Multi-Modal Last-Mile Delivery: Developing Integrated Water- and Land-based Transportation Systems for City Logistics

by

L.C. Brockhoff

Graduation Assignment

in partial fulfilment or the requirements for the degree of

Master of Science
in Mechanical Engineering

at the Department Maritime and Transport Technology of Faculty Mechanical, Maritime and Materials Engineering of Delft University of Technology,

Student number:	4492528	
MSc track:	Multi-Machine Engineering	
Report number:	2022.MME.8953	
Supervisors:	Dr. B. Atasoy,	TU Delft
	Ir. C. Karademir,	TU Delft
	Ir. M.W. Ludema,	Municipality of Amsterdam

TU Delft

Acknowledgements

This report is written for the graduation assignment in the Master of Science program in Mechanical Engineering, specialising in Multi-Machine Engineering at Delft University of Technology. I have enjoyed working on this project, especially the modelling part, which I see as solving big puzzles. I hope this report effectively reviews all the work I have done.

I would like to express my gratitude to Çigdem Karademir for her always very detailed feedback and comments. It has been great to witness her knowledge about modelling Integrated Water- and Land-Based Transportation systems. Her expertise has been very important to me, as she understands the complexities involved in the modelling and knows how to tackle, and when not to tackle, these challenges. Additionally, her immediate assistance when I had problems with the server prevented many delays. I admire her hard work and her constant readiness to help.

I am also very grateful to Dr. Bilge Atasoy for always guiding me in the right direction and knowing when to steer me towards other parts of the project. Her ability to ask the right questions made me reconsider my approach when necessary. Moreover, she helped me recognise when my work was sufficient and it was time to move on. Furthermore, I want to thank her for all of the pleasant meetings, which I always left with a positive feeling.

I want to acknowledge Marcel Ludema for continuing to guide me as a supervisor from the municipality, even after his time there was over. His focus on practical applications was invaluable, especially since practical considerations can sometimes be overshadowed when I focused on the model. Even though I was not always able to read all the comments, I appreciate the insights and your taking the time to provide feedback despite not being obligated to do so.

Furthermore, I want to thank my friends and family for accepting my absence during the peak moments of my graduation. Your understanding and support were very important, and you helped me maintain a balance by reminding me to take breaks and do things besides work and study. A special thanks to my mother and sister for telling me when to stop.

Lastly, I want to thank Javier, who is the person making sure I eat and sleep during periods of hard work. I hope you continue to cook for me even though I have more time now. Of course, I should not forget to give a special shout-out to Cooper for literally being by my side every moment of writing this report.

I hope my work meets the expectations and that the results prove useful for the municipality. I look forward to seeing the IWLT system for Horeca supply implemented. Additionally, I will provide my model in an accessible manner for the municipality, enabling further investigation into the system scenarios they have in mind.

Thank you all for your support and guidance throughout this project.

Summary

This research addresses the negative consequences of increasing urban traffic in city centres by developing a decision model for integrated water and land-based transportation (IWLT) systems. The motivation behind this study is to shift a portion of transport from roads to waterways to alleviate urban traffic congestion. Implementing IWLT systems is challenging due to the numerous design decisions required. A decision model has been developed to assist in the decision-making process. The model is complex due to the required synchronisation of the transportation nodes. The problem is defined as a two-echelon multi-trip location routing problem with satellite synchronisation (2E-MTLRP-SS), incorporating capacitated vehicles, multiple depots and a global time window. A decomposition-based decision model is introduced, breaking down the problem into manageable sub-problems interconnected through synchronisation in time, space, and load. The decision model uses metaheuristics to be able to handle large-scale, realistic problems and provide feasible solutions for real-life applications.

The research is conducted in collaboration with the municipality of Amsterdam, and the model's effectiveness is demonstrated through a case study in Amsterdam, supplying the city's Horeca (hotels, restaurants, and cafes), showing the potential of IWLT systems to reduce urban traffic and its negative aftereffects. Different scenarios for the IWLT system are investigated, to assist Amsterdam's system developers in making design choices for implementation. The proposed decision model is widely applicable to multi-modal transportation systems worldwide.

The total case for the entire city centre of Amsterdam contains 3 vessel depots, 5 road vehicle depots, 56 potential satellite locations and 1635 Horeca locations, of which the number of locations with demand varies per demand set. Since this is a large problem, a smaller test case is created to quickly investigate some scenarios and analyse the model's sensitivity. A busy neighbourhood, the Wallen, containing 345 Horeca locations is chosen, which is approximately 21% of the total case.

In this case study, vessels transport the supply from depots outside of the city centre satellites, which are transshipment locations in the city centre, where load is transferred between water and road vehicles. Due to limited space in the city centre, satellites do not have storage facilities. The objective is to minimise the distance travelled on roads while using as few vehicles as possible to ensure the system's feasibility for real-life applications.

The modelling strategy involves decomposing the problem into a facility location problem (FLP) and two separate vehicle routing problems (VRPs) for water and road transport, incorporating integration and synchronisation. Finally, scheduling models are used to enable multiple trips and reduce the number of vehicles required. A combination of heuristic and exact methods is employed to achieve high-quality results in a reasonable time.

Various experiments are conducted to assess the performance of the IWLT system for different system scenarios and the sensitivity of the model.

The first experiments focus on the computation time allowed for the model. These experiments determine the required computation time for the remaining experiments, to strike a balance between computation time and solution quality.

Next, two strategies to limit the number of customers assigned to a satellite are evaluated. The first method is to set a maximum number of customers that can be assigned to a satellite straightforwardly. The second method limits the throughput of a satellite. Experiments indicate that allowing more customers to be assigned to satellites results in fewer kilometres travelled on the roads, and a factor of $b = 1.5$ times the evenly divided number of customers per satellite provided a balance between optimal

assignments and even distribution of satellite utilisation, minimising urban disruption.

Deciding on the number of satellites to effectively cover the demand is crucial in designing an IWLT system. Fewer satellites mean satellites are used intensively and potentially create a nuisance for city residents. Understanding how the number of satellites affects road and water kilometres provides valuable insights for informed decision-making in system design. The best performing number of satellites was found to be between 9 and 13 for supplying the entire Horeca sector in Amsterdam. Beyond 13 satellites, the system performance declined due to sub-optimal customer assignments and increased vehicle travel. Experiments with smaller customer sets indicated that the optimal number of satellites decreased linearly with the total demand. For the Wallen neighbourhood, fewer satellites (2-4) performed most efficient, considering different demand sets.

Furthermore, the impact of the available time period on the system requirements is investigated. The time span in which the deliveries are performed is important for the IWLT system to be feasible in real-life applications. Extending the maximum time span for transshipment operations yields a significant reduction in required vehicles. Longer time spans enable vessels to perform multiple trips, thereby alleviating peak loads on road vehicles and ultimately reducing the overall number of vehicles needed. This decrease in road vehicles correlates with a reduction in total distance travelled on the roads, since this includes the distance travelled from road vehicle depots. Fewer road vehicles means fewer vehicles have to travel from depots to satellites. The distance travelled on the waterways remains unchanged for longer time spans, as all vessel trips originate from the same depot.

The analysis of various storage scenarios at satellites for both the Wallen neighbourhood and the entire city centre reveals several key insights. Introducing storage capacity at satellites significantly reduces the required number of vessels, with a 25% reduction observed for $15\mathrm{m}^3$ storage at selected satellites for the entire city centre. These findings suggest that having some storage available provides sufficient flexibility for the system to operate more efficiently. While increasing the storage capacity can further improve performance, the marginal gains become less significant beyond a certain point.

In the IWLT system under consideration, the vessel depots are located quite far from the city centre, leading to long travel distances to and from the depots, which in turn results in prolonged travel times for the vessels. The effect of placing depots closer to the city centre was investigated, showing a 27% reduction in waterway distance and a 33% reduction in the number of vessels required.

The performance of the IWLT system is also highly dependent on the capacity of the road vehicles. Smaller capacities necessitate more trips, thereby increasing both the distance travelled on the roads and the number of road vehicles needed. Experiments indicate a substantial reduction in road vehicles when increasing capacity from $5\mathrm{m}^3$ to $10\mathrm{m}^3$, with further improvements observed up to $15\mathrm{m}^3$. Additional increases in capacity continue to reduce the number of road vehicles but offer diminishing returns.

Sensitivity analyses are conducted to examine the system's response to different parameters and conditions. First, the behaviour under different demand sets is investigated. The experiments involved creating extreme demand scenarios to test the system's adaptability, alongside basic demand sets. The results indicated a near-linear increase in the required number of vehicles with the demand set size. Additional experiments were conducted to better understand the impact of demand characteristics, focusing on the Wallen case. These experiments revealed that the relationship between total demand and distances/number of vehicles is nearly linear, while the influence of the number of customers on these metrics is less significant.

Given the potential for variability and uncertainty in the transshipment processes at customers, conducting a sensitivity analysis of this parameter is important. The sensitivity analysis involves testing the IWLT system requirements under different transshipment times at customers. The system shows resilience up to a point, accommodating increased transshipment times without a proportional increase in vehicle requirements. Increasing the transshipment time from that point, a linear relation with the number of vehicles is indicated.

