: Testing of branching constant when $\overline{M} = 1$, V2

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V2 ($B = x$, $F = 12$)
Clustering = False
Pricing problem = False

		B = 3	**B = 5**	**B = 7**	**B = 9**
Instance	**Hour**	obj.val. / time	obj.val. / time	obj.val. / time	obj.val. / time
1_8					
	07:00	55.67 / 1.11	55.67 / 0.96	55.67 / 0.86	55.67 / 0.84
	17:00	18.01 / 0.65	18.01 / 0.45	18.01 / 0.65	18.01 / 0.57
2_50					
	07:00	213.31 / 1.04	213.31 / 2.52	213.31 / 9.22	213.31 / 24.47
	17:00	166.34 / 0.96	159.20 / 3.31	158.81 / 4.89	158.81 / 9.69
3_100					
	07:00	396.50 / 1.12	395.52 / 1.96	390.40 / 6.07	390.40 / 16.95
	17:00	244.87 / 1.27	240.85 / 2.67	239.51 / 8.52	239.51 / 10.98
4_158					
	07:00	541.95 / 1.92	540.70 / 3.28	540.68 / 6.41	536.51 / 21.09
	17:00	377.61 / 1.83	370.76 / 3.07	365.59 / 6.66	365.59 / 12.40

For test instance 1_8 and 2_50 at 7:00, the best solution is found even with the lowest branching constant. This is expected as the test instances are relatively small, and the total number of possible columns are limited. For test instance 3_100 and 4_158 the objective value improves as the branching constant increases. This is expected, as more columns increase the solution space in the master problem. However, the computational time is considerably increased when B increases. For test instance 4_158 at 07:00, the computational time increases with 998% when the branching constant is increased from 3 to 9. A branching constant of 7 seems to be the best performing value as this is the highest values that keeps the computational time reasonable. When each station is allowed two visits, different branching constants yield the results presented in Table B.14. The branching constant is varied from $B = 3$ to $B = 7$, incremented by 2.

Table B.14: Testing of branching constant when $\overline{M} = 2$, V2

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 2$)
Solution method V2 ($B = x$, $F = 12$)
Clustering = False
Pricing problem = False

		B = 3	**B = 5**	**B = 7**
Instance	**Hour**	obj.val. / time	obj.val. / time	obj.val. / time
1_8				
	07:00	55.67 / 4.62	55.67 / 5.28	55.67 / 5.39
	17:00	18.01 / 1.26	18.01 / 1.58	18.01 / 1.60
2_50				
	07:00	213.31 / 2.46	213.31 / 15.67	213.31 / 309.19
	17:00	166.34 / 2.07	159.20 / 13.41	158.81 / 68.60
3_100				
	07:00	396.50 / 3.54	395.52 / 8.09	390.41 / 75.92
	17:00	244.87 / 3.28	240.85 / 12.87	239.251 / 45.84
4_158				
	07:00	541.95 / 5.62	540.70 / 13.74	540.68 / 55.89
	17:00	377.53 / 5.14	370.51 / 15.75	365.59 / 70.46

As shown in Table B.14, the computational times are increased immensely when each station is allowed multiple visits. With a branching constant of 5, the computational time has increased on average 570% for all test instances compared to when $\overline{M} = 1$. However, the model is rarely able to detect better solutions than when $\overline{M} = 1$. Because of the drastic increase in computational time and marginally improvement in solution, allowing for multiple visits is concluded to be unnecessary.

Flexibility Parameter

The flexibility parameter F in version 2 allows for some slack in the predetermined loading quantities and visit times. When $F = 23$, version 2 has full flexibility and equals version 1 as the MP has complete control over this determination. When $F = 0$ and $\overline{M} = 1$, version 2 has no flexibility, and equals version 3 as the loading quantities are now entirely predetermined in the initialization process. If the predetermined loading quantity q^L_{imv} equals 8, a flexibility of $F = 12$, means that the final loading quantity can be in the interval $(8-12, 8+12)$. However, capacity and non-negativity constraints must still be complied. The flexibility parameter is tested for $F = 6$ to $F = 23$, incremented by 6. The aim is to observe the flexibility value that results in the best objective value in shortest possible time. The results are shown in Table B.15.

Table B.15: Testing of flexibility parameter, V2

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V2 ($B = 7$, $F = x$)
Clustering = False
Pricing problem = False

		F = 6	**F = 12**	**F = 18**	**F = 23**
Instance	**Hour**	obj.val. / time	obj.val. / time	obj.val. / time	obj.val. / time
1_8					
	07:00	56.21 / 1.35	55.67 / 0.96	55.67 / 0.85	55.67 / 1.09
	17:00	19.68 / 0.61	18.01 / 0.49	18.01 / 0.71	18.01 / 1.15
2_50					
	07:00	214.32 / 10.01	213.31 / 11.66	213.31 / 10.00	213.31 / 11.90
	17:00	161.31 / 5.08	158.81 / 4.17	158.81 / 4.23	158.81 / 4.41
3_100					
	07:00	390.90 / 5.24	390.40 / 5.34	390.40 / 6.51	390.40 / 4.46
	17:00	239.25 / 7.05	239.25 / 8.17	239.25 / 7.07	239.25 / 7.78
4_158					
	07:00	540.68 / 5.99	540.68 / 6.48	540.68 / 7.23	540.68 / 7.29
	17:00	367.58 / 7.35	365.59 / 6.98	364.72 / 6.38	364.72 / 7.33

By comparing the results obtained with a flexibility parameter of $F = 18$ and $F = 23$, we observe that the objective values are the same. This may indicate that the objective values have converged. In addition, the computational times tend to increase when F is set to 23, and we conclude that $F = 18$ dominates $F = 23$. When $F = 12$, one out of the eight test instances results in a worse objective value than when $F = 18$. Moreover, we cannot conclude that the computational times are lower for $F = 12$ than for $F = 18$ as $F = 12$ only results in a lower computational time in 50 percentage of the cases. Thus, we conclude that $F = 18$ also dominates $F = 12$. All results produced when $F = 18$ are obtained within reasonable time. based on this, $F = 18$ is concluded to be the best value.

Effect of Introducing Clustering

Table B.16 compares the objective values for all instance/hour combinations when clustering is included and excluded. The objective value obtained with clustering is marked as red, green or with nothing depending on whether it produces a higher, lower, or unchanged value, respectively.

Table B.16: Testing of clustering, V2

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V2 ($B = 7$, $F = 18$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = False

		Clustering = False	**Clustering = True**
Instance	**Hour**	Objective value	Objective value
2_50			
	07:00	213.31	213.31
	17:00	158.81	158.81
3_100			
	07:00	390.40	🟩 389.36
	17:00	239.25	239.25
4_158			
	07:00	540.68	🟩 535.15
	17:00	364.72	🟩 362.77

When clustering is including in version 2, the objective value is reduced for three of the six instance/hour combinations. As the objective value does not worsen for any instances, clustering is concluded to have a positive effect on the model and is used for version 2.

Effect of Introducing Pricing Problem

The number of runs n^{pp} is varied from 1 to 3, and the branching constant in the pricing problem B^{pp} is varied from 2 to 4. Similarly as for version 1, a branching constant of 7 gives computational times just slightly within reasonable time. However, when the pricing problem is applied while B remains 7, the computational times become unreasonable. Hence, the branching constant B must be lower. A branching constant of $B = 5$ also gives unreasonable computational times, and the branching constant is set to 3 in this test. When conducting the test, we observe that instance 3 and 4 at both hours are the only instances where the objective values change as a function of B^{pp} and n^{pp}. Hence, Table B.17 presents the results obtained when testing on those two instances.

Table B.17: Testing of pricing problem parameters, V2

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V2 ($B = 3$, $F = 18$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = True($n^{pp} = x, B^{pp} = x, p^{pp} = 0.4$)