Based on the experimental analysis, it is essential to evaluate how the IWLT system performs compared to the current situation. Leveraging insights from the experiments, four system scenarios are selected to assess performance, identify bottlenecks, and compare the results with the current state. The scenarios for key design choices are combined to create four distinct scenarios: the expected lowest-performing plausible scenario (A), a baseline realistic scenario (B), an enhanced realistic scenario (C), and the expected best-performing scenario (D). The IWLT system scenarios result in vehicle kilometres reductions of 22%, 24%, 27% and 28% compared to the current situation, for scenario A, B, C and D, respectively. These reductions are a positive step, but the primary goal of the IWLT system is to minimise distance on the roads. All three scenarios accomplish this goal with substantial reductions, 70% for scenario A, 71% for scenario B and 72% for scenario B and C, signifying major improvements over the current situation.

The developed model demonstrates the capability to handle large-scale logistical challenges and provide practical solutions. It offers valuable insights for logistics providers and system designers, supporting the development of IWLT systems.

The results from the Amsterdam case offer realistic estimates for vehicle requirements, suggesting the feasibility of implementing the IWLT system in the city. Additionally, they highlight the potential of utilising waterways to alleviate urban traffic and its associated impacts.

Contents

1 Introduction

1.1. Societal Relevance

More and more people are living in urban areas, and the percentage of the population living in cities keeps growing (Ritchie, 2018). All these people need food and beverages, their waste must be collected, and many must commute. While at the same time, e-commerce is rapidly expanding (Huang et al., 2018). This together results in a growing number of vehicles in urban areas, which has, among other things, a negative impact on the quality of life in cities (Daggers & Heidenreich, 2013). This increase in urban traffic has many consequences for city residents and beyond.

Increased urban traffic results in more urban **road congestion**. Most cities were not designed with this amount of traffic in mind, old city centres often have many one-way streets and few parking spaces. This reduces the traffic flow and can result in traffic jams, for example, when a truck is unloading on a one-way street without available parking spaces. Road congestion results in service delays, traffic idling times, more pollution and stress for the city's citizens (Bull et al., 2004). Not only private cars and freight transport is delayed by congestion, but also public transport like busses and trams suffer from it, which decreases passenger mobility (Bull et al., 2004).

Another important consequence of more urban traffic is the increase in **air pollution and greenhouse gas emissions**. Air pollution is known to have negative effects on human health, such as respiratory problems and cardiovascular disease (Organization, 2022). Moreover, the emission of CO2 and NOx by urban road transportation contributes to climate change. While urban freight transport only represents about 10-15% of the vehicles-km in urban areas, it is responsible for 19% of energy consumption of road transportation, 25% of CO2 emissions, 30% of NOx emissions, and 50% of particles (Janjevic & Ndiaye, 2014). Therefore, improving urban logistics could significantly reduce air pollution and greenhouse gas emissions.

Next to these obvious consequences of increasing urban traffic, some other societal issues arise. The constant **busyness in the streets** causes a nuisance to city residents. Many citizens are bothered by the noise produced by the traffic, just like the visual intrusion and loss of city character (Demir et al., 2015). Next to this, the growing number of vehicles in the streets leads to more accidents (Demir et al., 2015) and therefore **less safety**. Road congestion exacerbates busyness in the streets and, therefore, also the consequences of busyness.

Some cities suffer from extra consequences caused by urban traffic. For example, the city of Amsterdam is dealing with **damage** to its quay walls, connected to the repetitive load of heavy trucks and other traffic (Cordaan et al., n.d.). The quay walls and bridges connect neighbourhoods, and the canals give the city character. Hundreds of bridges and 200km of quay walls are in bad condition. Restoration is costly and results in road blockages in the city centre (Gemeente Amsterdam, 2020). This further increases congestion in the city centre.

Figure 1.1 gives an overview of the consequences of increased urban traffic introduced in this chapter. On the left is the problem itself, and the middle column shows the consequences mentioned before. The dashed lines connect these consequences with their aftereffects. The figure categorises the aftereffects as environmental, societal and economic. As mentioned before, many of the consequences and their aftereffects have a reinforcing effect on each other. For example, less safety leads to more accidents, accidents cause road blockages and road blockages cause congestion. Congestion increases busyness and, therefore, also increases noise and visual intrusion. This is just one example of the reinforcing effect of the consequences on each other, but most consequences are interconnected in ways like this.

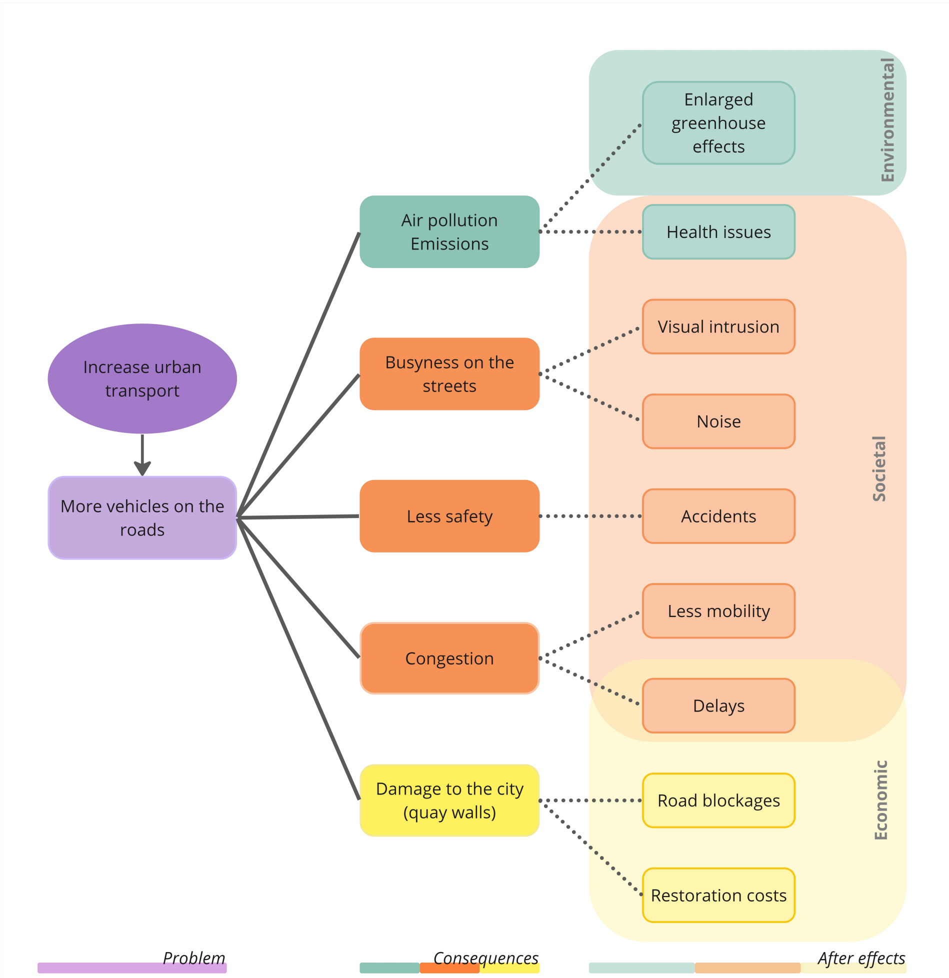


Figure 1.1: Impact of urban traffic

This study is motivated by the various negative consequences of increased urban traffic on the quality of life inside cities (Daggers & Heidenreich, 2013) (Benjelloun et al., 2010) ("Towards sustainable urban distribution using city canals: the case of Amsterdam", 2017) and investigates methods to reduce the effect of urban freight logistics. Change is needed to reduce urban traffic since the quality of life in cities keeps worsening. It is important to find solutions to the core of the problem, increased urban traffic, instead of mitigating the consequences. Therefore, this paper identifies a possible innovative solution for urban logistic systems and investigates its implementation.

1.2. Current Research Gaps

The consequences of increasing urban traffic outlined in the introduction create a need for improvement and innovation in city logistics. To encourage development in city logistics projects, the European Union initiated the mission to make 100 European cities climate-neutral and smart by 2030 (European Union, 2021). A smart city is a city that uses technology and policies to improve its community (Lehr, 2017). Part of the smart city goal is implementing smart urban logistics to achieve more efficient urban logistic systems by intelligent and optimised solutions (Büyüközkan & Ilıcak, 2022).

The initiative of the European Union provides the 100 European cities with resources to reach their goals. These resources include, among others, funding, research and exchange of experiences (European Union, 2021). This initiative encourages cities to innovate and helps the cities reach the goal of being climate-neutral by 2030. However, the initiative is only a way to push cities in the right direction, it is still up to the cities to develop and implement city logistics projects.

Some current city logistics projects only mitigate the consequences of the increasing urban traffic. These projects include the use of electric vehicles to reduce emissions, weight restrictions on vehicles to reduce damage to the city, or time restrictions for supplying stores to reduce busyness and congestion during the day. These projects can help improve the quality of life in cities but only solve part of the problem since the same road transport activities are performed in other time-frames or by other vehicles. These projects do not tackle the core of the problem, namely the burden on the existing road infrastructure, leading to congestion and related issues.

The case of mitigating the consequences of increased urban traffic is also happening in Amsterdam. Many new policies and regulations are being instated, like the maximum weight of heavy vehicles in the city centre is set to 30 tonnes and the length has to be less than 10 meters (City of Amsterdam, n.d.), which reduces the burden on the quay walls. Also, the city centre has been a low-emission zone since 2020, and in 2028, it will even be a zero-emission zone for logistics, together with the city centres of 40 other Dutch cities. Starting in 2025, all new delivery vans and registered lorries need to be zero-emission to be allowed to enter the zero-emission zone for city logistics ("Amsterdam Emission Zones", n.d.). These regulations help to reduce air pollution, emissions and damage to the quay walls. However, they do not reduce the number of vehicles that travel the roads in the city centre. The regulations should compel stakeholders to design more efficient systems and modernise their fleet (Dablanc & Montenon, 2015). When designing logistic systems, the regulations in cities need to be considered since they further limit the feasible choices, especially compared to traditional fossil-fueled vehicles. From these policies that mitigate consequences, it can be seen that policymakers lack knowledge about the alternatives.