			$n^{pp} = 1$	$n^{pp} = 2$	$n^{pp} = 3$
Instance	**Hour**	B^{pp}	obj. val. / time	obj. val. / time	obj. val. / time
3_100					
	07:00	2	389.82 / 3.38	389.70 / 6.02	389.61 / 9.82
		3	389.80 / 4.13	389.64 / 9.72	389.59 / 14.09
		4	389.87 / 5.54	389.87 / 15.54	389.66 / 26.05
	17:00	2	240.44 / 3.73	241.30 / 7.24	240.09 / 9.55
		3	240.92 / 4.55	239.90 / 8.28	238.35 / 13.03
		4	240.93 / 5.56	238.36 / 10.38	237.05 / 17.76
4_158					
	7:00	2	539.22 / 5.48	538.47 / 9.23	537.04 / 13.06
		3	538.03 / 6.74	537.13 / 13.76	535.82 / 21.83
		4	535.94 / 9.08	536.37 / 18.84	534.30 / 33.13
	17:00	2	365.62 / 5.07	365.53 / 8.26	365.51 / 13.08
		3	364.78 / 6.18	365.13 / 12.00	363.58 / 13.88
		4	365.63 / 7.24	364.85 / 16.57	364.43 / 30.08

Table B.17 illustrates that the model produces best results within reasonable time when $n^{pp} = 1$ and $B^{pp} = 3$.

Introducing the pricing problem is at the expense of a higher branching constant B in the initialization as it must be reduced to 3 to keep the computational time reasonable. Table B.18 presents the objective values for all instances and hours with and without the pricing problem, and with a branching constant of $B = 7$ and $B = 3$, respectively. The objective value obtained with the pricing problem is marked as red, green or with nothing depending on whether it produces a higher, lower, or unchanged value, respectively.

Table B.18: Effect of introducing pricing problem, V2

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V2 ($B = 7/3$, $F = 18$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = True($n^{pp} = 1, B^{pp} = 3, p^{pp} = 0.4$)

		B = 7	**B = 3**
		Pricing problem = False	**Pricing problem = True**
Instance	**Hour**	Objective value	Objective value
1_8			
	07:00	55.67	55.67
	17:00	18.01	18.01
2_50			
	07:00	213.31	213.31
	17:00	158.81	159.06
3_100			
	07:00	389.36	389.80
	17:00	239.25	240.92
4_158			
	07:00	535.15	538.03
	17:00	362.77	364.78

We observe that the objective values increase or remain unchanged when the pricing problem is applied. Thus, we conclude that the pricing problem should not be applied.

Number of Vehicles

The model may be solvable with a higher number of service vehicles. However, the branching constant must be adjusted to keep the computational time reasonable. Table B.19 shows the highest possible branching constant that can be applied with an increasing number of service vehicles. Note that all configurations concluded are kept constant.

Table B.19: Highest possible branching constant B for different numbers of vehicles V, V2

Test instance x ($\overline{T} = 20$, $V = x$, $\overline{M} = 1$)
Solution method V2 ($B = x$, $F = 18$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = False

		V=3	**V=4**	**V=5**
Instance	**Hour**	Highest possible B	Highest possible B	Highest possible B
1_8				
	07:00	7	7	7
	17:00	7	7	7
2_50				
	07:00	7	4	3
	17:00	6	3	3
3_100				
	07:00	6	3	2
	17:00	5	3	2
4_158				
	07:00	5	3	2
	17:00	5	3	2

As observed in Table B.19, if extra service vehicles are added, the branching constant must be decreased to keep the computational time reasonable. The highest possible branching constant when 3, 4, and 5 service vehicles are included is 5, 3 and 2, respectively.

B.3 Parameter Tuning for MP Version 3

In version 3, the loading quantities and visit times are entirely predetermined in the initialization. Oppose to the other versions, the stations are restricted to no more than one visit, i.e. $\overline{M} = 1$, and the optimal combination chosen by the master problem consists of unique station visits. Hence, violations and deviations may also be predetermined in the initialization and a significant simplified mathematical model of the master problem is implemented.

Weights in the Criticality Score

Version 3 is tested with a branching constant of $B = 10$. Compared to the testing of version 1 and version 2, the branching constant is drastically increased as our hypothesis is that version 3 can be solved with a considerably lower computational time. Table B.20 summarizes important findings from the test.

Table B.20: Testing of criticality score weights, V3

Test instance x ($\overline{T} = 20, V = 2\ \overline{M} = 1$)
Solution method V3 ($B = 10$)
Clustering = False
Pricing problem = False

Instance	**Hour**	Lowest obj.val.	Highest obj.val.	% with lowest obj.val.
1_8				
	07:00	56.34	56.34	100
	17:00	27.15	27.15	100
2_50				
	07:00	214.738	218.38	15.0
	17:00	167.83	177.14	49.4
3_100				
	07:00	391.14	401.79	33.3
	17:00	240.11	125.35	2.7
4_158				
	07:00	538.53	547.06	14.3
	17:00	369.39	381.00	6.2

We observe that any weight combination yield the same results for test instance 1_8. Again, we see that the % of weight combinations with the lowest objective value decreases with larger test instances. This means that it is more critical to use good weights when larger test instances are used.