One way to reduce the increase of urban traffic itself is to shift modality or integrate different modalities. Alternative modalities for roads could be railways, waterways, underground or through the air. However, some of these modalities need integration to reach all customers in cities.

Within city centres, railways can only reach a few predetermined stations and, because of this, cannot solve the whole problem of urban traffic. Underground transportation could be an option since many large cities have an extensive metro network with many stations. However, transferring freight to and from the underground network would be a difficult and time-consuming activity (Daduna, 2019). Furthermore, the underground network is often occupied by public transport (Daduna, 2019), so additional platforms must be constructed to minimise the effect on public transportation.

Researchers pay significant attention to the use of uncrewed aerial vehicles like drones, which have the potential to solve part of the problem. However, the reach and capacity are currently insufficient to take the pressure off the roads (Moshref-Javadi et al., 2020).

Many cities with waterways running through them (which are a lot, since in earlier times, rivers were an important factor in the location of settlements) could include waterways in the infrastructure (Wojewódzka-Król & Rolbiecki, 2019). The capacity of inland waterways is currently underused, due to the greater preference of the roads by the logistics service providers. Transport using inland waterways has the lowest external costs in terms of emissions, noise, accidents and bottlenecks (Economic & Committee, 2014) compared to other modes of transport. Waterway transport is also economical and has less social impact (Divieso et al., 2021). However, despite the advantages, waterways are not often implemented in city logistics yet (Economic & Committee, 2014), due to limited knowledge on the operations and expensive investment and transshipment costs to integrate waterways into city logistics.

One initiative that is looking into the use of waterways is the TRiLOGy project. It aims to "unlock the potential of transportation and logistics in urban waterways with electric and autonomous vessels by enabling safer, more sustainable and efficient operations." (TRiLOGy, n.d.). This project is a collaboration between companies, the municipality of Amsterdam, Delft University of Technology and the Massachusetts Institute of Technology, which brings together interdisciplinary methodologies. Two case studies are considered: city logistics for transportation and mobility on demand.

Despite the growing interest on multi-modal or integrated solutions, most applications are on a small scale (Wojewódzka-Król & Rolbiecki, 2019). This is mainly because large-scale implementations would mean large investments and require significant research to model these networks. Fortunately, interest in the use of waterways is growing. More research is conducted on implementation, and the potential of currently underused waterways is visible (Janjevic & Ndiaye, 2014). Some challenges must be overcome to realise more efficient use of inland waterway capacity.

Firstly, the alternatives to current transportation modes should be widely known, and methods for implementation should be clear. Specific requirements for every case make it hard to find a suitable solution, no one size fits all solution exists (Jandl, 2016). Many design decisions have to be made to determine an efficient system. Easily accessible and structured information about the possible systems, including waterways in city logistics, is needed.

Secondly, due to high investment costs and expensive transshipment operations, service providers do not prefer transportation systems with multiple modalities. To make such a system profitable and encourage service providers to make a shift in modalities, efficient use of resources and collaboration between the modalities is necessary. It is therefore important to design methods to model the system and evaluate its performance and logistics costs (Groothedde et al., 2005). Moreover, models covering the entire system can assist service providers in assessing the effects of an integrated network (Caris et al., 2014) and making design choices for implementation.

Lastly, the connection between research, policymakers and service providers is missing, a significant gap exists between research and practice (Van Duin & Quak, 2007). To close this gap, possible real-life systems including waterways in city logistics should be connected to available research on models and solving methods.

To tackle these challenges, service providers, system designers, and policymakers need guidance in developing IWLT systems for transition from current logistics systems. There is a need to bridge the gap between the research and real-life applications, especially regarding the decision-making process to design IWLT systems.

Policymakers from the municipality of Amsterdam have noticed the potential use of waterways for city logistics to supply Horeca. However, more information is needed to prove the feasibility of such logistic systems. This research is conducted in collaboration with the municipality of Amsterdam. Before waterways can be implemented in the city logistics, there is a need to investigate the trade-offs, system requirements and design choices for a feasible system and guidance in the development process.

To provide these insights, the main research question of this thesis is:

What is the potential of integrated water- and land-based inland transportation systems to improve city logistics towards liveable cities?

The answer guides the system designers in the developing and decision-making processes. This will enable a more accessible and easier transition from current transportation systems to integrated water- and land-based transportation systems.

One of the challenges to develop integrated water- and land-based inland transportation systems for city logistics is the absence of know-how. Even if service providers or policymakers would like to use waterways in city logistics, it is hard to know how to do so since the existing knowledge is mostly about road transportation or simplified small-scale use cases for freight logistics over waterways in cities. Current research does not often address the process of decision-making. Mostly, a system is defined, and solution methods are developed without discussing other optional systems and the design choices. It is therefore hard for a system designer to determine which system would be suitable for its desired application.

Many design choices at the strategic and tactical levels need to be made to develop an integrated water- and land-based inland transportation system. This makes it difficult to develop such transportation systems since some of these choices are hard to make with limited prior knowledge and limited known quantitative approaches. The municipality of Amsterdam currently encounters this challenge in the development process for an integrated water- and land-based transportation system. It is useful to simplify the decision-making process to assist the municipality of Amsterdam and encourage more companies and municipalities to implement waterways in city logistics. It would be very beneficial to have a decision support model that enables users to test different design choices for service network design and determine the system requirements.

Unfortunately, currently, no known solution method is available that covers all/most of the system possibilities, which means multiple solving methods should be tried out to compare the results of making different decisions. This is not only a time-consuming activity but also requires expertise that might not always be available within the team that wants to implement waterways in city logistics.

However, some challenges exist for a decision model that covers all system options. Such a model will have a large computation time since the number of possible solutions grows exponentially with the number of attributes and the size of the instances (Vidal et al., 2020). Therefore, the more realistic system options are covered, the longer the computation time, and the smaller the problem instances that are solved. There is a need to develop efficient solution methods to solve larger instances within highly integrated systems.

Another method to simplify the decision-making process is to provide a model that can be modified with respect to different options including their extra limitations or flexibilities. Then, the resulting system alternatives can be evaluated based on the trade-offs if multiple goals exist. This tool would not have to run one decision model to cover all system options but can evaluate the results of separate decision models. This way, the computational burden is less, since the computation time of the individual decision models is added up, instead of the exponential growth that would come from adding more decision variables to one decision model.

This tool needs to provide the possibility to enable/disable some system options, to adjust the model to specific problems and allow for some design choices to be made a priori. A tool that evaluates all system variations would eliminate a hard part of the process of developing IWLT systems and, therefore, encourage more logistics service providers and municipalities to implement waterways in city logistics.

1.3. Research Approach

This research aims to investigate the potential of integrated water- and land-based transportation systems for city logistics, guided by the design of a decision-support model for these systems. This decision support model should specifically help the municipality of Amsterdam to confirm the feasibility of an IWLT system for the city and give clear indications for the system requirements. To reach this goal, a combination of research methods is necessary, including literature research, data analysis, model development, simulation experiments, analysis of the results and evaluation.

The decision support model of this project is developed with the general problem definition of an integrated water- and land-based transportation system in mind. The model is specifically designed for the case of the city of Amsterdam, however, with some adjustments it is widely applicable for similar problems.

Since the research question is complex, sub-questions are formulated to guide the search to answer the main question. These sub-questions help identify specific aspects of developing IWLT systems that must be addressed to answer the main questions. Each sub-question addresses a research phase to develop a decision model for IWLT systems.

The first sub-question aims to determine significant system options for IWLT systems. Next, information is gathered about the possible system types, design choices, objectives and trade-offs for these systems. Data collection and analysis are performed through literature review, desk research, and expert interviews. The sub-question one is formulated as: *What are the significant design choices for developing integrated water- and land-based transportation systems?*

After the information about IWLT system options and design choices is collected, the focus shifts to the model development, for which the sub-question two is formulated. *What decision models for multi-modal transportation systems exist?* Knowledge about existing decision models will help in developing a decision support model for the IWLT system of this research.

With the available solution methods for IWLT systems known, the approach for the specific system in Amsterdam has to be determined. Sub-question three aims to identify the important aspects of this approach. *How to develop decision models for integrated water- and land-based transportation systems that allow to solve full-scale realistic problems?* The answer to this question investigates suitable options for the system and how the problem could be decomposed. This is done by combining the literature research with data collection and model development. First, the scope and goals of the decision model are determined, next, the specific problem definition is given, after which the modelling approach is described.

After this, the model is developed and described in Chapter 4. This chapter gives the mathematical formulation of the solution models for the sub-problems defined by the third sub-question.

With the developed model for different IWLT system options, it is important to validate the model approach used and thoroughly evaluate the results for different system scenarios. The last sub-question, sub-question four directs to results analysis and evaluation assisted by the case study for Amsterdam. *What is the performance of the proposed integrated water- and land-based transportation system under different scenarios of interest?* Answering this question also provides the municipality of Amsterdam with recommendations for implementing the IWLT system.