For test instance 2_50 and 3_100, 7:00, we observe that the best results are found when the driving time weight w^k is lower than 0.2, the deviation weight w^o higher than 0.4, and the time to violation weight w^t lower than 0.2. For instance 2, 17:00, the same findings hold except that the lowest objective value is also found when the time to violation weight is as high as 0.7.

For test instance 4_158, 7:00, the lowest objective value is found when the time to violation weight w^t is in the interval 0.1-0.3, the net demand weight w^n in the interval 0.0-1.0, the driving time weight $w^k = 0$, and the deviation weight w^o in the interval 0.4-0.9. At 17:00 we observe very similar trends, except that the time to violation weight w^t is in the interval 0.1-0.4.

Based on the trends discovered, the weight combination is set as $(w^t, w^n, w^k, w^o) = (0.1, 0.5, 0.0, 0.4)$. The chosen values are used for further testing for version 1.

Branching Constant

As the mathematical formulation of version 3 of the master problem is considerably simplified compared to the other version, a higher branching constant may be used to obtain better solutions. The branching constant is varied from $B = 20$ to $B = 50$, and the results are presented in Table B.21.

Table B.21: Testing of branching constant when $\overline{M} = 1$, V3

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)

Solution method V3 ($B = x$)

Clustering = False

Pricing problem = False

		B = 20	**B = 30**	**B = 40**	**B = 50**
Instance	**Hour**	obj.val. / time	obj.val. / time	obj.val. / time	obj.val. / time
1_8					
	07:00	56.34 / 0.15	56.34 / 0.16	56.34 / 0.38	56.34 / 0.28
	17:00	27.15 / 0.03	27.15 / 0.03	27.15 / 0.18	27.15 / 0.15
2_50					
	07:00	214.74 / 0.41	214.74 / 0.47	214.74 / 0.68	214.74 / 0.72
	17:00	167.83 / 0.27	167.83 / 0.66	167.83 / 0.67	167.83 / 0.72
3_100					
	07:00	391.14 / 0.47	391.14 / 1.21	391.14 / 2.86	391.14 / 3.75
	17:00	240.11 / 0.29	240.11 / 0.97	240.11 / 1.33	240.11 / 2.46
4_158					
	07:00	535.56 / 0.60	535.56 / 2.36	535.56 / 8.19	535.56 / 25.43
	17:00	369.39 / 0.40	369.39 / 1.21	369.39 / 2.59	363.39 / 9.64

As shown in Table B.21, the objective values are unchanged when the branching constant is increased. With $B = 20$, the branching algorithm branches to 20 pickup stations and 20 delivery stations in each node, which we assume are sufficient on these test instances.

Effect of Introducing Clustering

Table B.22 compares the objective values for all instances and hours when clustering is included and excluded. The objective value obtained with clustering is marked as red, green or with nothing depending on whether it produces a higher, lower, or unchanged value, respectively.

Table B.22: Testing of clustering, V3

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V3 ($B = 20$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = False

		Clustering = False	**Clustering = True**
Instance	**Hour**	Objective value	Objective value
2_50			
	07:00	214.74	217.52
	17:00	167.83	167.83
3_100			
	07:00	391.14	391.27
	17:00	240.11	240.11
4_158			
	07:00	535.56	535.56
	17:00	369.39	369.39

As shown in Table B.22, the objective value is worse in two out of six instance/hour combinations when clustering is included. The reason for this may be that the branching constant is so high in version 3 that the master problem has no problem finding non-overlapping geographical routes, and applying clusters only prevents some good columns from being creates. As there is a chance of worsening the objective value, clustering should not be included in version 3.

Effect of Introducing Pricing Problem

The number of runs n^{pp} is varied from 2 to 4, and the branching constant in the pricing problem B^{pp} is varied from 5 to 15, incremented by 5. All combinations are tested and all other parameters are as determined above. In theory, the solution should improve when n^{pp} or B^{pp} increases. However, due to randomness in the pricing problem this does not always occur in practice. When conducting the test, we observe that instance 3_100 at 07:00 are the only instance where the objective value changes as a function of B^{pp} and n^{pp}. As the purpose of the first test is solely to set the parameters in the pricing problem to their best-performing values, only test instance 3_100 is presented in Table B.23.

Table B.23: Testing of pricing problem parameters, V3

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V1 ($B = 20$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = True($n^{pp} = x, B^{pp} = x, p^{pp} = 0.4$)

			$n^{pp} = 2$	$n^{pp} = 3$	$n^{pp} = 4$
Instance	**Hour**	B^{pp}	obj. val. / time	obj. val. / time	obj. val. / time
3_100					
	7:00	5	391.14 / 1.15	391.14 / 1.43	391.14 / 1.80
		10	391.14 / 1.22	391.14 / 1.67	391.14 / 2.19
		15	391.03 / 1.46	391.10 / 2.28	391.03 / 3.29

As opposed to the other versions, including the pricing problem is not at the expense of the branching constant B, as computational time are more than reasonable. As shown in Table B.23, the pricing problem is able to create new columns that improved the solution when $n^{pp} = 2$ and 4, and when $B^{pp} = 15$. As $n^{pp} = 4$ yield the same result as $n^{pp} = 2$, but with a higher computational time, the best-performing value of n^{pp} is concluded to be 2. $B^{pp} =$ is set to 15. Additionally, we observed that the computational time is reasonable when test instance 4_158 is solved with the pricing problem.

Further, the effect of introducing the pricing problem is tested. The first test was solely to set the best parameter values for the pricing problem, and thus, only the test instance with changes in the objective value with changing parameter values was presented. Next, we want to observe if introducing the pricing problem enables better solutions to be generated. Table B.24 compares the objective values for the eight instance/hour combinations when the pricing problem is included and not included. The objective value obtained with the pricing problem is marked as red, green or with nothing depending on whether it produces a higher, lower, or unchanged value, respectively.

Table B.24: Effect of introducing pricing problem, V3

Test instance x ($\overline{T} = 20$, $V = 2$, $\overline{M} = 1$)
Solution method V3 ($B = 20$)
Clustering = True($C^H = 70$, $C^L = 30$)
Pricing problem = True($n^{pp} = 2, B^{pp} = 15, p^{pp} = 0.4$)

		Pricing problem = False	**Pricing problem = True**
Instance	**Hour**	Objective value	Objective value
1_8			
	07:00	56.34	56.34
	17:00	27.15	27.15
2_50			
	07:00	214.74	214.74
	17:00	167.83	167.83
3_100			
	07:00	391.14	391.03
	17:00	240.11	240.11
4_158			
	07:00	535.56	535.56
	17:00	369.39	369.39

The objective value improves for test instance 3_100 at 07:00 when the pricing problem is included. As larger instances are prioritized, including the pricing problem in version 3 is concluded to be appropriate.

Number of Vehicles

This version of the master problem is able to run within reasonable computational time for all instances when $B = 20$, even when the number of service vehicles is set to five. Thus, the branching constant of 20 is kept regardless of the number of service vehicles.

Appendix C

Final Configurations for Column Generation Heuristic

Table C.1: Final configurations for CG heuristic

Parameter	**Value**
Master problem version	3
Crit. score weight, time to violation w^t	0.1
Crit. score weight, net demand w^n	0.5
Crit. score weight, driving time w^k	0.0
Crit. score weight, deviation w^o	0.4
Number of possible visits $\overline{M}$	1
Branching constant B	20
Clustering	False
Dynamic clustering	False
Pricing problem	True
Number of pricing problem iterations n^{pp}	2
Branching constant in pricing problem B^{pp}	15
Route re-generation point t^{route}	First vehicle arrival
Time horizon $\overline{T}$	20 minutes

Bibliography

Andersen, M. (2016). How much does each bike share ride cost a system? Let's do the math. http://betterbikeshare.org/2016/08/16/much-bike-share-ride-cost-system-lets-math/. (Accessed: 2017-September-22).

Antoniades, P. and Chrysanthou, A. (2009). European Best Practices in Bike Sharing Systems. *Students Today Citizen Tomorrow*.