1.4. Research Contribution

This research focuses on bridging the gap between research and real-life applications by developing a decision-support model for IWLT systems. This model helps to make design choices in developing real-life applications of IWLT systems, making it more accessible to evaluate the effect of different scenarios on the system requirements.

Many models for multi-modal transportation systems exist, but most focus on one specific case where the design choices are already made, or consider only a few system options. This research models the IWLT system in a manner that allows testing different scenarios for the design choices.

Furthermore, the existing models for multi-modal transportation systems that include a high level of synchronisation do not apply to large-scale realistic problems. This research uses a decomposition-based modelling approach to tackle large-scale IWLT problems for different system decisions under synchronisation constraints. The specific decomposition used in this research is widely applicable to multi-modal transportation problems. It provides insights into the performance of these multi-modal transportation systems under different system designs.

The decision model is used for the case of Amsterdam, to provide insights and recommendations for policymakers about the system decisions and requirements. At the same time, this case helps to validate the decomposition approach used.

1.5. Thesis outline

The remainder of this thesis is structured as follows. In Chapter 2, literature is reviewed to answer the first two sub-questions. First, design choices for the development of IWLT systems are investigated, answering sub-question one. Second, current state-of-the-art decision models for multi-modal transportation systems are discussed, focusing on sub-question two. Chapter 3 outlines the modelling methodology employed in this thesis, including the problem definition and the approach taken, which answers sub-question three. In Chapter 4, details the models for each of the sub-problems defined in the modelling methodology. Chapter 5 first introduces the case study for the city of Amsterdam. Next, experiments are conducted for different system scenarios and model settings. The results are evaluated to answer sub-question four. Finally, Chapter 6 answers the main research question and provides recommendations for practice and future research.

2
Literature study

As highlighted in the introduction, there is a growing interest in utilising waterways for city logistics due to their potential benefits, such as reducing truck movements and emissions. However, implementing large-scale city logistics on inland waterways requires significant time and effort. There is no standardized step-by-step plan, forcing each company or municipality to develop its own approach. Many design choices must be made, and the system must be thoroughly modelled to ensure efficiency. Additionally, the implementation costs can be high. Consequently, the current use of waterways in city logistics falls short of its potential. Despite these challenges, the urgency for a modal shift in urban transportation is increasing, driven by growing city populations, resulting in the need for alternatives to road transportation. The alternatives for road transportation are investigated in this chapter.

Furthermore, this chapter answers two sub-questions. The first question answered is: *What are the significant design choices for developing integrated water- and land-based transportation systems?*. Answering this question helps to identify alternative logistics systems to traditional road transportation systems. Operational challenges and benefits of waterways in city logistics are analysed, and this chapter provides an overview of practices implemented or tested by different service providers. Different service design choices and their implications are discussed, such as modes of transportation, service type, the storage capabilities at satellites and transfer methods.

Then, the second sub-question is answered: *What decision models for multi-modal transportation systems exist?*. This answer provides information about current state-of-the-art decision models for comparable multi-modal systems. Their significance and implementability for the specific IWLT system of this paper are discussed in the next chapter.

2.1. Multi-Modal Transportation Systems

Before investigating the design choices for integrated water- and land-based transportation systems, it is useful to clarify this definition and explore some of its applications.

A multi-modal transportation system coordinates the use of two or more modes of transport. This can, for example, be trucks and cargo bikes, vans and drones or, as in this research, vessels and light electric freight vehicles. A basic example of a multi-modal transportation system is shown in Figure 2.1. First-mode vehicles are used for transport to satellites from depots. Satellites are transshipment locations where the cargo is transshipped between the transportation modes. From the satellites, second-mode vehicles perform deliveries to customers.

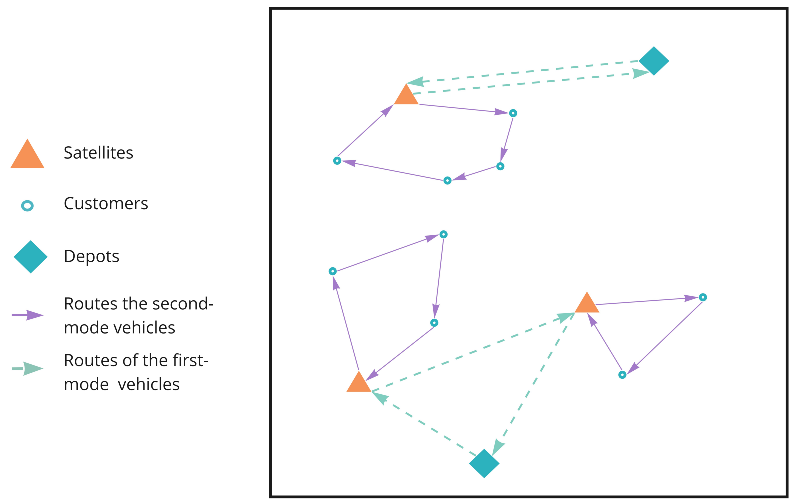


Figure 2.1: Illustrative example of a multi-modal transportation system

Multi-modal systems are already used for most forms of transportation. Usually, cargo is not directly transported from its origin to its final destination by one vehicle. Instead, cargo is first transported from large depots to smaller hubs by large vehicles, like aeroplanes, ships or trucks. Then, smaller vehicles transport the cargo (closer) to their destination. The same is true for passenger transportation. Think about people going on vacation. Often, they first go by aeroplane or train, followed by the use of buses, trams, metros or cars.

The next section briefly highlights various multi-modal transportation systems that do not use waterways, offering an impression of the range of existing options. Following this overview, the discussion shifts to current multi-modal transportation systems that utilise waterways. These systems are examined in more detail as they closely align with the focus of this research, providing relevant examples for the integrated water- and land-based transportation system under investigation.

2.1.1. Multi-Modal Systems without Waterways

Since the problems arising with urban logistics are known, much research is (being) conducted to change last-mile delivery systems. To avoid, for example, congestion in a busy street or to comply with city regulations, many innovative ideas have arisen. This section highlights ideas for multi-modal transportation systems that do not use waterways. Most multi-modal systems aim to reduce the number of (large) trucks in city centres. This is often done by using larger vehicles to transport freight to points within or close to city limits and then using smaller vehicles for last-mile deliveries. By applying this method, the larger vehicles do not have to move as much through the city centres. There has been growing interest in the literature as well as in applications on multi-modal systems using different types of vehicles, summarised as follows:

- Trucks and drones
- Trucks and small vans
- Trucks and cargo bikes
- Busses and rolling containers
- Busses and cargo bikes
- Trains and electric bicycles
- Trams and electric vehicles

2.1.2. Multi-Modal Systems Including Waterways

Next to the innovations in multi-modal systems described above, some companies and cities already use waterways in their transportation systems, and many others are conducting pilots (Wojewódzka-Król & Rolbiecki, 2019). In this section, the different modalities used are investigated.

In Amsterdam, DHL uses a floating distribution centre to deliver and pick up packages in some areas of the city centre. Already, 60% of DHL business in the inner-city areas is delivered by cargo bikes. For part of this business, it uses a vessel of seventeen meters, which replaces about five delivery trucks. Cargo bikes follow the boat's progress to pick up or deliver packages and meet up with the vessel along three regular docking points. The cargo bikes and vessels stay in contact using mobile phones. Switching to the multi-modal system allowed DHL to reduce the number of vans they use in the city centre from ten to two. This results in a reduction of 150.000km every year, saving 12.000 litres of fuel. The same system is used for collections, on the return journey, cargo bikes can collect packages and bring them to the vessel. (Parr & Register of Initiatives in Pedal Powered Logistics, 2017)

Also, in Amsterdam, a pilot is being conducted, using small garbage trucks to pick up garbage in the city centre and bring it to an electric vessel. New regulations in Amsterdam prohibit vehicles weighing more than 7.5tons from entering the city centre (Gemeente Amsterdam, n.d.). If the use of heavier vehicles cannot be avoided, a permit has to be requested. These small garbage trucks remain below this limit, even when filled with waste. The CO2 emission is reduced by more than 90% by using this multi-model system instead of the large garbage trucks that currently collect waste in the city centre (Gemeente Amsterdam, 2021).

Other cities in the Netherlands are implementing waterways in city logistics as well. Utrecht is the owner of the Beerboot. This barge supplies part of the hospitality industry in the inner city. Last-mile deliveries are organised by the 'Cargohopper', an electric vehicle that pulls multiple carts (Mommens & Macharis, 2012). The Beerboot ensures a reduction of CO2 emissions of 94% compared to conventional vans (Maes et al., 2012).

Not only in the Netherlands, waterways are also gaining interest for urban logistics in other places. In Paris, France, multiple applications exist. One example is the company Franprix's use of waterways to supply groceries to stores close to the Eiffel Tower. A barge with 26 containers and 450 pallets sails close to the Eiffel Tower and transfers the products to regular diesel trucks, which take care of the final deliveries (CCNR, 2021) (HAROPA - Ports de Paris, 2012). Further, in Paris, Fludis, a company in the urban logistics sector, provides a decarbonised solution that avoids costly stock-outs in warehouses. On the outbound journey, a fully electric boat delivers office supplies for the company Lyreco, which are transported by bike to their final destinations. On the return journey, the boat collects electronic waste (CCNR, 2021). In another city in France, Strasbourg, Urban Logistics Solutions uses a rental vessel to transport parcels to a platform in the city centre. From the platform, fifteen cargo bikes are used for last-mile transportation. On the return journey, recyclable waste is collected. For now, the vessel only makes one trip daily, but there are plans to increase the frequency. (CCNR, 2021) (Observatoire Régional Transports & Logistique - Grand Est, 2020)

In Ghent, Belgium, an implemented project is the Bioboot. Crops are transported into the city by a small solar-powered vessel directly from the production site. The crops can be picked up at the dock or delivered by bicycle trailers. (CCNR, 2021) (Goededinge.be, 2020)

A project in Berlin, still in its pilot phase, uses small autonomous vessels for transportation between docks out of the city and the inner-city area. From there, self-propelled land-based vans or cargo bikes can be used for final deliveries. The autonomous vessels can sail individually or as a coupled convoy (swarm). (CCNR, 2021) (Technische Universität Berlin, 2022)

While waterway transportation has many promising advantages, some challenges exist. Transshipment is needed between vehicles to implement waterways in city logistics, which can be expensive. Next to this, finding transshipment locations can be difficult in densely populated areas. Dock dues can be high, especially compared to trucks that do not need a transshipment location (CCNR, 2021). The speed of vessels is also lower compared to road vehicles. Another aspect that should be considered is the technical capacity of vessels/barges because this relies on the depth of the water, which can vary by seasonal conditions. Furthermore, clearance under bridges and the width of the waterways must also be considered (Diziain et al., 2014).

These disadvantages have to be outweighed by the advantages of waterway transportation to have a business case for such systems. Many kilometres on the road can be avoided, resulting in less use of fuel and lower emissions. Another significant advantage is reducing the burden on the roads, and thus a reduction of impact by urban traffic as shown in Figure 1.1. To make IWLT systems a viable business, the system has to be designed efficiently. Furthermore, stricter city regulations on road vehicles can encourage the shift of modalities.

2.2. Design Choices

Based upon the previously discussed applications of waterways in city logistics, it is clear that many promising IWLT systems exist, but many design choices have to be made to provide the service efficiently. This section identifies aspects of the IWLT systems on which decisions must be made and provides knowledge about the decision-making process. A lot of the choices depend on the regulations and infrastructure of the city, as well as the goals of the stakeholders. Furthermore, this section discusses design choices that influence system operations and the type of problem to solve at an operational level. Some bounds are specified and distinguished by where they have to be made in the process. The first section gives an overview of the decisions. The second part describes the choices that are affected by city regulations, restrictions and other externalities.

2.2.1. Decisions for a Service Network Design

Some design choices are necessary to determine a suitable problem definition for the services provided. The two most important choices to make are the determination of transfer locations and the transfers method: direct, indirect or unloading of loaded vehicles. The way of executing the transfers immediately narrows the selection of available models down. When there is no storage at the satellites, meaning direct transfers are necessary, considerable synchronisation between the modalities is required. For indirect transfers, storage space has to be available at the satellites, which makes synchronisation in time less significant. The variant of unloading loaded vehicles requires less synchronisation, especially if the unloaded vehicles return to the depot by themselves. The placement of the satellites also plays a large role in selecting a decision model. Different model classes exist for the case of determining the satellite locations and the case where the locations are predetermined. Chapter 4 explains the connections between these design choices and the decision models further.

In some cases, the decision in method and locations of transfers is easily made by the system designer. For example, if the system designer wants to use predetermined docking points with cranes and storage on shore. However, if multiple options have to be investigated, a model that covers the different options is required to find the most suitable system for the application.

Other design choices have less impact on the type of model, but information is still needed to select a model variant. These choices include single— or multi-trip, single— or multi-depot, pick-up/delivery or both, and homogeneous or heterogeneous fleet.

Whether single- or multi-trip options for the vehicles are desired largely depends on the type of vehicles used. For smaller vehicles, like cargo bikes, routes include fewer stops because of the limited capacity. If every cargo bike performs one short route and remains unused for the rest of the time period, a larger number of cargo bikes is needed to perform all deliveries and/or pick-ups. Therefore, it is feasible to include the multiple trips option. This option is less important for larger vehicles since it takes longer to perform one trip, so less time remains unused. However, the multiple-trip option gives the possibility of performing multiple trips, which will only be used if it has a positive effect on the objective.

Some models work with one single depot from which the first-level vehicles depart. If the desired system has multiple depots or it is desired to investigate the option of multiple depots, some models cannot be used.

Pick-up systems can be seen as equal to delivery problems, but with reverse flows. The difference lies in the operations at the satellites (Karademir et al., 2022). However, if pick-ups and deliveries are desired, the capacity of the vehicles is not just decreasing or increasing on a trip, which produces extra algorithmic challenges (Sluijk et al., 2023). Not all models are designed for this problem, so only a selection of models can be used.

The last of these choices is the fleet composition, which is heterogeneous or homogeneous. Information about the vehicle fleet is needed for the models, the paragraph below elaborates on what this information includes. Most models only allow for one type of vehicle per echelon, which is called a homogeneous fleet. Some models have the feature of a heterogeneous fleet, which enables the use of vehicles with different characteristics. This feature is useful when no decisions on the vehicle types are made beforehand, and the options need to be investigated. In this category, it is also important to note the implications of electric vehicles. These vehicles have range limitations, which must be incorporated into the model. This range can be increased by adding battery swapping or recharging stations. (Sluijk et al., 2023)

Besides the previously mentioned design choices that impact the type of model needed, some more input is required to solve the problems on hand. Depending on the model type, this input can include the capacity of the vehicles, cost per km of the vehicles, speed of the vehicles, the daily cost for operating the vehicles, the maximum number of vehicles, maximum distance of vehicles to travel, cost of opening a satellite, the capacity of the satellites, transshipment time, transshipment cost or other specifics. Vehicles indicate both land and water-based vehicles. Depending on the design choices made before and the type of model(s) selected, some of these inputs are not applicable.

The rest of this chapter investigates how the design choices can be made or what bounds can be applied. With these outcomes, decisions about the model type and variants can be made, and inputs are determined. After applying this information to define the system limits, the system can be optimised for the fleet size, the number and locations of satellites and the routes of the vehicles with respect to the objective.

2.2.2. Decisions by Regulations, Restrictions and External Factors

Part of the design choices can be made without the use of models, or at least be bounded. External factors, like city regulations, available space and stakeholders, can place restrictions on the design space. When these external factors only put bounds on the design choices, the design choices should still be implemented in the decision-making process, which means these design choices are variables to be optimised by modelling but have to abide by some bounds. It is helpful to identify bounds since it narrows down the feasible decisions and reduces unexpected infeasibilities or costs. Below, some of the possible bounds are discussed.

First, the most obvious bounds are given by restrictions and regulations. Many cities in Europe have introduced low- or zero-emission zones. In Amsterdam, the city centre has been a low-emission zone since 2020, and in 2028, it will even be a zero-emission zone for logistics, together with the city centres of 40 other Dutch cities. Starting in 2025, all new delivery vans and registered lorries need to be zero-emission to be allowed to enter the zero-emission zone for city logistics (“Amsterdam Emission Zones”, n.d.). Also, the maximum weight of heavy vehicles in the city centre is 30 tonnes, and the length has to be less than 10 meters (City of Amsterdam, n.d.), which reduces the burden on the quay walls. Similar regulations are introduced in many cities to reduce pollution, emissions, noise, congestion and overall improve the quality of life in cities. However, few cities investigated the impact of these restrictions on urban freight transport. Noticeably, the regulations compel stakeholders to design more efficient systems and modernise their fleet (Dablanc & Montenon, 2015). When designing a logistic system, city regulations must be considered since they can put bounds on the design choices, especially with regard to vehicle characteristics.

Furthermore, many municipalities have regulations for operating windows, due to the noise associated with it. These time restrictions influence the system, since smaller time windows to operate require a larger number of vehicles or vehicles with a higher loading capacity. Some cities might also have general noise restrictions in place. Usually, the noise cap will not be violated by transport systems, but they should be considered when deciding on equipment.

The noise associated with transshipment activities can also play a role in determining satellite locations. It might be reasonable to give preference to satellite locations where noise does not affect residents, this can be applied, for example, by assigning costs to opening satellites based on the inconvenience for city residents.

Other external factors, beyond regulations and restrictions, must also be considered. These factors are often tied to a city's existing infrastructure. To minimise costs, it is ideal to leverage as much of the existing infrastructure as possible. However, this infrastructure may not be suitable for all types of vehicles or transfers. Weather conditions are another important consideration. For example, seasonal changes can alter water levels, resulting in varying passage height and width of waterways, imposing constraints on vehicle specifications. Furthermore, extreme weather events such as droughts, floods, or frozen waterways can also occur. While these factors do not impose strict design constraints, their potential impacts should be taken into account.

Ignoring or underestimating these external factors can lead to infeasible or costly operations. Factors restricting maximum vehicle weights, fuel type, accessibility limitations, and labor regulations can significantly affect system profitability. Considering these limitations during the service network design phase can lead to better strategic and tactical decisions.

2.2.3. Decisions through Modelling

The sections above explain how the regulations and restrictions provide some handles to make the design choices, like bounds for some design choices of the logistic system. These bounds can act as constraints for the system, since for most decisions, they only narrow the range and are not enough to make the decision. So, many design choices remain, for which decision models are needed to investigate the best choices. This section will look deeper into these design choices and explain how these can be included in a decision model.

Below, design choices that can be made by modelling the IWLT system are listed. These choices can either be implemented as decision variables in the model or evaluated through experiments with different input scenarios.