Beheshti, A. K. and Hejazi, S. R. (2015). A novel hybrid column generation-metaheuristic approach for the vehicle routing problem with general soft time window. *Information Sciences*, 316:598–615.

Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684.

Brinkmann, J., Ulmer, M., and Mattfeld, D. (2015). Short-term strategies for stochastic inventory routing in bike sharing systems. *Transportation Research Procedia*, 10:364–373.

Chen, L., Zhang, D., Wang, L., Yang, D., Ma, X., Li, S., Wu, Z., Pan, G., Nguyen, T.-M.-T., and Jakubowicz, J. (2016). Dynamic cluster-based over-demand prediction in bike sharing systems. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 841–852. ACM.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581.

Clemitson, S. (2017). *A History of Cycling in 100 objects*. Bloomsbury Publishing, London, UK.

Dell, M., Iori, M., Novellani, S., Stützle, T., et al. (2016). A destroy and repair algorithm for the bike sharing rebalancing problem. *Computers & Operations Research*, 71:149–162.

DeMaio, P. (2008). The Bike-Sharing Phenomen. *Carbusters*, page 12.

EcoBici (2016). What's ECOBICI? `https://www.ecobici.cdmx.gob.mx/en/service-information/what%20is%20ecobici`. (Accessed: 2017-September-20).

ESCI (2016). Hangzhou Public Bicycle. `https://esci-ksp.org/project/hangzhou-public-bicycle/?task_id=651`. (Accessed: 2017-September-20).

Espegren, H. M. and Kristianslund, J. (2015). Modeling the Static Bicycle Repositioning Problem. Technical report, Industrial Economics and Technology Management, NTNU.

Espegren, H. M. and Kristianslund, J. (2016). Optimal Repositioning in Bike Sharing Systems. Technical report, Industrial Economics and Technology Management, NTNU.

European Commission (2017). The Digital Economy and Society Index country profile.

Fricker, C. and Gast, N. *Euro journal on transportation and logistics = , volume=5, number=3, Pages = 261–291, Title = Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity, Year = 2014*.

Ghosh, S., Trick, M., and Varakantham, P. (2016). Robust Repositioning to Counter Unpredictable Demand in Bike Sharing Systems. Technical report, Institutional Knowledge at Singapore Management University.

Ghosh, S., Varakantham, P., Adulyasak, Y., and Jaillet, P. (2015). Dynamic redeployment to counter congestion or starvation in vehicle sharing systems. *Twenty-Fifth International Conference on Automated Planning and Scheduling*, pages 79–87.

Gleditsch, M. D. and Hagen, K. (2017). A Dynamic Rebalancing Model for Bike Sharing Systems. Technical report, Norwegian University of Science and Technology, Department of Industrial Economics and Technology Management.

Goodyear, S. (2015). The Bike-Share Boom. `www.citylab.com/city-makers-connections/bike-share`. (Accessed: 2017-September-10).

Gray, A. (2017). China's 'Uber for bikes' model is going global. `https://www.weforum.org/agenda/2017/06/`

china-leads-the-world-in-bike-sharing-and-now-its-uber-for-bikes-model-is-going
(Accessed: 2017-September-16).

Guedes, P. C. and Borenstein, D. (2015). Column generation based heuristic framework for the multiple-depot vehicle type scheduling problem. *Computers & Industrial Engineering*, 90:361–370.

Ho, S. C. and Szeto, W. (2014). Solving a static repositioning problem in bike-sharing systems using iterated tabu search. *Transportation Research Part E: Logistics and Transportation Review*, 69:180–198.

Ho, S. C. and Szeto, W. (2017). A hybrid large neighborhood search for the static multi-vehicle bike-repositioning problem. *Transportation Research Part B: Methodological*, 95:340–363.

Ho, S. C. and Szeto, W. Y. (2016). Grasp with path relinking for the selective pickup and delivery problem. *Expert Systems with Applications*, 51:14–25.

Hughes, C. (2017). Bike Share: The Dawn of the Smartbike (and the Death of Dock-Blocking). https://medium.com/social-bicycles/bikeshare-the-dawn-of-the-smartbike-and-the-death-of-dock-blocking-9f52bb642ae. (Accessed: 2017-September-20).