- Number of vehicles
- Capacity of vehicles
- Number of satellites
- Locations of satellites
- Storage capacity at satellites
- Operational time span
- Depot locations

Whether the design choice is implemented as a decision variable or evaluated through the system's performance under different design choices depends on the selected model and preferences. For instance, if the number of vehicles to perform deliveries is implemented as a decision variable, the operational time span can be an input parameter, allowing evaluation of the system's performance under varying time spans. Conversely, if the operational time span is a decision variable, the number of vehicles can be a varying input parameter, and the time span required to supply the customers for different numbers of vehicles is evaluated.

It is also possible to implement both the number of vehicles and the operational time span as decision variables, with an objective to minimise both. In this case, importance values must be assigned to each decision variable, determining a ratio between reducing vehicle numbers and operational time span. Alternatively, both the operational time span and the number of vehicles could be input parameters, of which the values are varied to evaluate the system's performance. However, this could render the system infeasible, as too few vehicles might fail to supply all customers within the specified time span.

This example illustrated how design choices interact and can be implemented in a decision model. Some design choices are more complex to determine in advance, making them suitable as decision variables. For instance, if a system designer has a specific fleet available, it might be more relevant to determine the time span needed to supply customers utilising the available fleet. However, if the fleet is not yet determined and there are regulations regarding the time window for transshipment activities, it is useful to treat the number of vehicles as a decision variable while keeping the operational time span as an input parameter.

Next to these design choices, some operational decisions exist, the most important being the routes of the vehicles. These routes are determined using a decision model, commonly with the objective of minimising the distances travelled. The routes do not impact the decision variables directly but contribute to the objectives and affect the number of vehicles required.

The system's objective is important in determining preferred values for these design choices. Different objectives can result in very diverse optimal systems. For example, minimising a system's costs will most likely not minimise its emissions. An example of this trade-off is given in the Waste-On-Water project (Huijgen et al., 2022), where the impact of collecting industrial waste with EVs in combination with vessels is compared with the regular collection by trucks. The impact is evaluated based on CO2, NOx and PM10 emission, km/ton, costs, employment, safety and congestion. The costs are higher for the proposed system, while the emissions and congestion are reduced.

2.2.4. Overview of System Decisions

This section answers the first research sub-question: *What are the significant design choices for developing integrated water- and land-based transportation systems?*. The design choices for IWLT systems and their bounds are discussed. Table 2.1 gives an overview of the design choices and operational decisions discussed in this chapter for an integrated water- and land-based transportation system. The next chapter will dive deeper into how these choices and bounds are used as inputs for the model and how they influence the type of model needed.

Table 2.1: Design choices integrated water- and land-based transportation system

General	Transfers	Water level	Street level
Pick-up/Delivery	Number of satellites	Vehicle characteristics	Vehicle characteristics
Time span	Locations of satellites	Number of vehicles	Number of vehicles
Single-/Multi-Trip	Storage capacity at satellites	Routes	Routes
Single-/Multi-Depot	Transfer method		

2.3. Available Decision Models

The focus of this research is on developing integrated water- and land-based inland transportation systems to improve the quality of life in cities. The previous sections explore the possible systems and the design choices to make for these systems. So, now the possible transportation systems are known, the following sub-question arises: *What decision models for multi-modal transportation systems exist?*. The answer to this question provides insights into current state-of-the-art solution methods for the IWLT systems determined in the previous chapters. In this chapter literature is reviewed to find suitable models for these logistic systems. First, the type of problems to be solved for the systems are defined. Next, the difficulties with integration and synchronisation of the system are pointed out. After that, a deeper search is conducted considering the specifics of each system and what models and algorithms exist to solve them.

2.3.1. Problem Classification

As mentioned, the interest lies in integrated water- and land-based transportation systems, which means the system consists of two separate transportation networks, one over water and one over land. These two networks are connected by intermediate facilities (satellites). So, freight moves through one of the networks, then via an intermediate facility to the second network, to its destination. In literature, such a system is referred to as a two-echelon system, where each echelon refers to one level of the transportation network and these levels are connected by satellites (Cuda et al., 2015). When these two levels use different modes of transport, the two-echelon system is also a multi-modal system. The definition two-echelon is clarified in the problem variations below, but first the main problem classes are explained below.

Routing, Facility and Location Problems

To model the systems, they are connected to common problem types, which are widely addressed in the literature. The multi-modal system with intermediate facilities exists out of multiple parts; routing of the first modality, locating the facilities and routing of the second modality. Below, the basic problems are explained.

Vehicle routing problem | VRP
The vehicle routing problem (VRP) is used to determine the optimal set of routes to serve a given set of customers (Toth, 2002). Thus, the VRP only covers the routing aspect of the system.

Facility location problem | FLP
Facility location problems (FLPs) are used for locating or positioning facilities in order to optimise (minimise or maximise) at least one objective function (like cost, profit, revenue, travel distance, service, waiting time, coverage and market shares) (Farahani et al., 2010).

Location routing problem | LRP
Location routing problems (LRPs) are a combination of the VRP and the FLP (Prodhon & Prins, 2014), that can be used to determine which facilities should be used and the routes of the echelons to these facilities (Schneider & Drexl, 2017).

Piggyback Transportation Problem | PTP
Piggyback Transportation Problems (PTPs), in this context, refer to the problem in which a large vehicle moves batches of small vehicles to a launching point, from where the small vehicles depart to perform last mile deliveries (Wang et al., 2022). This process can be repeated until all shipments are performed. The PTP can also be seen as a variant of the VRP or LRP and the problem can be approached using algorithms for the VRP and LRP.

Modelling the systems could be separated into multiple problems, the routes of the modalities and the locations of satellites. However, since these problems depend on each other, solving them independently can result in sub-optimal planning results (Schneider & Drexl, 2017). This issue is addressed more elaborately in the next sections.

All in all, to model an IWLT system, a few options exist. The satellite locations can be defined with a FLP and the routes by two separate VRPs or they can be combined, which is discussed later. Another possibility is to define the satellite locations using other methods, for example with the use of Geographic Information Systems. Further, LRPs can be used to determine both the satellite locations and the routes of the echelons. In Subsection 2.3.2, the specifics of these problems are investigated regarding different applications and a search is conducted for suitable solution methods.

Variations of the Problems

To handle more realistic applications, the systems can be solved as a variant of VRP, LRP or a mix of them, including several attributes. Many attributes exist and the number of variations is growing rapidly (Vidal et al., 2020). In this paper only the variants essential to the IWLT system are reviewed. Besides the variants explained below, capacitated vehicles are assumed to be standard for the problems, meaning both modalities have vehicles with a maximum storage capacity. Since, if this capacity is ignored in the model, it could lead to solutions that are not feasible in real life.

Two-Echelon | 2E
A supply chain is composed of stages (also called layers or tiers). Transportation occurs between each pair of stages. Each stage represents one level of the distribution network and is usually referred to as an echelon. 2E-LRPs are problems where routes may be present at both echelons and location decisions have to be taken for at least one echelon. 2E-VRPs are problems where no location decisions have to be taken, only routes are determined for both echelons (Cuda et al., 2015). Two echelon problems connect the two distribution networks.

Multi-Trip | MT
The added value of enabling multiple trips is mentioned in Subsection 2.2.1 for accessing the customers in case of city regulations, and depends on the chosen type of vehicles for system. This attribute makes it possible for one vehicle to make multiple trips, for example, a cargo bike can collect its load at a satellite, deliver it all to customers, and then repeat this process.

Time Windows | TW
Time-windows are required by the last users for premium services or city regulations for logistics operations. They limit the systems and require more effort to achieve synchronisation and feasibility.

Satellite Capacity | SC
Real-life satellites have limited storage capacities, which can mean it is not possible to let a vessel deliver everything at once to one satellite and let road vehicles pick it up whenever they arrive. Therefore, the storage capacities of the satellites have to be taken into account in the model for scheduling transfers at the satellites to ensure the capacity of any satellite is not exceeded.

Satellite Synchronisation | SS
It is also possible there is no storage capacity in the satellites at all, synchronisation between the two-echelons is necessary to ensure direct transfers between vessels and street vehicles without leaving cargo at the satellites. Synchronisation can be implemented in different degrees and manners, on which the next section elaborates. Synchronisation is still required when storage is available at the satellites, but it reduces the degree of synchronisation necessary and ultimately the cost of the synchronisation.

Another attribute that is frequently mentioned in literature, is the option for direct services to the customers using only one of the available networks, i.e. only roads or waterways. The benefits of such flexibility depend on the proximity of the customers to the waterways or to the central warehouse.

Finally, split deliveries can be implemented, which means customers can be served from multiple vehicles. This can be interesting for minimising the number of vehicles since each vehicle can be used to its full capacity. However, visiting a customer multiple times can result in more kilometres.

Multiple attributes together can be added to the basic variants. Generally, the more attributes are enabled, the more realistic the model, but also the more complicated the problem is. The number of possible solutions grows exponentially with the number of attributes and exponentially with the size of the instances (Vidal et al., 2020). Therefore, the more realistic the problem is modelled, the longer the computation time to solve it. Because of this, more realistic models are able to solve smaller instances than basic models. This effect becomes visible in the size of the problems solved by algorithms of Subsection 2.3.2.

Integration and Synchronisation

The two-echelon problem can be solved separately in the two echelons, using a sequential approach. The first echelon is solved, and the outputs are used as inputs for the second echelon. However, this approach does not take into account the interdependence of the modalities. Optimising the route of the echelons separately does not automatically result in an optimal solution together, since one or the other ignores the cost of the integration. A two-echelon problem can also be solved as an integrated problem, taking into account the interdependencies between the two-echelons. An integrated approach involves solving both echelons simultaneously, considering the implications of a solution on global optimality. Integrated problem-solving causes a significant increase in the computational burden required, but it provides a better solution than solving each problem separately (Côté et al., 2017). Because of the computational burden, integrated solving of large-scale problems with multiple attributes is not always achievable.

Whether the two-echelon problem is solved in an integrated or sequential way, synchronisation between the two echelons is essential. Synchronisation refers to the coordination between the two echelons. In a sequential approach, synchronisation is achieved by linking the outputs of one echelon to the inputs of the other echelon. In an integrated approach, synchronisation is achieved by considering both echelons simultaneously and optimising the transportation system as a whole. However, this synchronisation needs to be modelled and does not happen spontaneously.

Different types of synchronisation exist, for instance, **temporal synchronisation**, refers to the coordination of the delivery schedules between the two echelons in terms of time, by determining the optimal delivery times for the first echelon, so the delivery is synchronised with the schedules of the second echelon vehicles. This helps to minimise waiting times at the satellites. **Spatial synchronisation** refers to the coordination of the delivery schedules between the two echelons in terms of space. This involves determining the optimal routes for the vehicles in the first echelon so that the deliveries to the satellites are synchronised with the routes of the vehicles in the second echelon. This can help to minimise the transportation costs. Another type of synchronisation is **load (cargo flow) synchronisation** (Drexl, 2012). Load synchronisation is used to ensure sufficient capacity to handle the freight in both echelons. Since it is generally assumed that second-echelon vehicles have a smaller capacity than first-echelon vehicles, with load synchronisation it is guaranteed all freight can be transported through both echelons.

The type of synchronisation needed for the system depends on whether or not there is some storage capacity at the satellites. When no storage is available at the satellites, temporal, spatial and load synchronisation are all essential. The required synchronisation synchronisation is less significant when storage is available at the satellites, as there is a buffer that allows for more flexibility in timing and routing. However, synchronisation is still necessary to avoid overloading the limited storage and to ensure that the flow of goods remains steady and efficient. If satellites had unlimited storage, the need for temporal and spatial synchronisation would be significantly reduced, as goods could be stored indefinitely, allowing more flexibility in scheduling and routing. However, in practical terms, even if storage is substantial, it is rarely unlimited, especially within city limits. Therefore, some degree of synchronisation is still critical to ensure smooth operations and to prevent inefficiencies.

In summary, the degree and type of synchronisation required depend on the storage capacity at the satellites, but some level of synchronisation is always necessary to ensure the efficiency and feasibility of the two-echelon transportation system.

2.3.2. Solution Methods

With the attributes described above, the problem definition for the IWLT system can be determined. Several options must be considered regarding satellite allocations and vehicle types. If the locations of the satellites are known prior to modelling, variations of the 2E-VRP are sufficient. If the satellite locations are not known, variations of the 2E-LRP are more suitable.

However, a selection from a few optional satellite locations can be made using the 2E-VRP by letting the model choose satellites for the transfers. This can be achieved by providing fixed costs for using a satellite, reflecting the objectives of different stakeholders, such as prioritising satellites based on proximity to public places like hospitals or schools or their importance for other activities on the waterways. The fixed costs associated with the satellites ensure they are only used if the costs of opening them are lower than the extra driving costs.

The 2E-LRP, by including satellite locations as decision variables, provides a more comprehensive and accurate representation of the system. Both the 2E-LRP and the 2E-VRP, along with their variations, are studied in this chapter. The relevant variations include multi-trip capabilities, time windows, capacitated vehicles, satellite capacity, and/or satellite synchronization. Existing solution methods for these problems are listed in this chapter.

As mentioned earlier, the PTP can be approached as a variant of vehicle routing or location routing problems, depending on the need for determining the launch locations. To be more specific, the

PTP can be seen as a two-echelon vehicle routing or location routing problem. An advantage of the PTP is that no synchronisation is necessary, since the second echelon vehicles are transported by the first echelon vehicles. However, if the process of launching second-echelon vehicles is repeated, so multiple trips are allowed, synchronisation can reduce the number of vehicles needed on the second echelon by merging trips for a single vehicle but ensuring the vehicle is synchronised with the first echelon vehicles for cargo replenishment. Synchronising the movements of first- and second-echelon vehicles can also reduce the waiting time at depots. Therefore, depending on the need for multiple trips, synchronisation can be included. Furthermore, the transportation of second echelon vehicles takes more capacity of the first echelon vehicles than only the freight, so adaptions in the maximum capacities must be inserted. Since capacitated vehicles are assumed for the basic variants of the VRP and the LRP, and synchronisation is investigated for both, the PTP will not be investigated separately.

Figure 2.2 shows the design choices that impact the type of model to use. These decisions add attributes or specific needs to the basic VRP or LRP model. These attributes are optional but do make the model more realistic. The decision for multiple trips largely depends on the capacity of the vehicles. Most vehicles used in city centres do not have large loading capacity, requiring them to perform multiple trips. Whether multiple depots are used is not given as a specific attribute in most literature, but since it is an important factor for selecting a model, it is included in the figure.

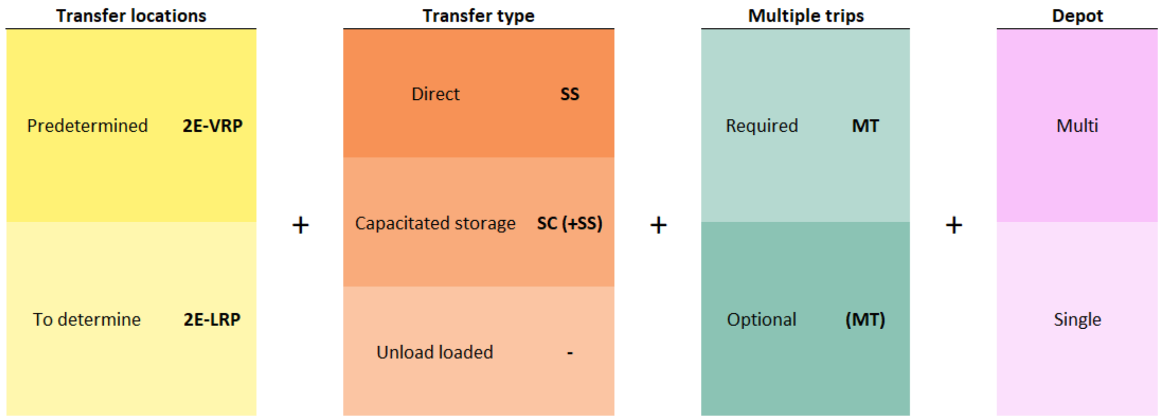


Figure 2.2: Design choices with impact on the model type

To determine the type of model necessary, the problem definition is found by selecting the bold text next to the chosen design and adding them together. These problem definitions can be connected to the solution methods in the next section. For instance, if the satellite locations still have to be determined, direct deliveries take placeand multiple trips are require, the problem is defined as a 2E-MTLRP-SS.

Both VRPs and FLPs are NP-hard. Since LRPs are a combination of these two, LRPs are also NP-hard (Dalfard et al., 2012) (Nikbakhsh & Zegordi, 2010) (Mirhedayatian et al., 2019) and therefore only small instances can be solved exact within a reasonable time. Most research to solving the problems uses heuristic solution methods or a combination of heuristic and exact methods.

Two-Echelon Vehicle Routing Problem

In this section, 2E-VRPs are investigated. The 2E-VRP is suitable for systems where the satellite locations are known, but can also select which satellites to use from a smaller set of satellite locations to enable changes in the set of the satellites used due to weekly or seasonal differences in demand distribution or the network flow capacity for the vehicles.

Two-echelon vehicle routing problems are extensively researched, and over the years, many variants have been studied (Sluijk et al., 2023). A large body of work exists on many variants and therefore, this paper will focus only on the two-echelon vehicle routing problems with satellite synchronisation and/or satellite capacity (2E-VRP-SS or 2E-VRP-SC), possibly with different side constraints. Basic variants of the two-echelon vehicle routing problem will not be included, since they ignore temporal synchronisation, which is essential to the systems provided in this study due to the lack of space in cities. When no storage is available in the satellites and direct transfers are required, satellite synchronisation is required. Not much research has been conducted on satellite synchronisation for direct transfers between the two echelons. Most researchers leave room for the possibility that more than one second-echelon vehicle is loaded simultaneously at one satellite, and/or all load of the first-echelon vehicles can be stored at satellites with infinite storage capacities. This might not be feasible in real-life situations, since equipment or space for simultaneous transfers is not always available.

First, exact solution algorithms for the 2E-VRP-SS are reviewed. Then, heuristic methods are investigated. For both solution methods, only problems that assume capacitated vehicles are considered.

Exact Solution Methods for the Two-Echelon Vehicle Routing Problem
Dellaert et al. (2019) study a two-echelon vehicle routing problem with time-windows (2E-VRPTW) and propose two mathematical formulations with branch-and-price-based algorithms, the first formulation defines the path over both first- and second-echelon tours, the second formulation decomposes the first- and second-echelon paths. They can solve some of their test instances with up to 6 depots, 4 satellites and 100 customers, optimally. Dellaert et al. (2021) propose decomposing the 2E-VRPTW into two VRPTWs and extending the problem to multiple commodities.