Høgåsen-Hallesby, J. (2017). What can cycling do for public transport?. Interview with CTO of UIP. https://medium.com/@urbansharing/what-can-cycling-do-for-public-transport-71b7ef751502. (Accessed: 2017-September-22).

Mahvash, B., Awasthi, A., and Chauhan, S. (2015). A column generation based heuristic for the capacitated vehicle routing problem with three-dimensional loading constraints. *IFAC-PapersOnLine*, 48(3):448–453.

Meddin, R. (2018). The Bike-Sharing World map. www.bikesharingmap.com. (Accessed: 2018-June-01).

Mes, M. R. and Rivera, A. P. (2017). Approximate dynamic programming by practical examples. In *Markov Decision Processes in Practice*, pages 63–101. Springer.

Moungla, N. T., Létocart, L., and Nagih, A. (2010). Solutions diversification in a column generation algorithm. *Algorithmic Operations Research*, 5(2):86–95.

Oslo Bysykkel (2017). About Oslo Bysykkel. Availableat:https://oslobysykkel.no/en/about. (Accessed: 2017-September-22).

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450.

Pfrommer, J., Warrington, J., and Schildbach, G. (2014). Dynamic vehicle redistribution and online price incentives in shared mobility systems. *IEEE Transactions on Intelligent Transportation Systems*, 15(4):1567–1578.

Pinto, T., Alves, C., and de Carvalho, J. V. (2018). Column generation based primal heuristics for routing and loading problems. *Electronic Notes in Discrete Mathematics*, 64:135–144.

Popova, M. (2016). EcoBici: Mexico City's Ambitious Bike-Sharing Program. http://bigthink.com/design-for-good/ecobici-mexico-citys-ambitious-bike-sharing-program. (Accessed: 2017-September-20).

Powell, W. B. (2014). Clearing the jungle of stochastic optimization. In *Bridging Data and Decisions*, pages 109–137. Informs.

Rainer-Harbach, M., Papazek, P., Hu, B., and Raidl, G. R. (2013). Balancing bicycle sharing systems: A variable neighborhood search approach. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 121–132. Springer.

Shui, C. and Szeto, W. (2017). Dynamic green bike repositioning problem - a hybrid rolling horizon artificial bee colony algorithm approach. *Transportation Research Part D: Transport and Environment*.

Singla, A., Santoni, M., Bartók, G., Mukerji, P., Meenen, M., and Krause, A. (2015). Incentivizing users for balancing bike sharing systems. In *AAAI*, pages 723–729.

Social Bicycles. Worldwide Rides. http://socialbicycles.com/. (Accessed: 2017-September-14).

Szeto, W., Liu, Y., and Ho, S. C. (2016). Chemical reaction optimization for solving a static bike repositioning problem. *Transportation research part D: transport and environment*, 47:104–135.

Teetor, P. (2011). *R cookbook*. O'Reilly Media, Sebastopol, United States of America.

The Pennsylvania State University (2018). Hypothesis Testing (P-value approach). https://newonlinecourses.science.psu.edu/statprogram/node/138/. (Accessed: 2018-May-28).

Urban Infrastructure Partners (2017a). Urban Sharing. http://urbansharing.com/about. (Accessed: 2017-September-22).

Urban Infrastructure Partners (2017b). Using Data to Optimize Resources. https://medium.com/@urbansharing/using-data-to-optimize-resources-c6f7b7f48328. (Accessed: 2017-September-16).

Van der Zee, R. (2016). How this Amsterdam inventor gave bike-sharing to the world. https://www.theguardian.com/cities/2016/apr/26/story-cities-amsterdam-bike-share-scheme. (Accessed: 2017-September-26).

Venkateshan, P. and Mathur, K. (2011). An efficient column-generation-based algorithm for solving a pickup-and-delivery problem. *Computers & Operations Research*, 38(12):1647–1655.

Victoria, J. F., Afsar, H. M., and Prins, C. (2016). Column generation based heuristic for the vehicle routing problem with time-dependent demand. *IFAC-PapersOnLine*, 49(12):526–531.

Walpole, R. E., Myers, R. H., Myers, S. L., and Ye, K. (2007). *Probability Statistics for Engineers Scientists*. Pearson Prentice Hall, London, UK, 8 edition.