Marquès et al. (2020a) suggest a mixed integer programming formulation for the problem with a branch-cut-and-price algorithm to solve it. They are the first to propose an exact algorithm for the two-echelon vehicle routing problem with multi-trip, time-windows and satellite synchronisation (2E-MTVRPTW-SS) and include the possibility of multiple depots. Mhamedi et al. (2022) also propose a Branch-Price-and-Cut algorithm, for solving the 2E-VRPTW-SS including multiple depots, but no multi-trips are allowed. They are able to solve some of the unsolved test instances by Dellaert et al. (2019). Both Mhamedi et al. (2022) and Dellaert et al. (2019) assume a second-echelon vehicle can only receive load from a single first-echelon vehicle. This assumption simplifies the models and algorithm, as well as the operations at the satellites (Sluijk et al., 2023). However, the model by Marquès et al. (2020a) allows for storage and consolidation of freight at satellites, which makes it relevant for more general problems.

According to Sluijk et al. (2023) the best performing exact algorithm for the multi-depot 2E-VRPTW instances with a single commodity is Marquès et al. (2020a), which is able to solve most instances with 100 customers to optimality. Marquès et al. (2020a) also performs best for the 2E-VRPTW instances with a single-depot. It solves more instances than Dellaert et al. (2019) and Mhamedi et al. (2022), and needs shorter computation times to do so.

The algorithm proposed by Marques et al. (2020b) solves the 2E-VRP with satellite capacity but without any other attributes. It is worth mentioning here since it is the best-performing exact algorithm at this moment (Sluijk et al., 2023). The algorithm can solve instances previously available in the literature with up to 200 customers and 10 satellites from one depot. They introduce a new set of 51 instances with up to 300 customers and 15 satellites, and were able to solve 23 of the new instances with up to 300 customers or 15 satellites.

Some relatively new research is being conducted by Karademir et al. (2022). The focus is on an IWLT system in the city centre, this is why they consider an important constraint, namely that only one transfer can take place at a time. Multiple transfer operations that happen simultaneously are not feasible in busy areas with limited space. They are the first to take this into account. The problem solved is a two-echelon vehicle routing problem with time-windows, multiple trips and satellite synchronisation and is formulated as a mixed-integer linear programming problem. They solve instances with one depot, four satellites and 10 customers.

Heuristic Solution Methods for the Two-Echelon Vehicle Routing Problem
Besides the exact solution algorithms, many heuristic methods exist. Only the research regarding interesting variations of the two-echelon vehicle routing problem for systems of this paper and the

best-performing heuristics are reviewed here.

Grangier et al. (2016) are the first to tackle the two-echelon vehicle routing problem with multi-trip, time-windows and satellite synchronisation (2E-MTVRPTW-SS) and propose an adaptive large neighbourhood search. They designed custom destruction and repair heuristics together with an efficient feasibility check and are able to solve instances with one depot, ten satellites and 200 customers.

Li et al. (2020) use a variable neighbourhood search heuristic to solve the two-echelon logistics system with on-street satellites that uses time windows and satellite transshipment constraints, which in the termination of this paper is equal to the 2E-MTVRPTW-SS. Their problem formulation is distinguished of Grangier et al. (2016) in their use of capacitated satellites. They can solve instances with one depot, up to 30 satellites and 900 customers in under two hours. Next to this, they evaluate the economic difference between the use of electric or diesel vehicles and different vehicle capacities.

Anderluh et al. (2021) use a large neighbourhood search embedded in a heuristic rectangle/cuboid splitting to solve the two-echelon vehicle routing problem with multi-trip and satellite synchronisation (2E-MTVRP-SS). They neglect time-windows and the instances they solve are smaller than those of Li et al. (2020), but what makes their research interesting is its option to use multiple objectives, the standard economic objective, but also negative external effects, like emissions and disturbances, caused by congestion and noise. This possibility makes their solution method especially interesting when design choices still have to be made.

Jia et al. (2022) provide both a heuristic and exact solution method for the two-echelon vehicle routing problem with multiple depots, time-windows, satellite capacity and satellite synchronisation. A mixed-integer programming model and an adaptive large neighbourhood search are developed. They are able to solve problems with 2 depots, 10 satellites and 260 customers.

Relatively new research is conducted by Bijvoet (2023), who solve a two-echelon multi-trip vehicle routing problem with synchronisation with decomposition-based heuristics. Special in the work is their consideration of multiple trips for both echelons and usage of a heterogeneous fleet for the second echelon. They solve large-scale instances with one depot, 45 satellites and 758 customers.

According to Sluijk et al. (2023) the neighbourhood search heuristics are best performing for two-echelon vehicle routing problems with one depot. Yet, there is not one heuristic that is clearly the best performing overall.

Table 2.2 presents the solution methods for two-echelon vehicle routing problems discussed in this chapter. The table gives an overview of the attributes covered by the solution methods, whether single or multiple depots are used, what problem size they can solve and whether exact and/or heuristic methods are used. The problem sizes are indicated as *d/s/c*, meaning problems solved with *d* depots, *s* satellites and *c* customers. Notable in the table is that only Li et al. (2020) cover all four attributes, but only use a single depot. This indicates no solution method is available for multiple depots with time-windows, multiple trips, satellite synchronisation and satellite capacity.

Two-Echelon Location Routing Problem

The available research on the two-echelon location routing problem is significantly less than that on the two-echelon vehicle routing problem, however, interest has been increasing over the last few years. Because the research on the 2E-LRP is limited, especially with regard to variations like satellite synchronisation, also a few solutions to the basic problem are discussed. Due to the complexity of 2E-LRPs, large-sized instances are mostly solved by metaheuristics, or exact methods combined with decomposition strategies (Escobar-Vargas et al., 2021). 2E-LRPs are often decomposed in multiple stages, decomposed in two LRPs, or a separate FLP and VRP for both echelons, resulting in as many as four sub-problems (Contardo et al., 2012).

Most of the early papers on 2E-LRPs consider location decisions for only one of the echelons (Cuda

Table 2.2: Overview solution methods two-echelon vehicle routing problem

Paper	Attributes				Depot		Problem size		
	TW[1]	**MT**[2]	**SS**[3]	**SC**[4]	**Single**	**Multi**	**d/s/c** [5]	**Exact**	**Heuristic**
Grangier et al. (2016)	x	x	x		x		1/10/200		x
Dellaert et al. (2019)	x		x			x	6/4/100	x	
Li et al. (2020)	x	x	x	x	x		1/30/900	x	x
Marquès et al. (2020a)	x	x	x			x	6/4/100	x	
Marques et al. (2020b)				x	x		1/15/300	x	
Anderluh et al. (2021)		x	x		x		1/18/100		x
Dellaert et al. (2021)	x		x			x	3/5/100	x	
Jia et al. (2022)	x		x	x		x	2/20/260	x	x
Karademir et al. (2022)	x	x	x		x		1/4/10	x	
Mhamedi et al. (2022)	x		x			x	6/4/100	x	
Bijvoet (2023)	x	x	x		x		1/45/758		x

[1] Time-Windows, [2] Multi-Trip, [3] Satellite Synchronisation, [4] Satellite Capacity, [5] Number of depots/satellites/customers

et al., 2015), however, it might be useful to find the best locations for both the depots and the satellites. Depending on the needs of the designer, a suitable solution algorithm that considers location decisions for either the first echelon, second echelon or both. Nearly all papers on 2E-LRPs ignore synchronisation (Drexl & Schneider, 2015).

Boccia et al. (2011) seem to be the first to tackle the 2E-LRP, however only for small instances. They propose three mixed integer programming models. The models find locations and numbers of the depots and the satellites and determine routes and the number of vehicles for both echelons. For instances with 3 possible depot locations, 5 possible satellite locations and 10 customers it finds optimal solutions within reasonable time (Prodhon & Prins, 2014). For larger instances, the computation time and gap with the best-found solution grow quickly.

Hemmelmayr et al. (2012) present an ALNS metaheuristic for the 2E-VRP with one depot and the authors show how a standard LRP can be modelled as a 2E-VRP. Even though they do not solve the 2E-LRP, their research is worth mentioning, since this decomposition simplifies the LRP. They connect the depot en satellites by dummy vertexes with a fixed opening cost to determine which satellites should be opened to minimise costs. Also, the satellite capacity is enforced by allowing only one dedicated capacitated vehicle to visit its assigned satellite, with a capacity equal to that of the corresponding satellite.

Contardo et al. (2012) observe the 2E-LRP can be decomposed in two LRPs, connected by capacitated satellites. They use a branch-and-cut algorithm to solve the problem with multiple depots. An initial solution for the second echelon is constructed based on the manner used in Hemmelmayr et al. (2012). After this, a solution for the first echelon is constructed by randomly selecting one depot and serving all satellites from it. A destroy-repair iteration is performed on the second-echelon and then on the first-echelon problem. Local Search is only performed on the second-echelon problem.

Winkenbach et al. (2016) present a mixed-integer linear programming (MILP) model to solve large-scale static and deterministic two-echelon location routing problems, which can account for access restrictions to certain city areas by assessing various vehicle types. They propose two models, one single-stage numerical optimization and an optimization heuristic that reduces the computation time by splitting the optimization problem into two interdependent sub-problems. They show numerically that the loss in solution precision is negligible. The one-stage model can solve instances with 900 nodes with 225 possible satellite locations in $3107s$. The two-stage model is iterative and is much faster because the number of active satellites is not a decision variable anymore, the model is repeatedly executed with an increasing number of active satellites for every iteration. The solution with the lowest objective is then used as input for the second stage, in which the routing decisions are made. This two-stage model is able to find solutions for instances of 1600 nodes and 400 possible satellite locations in $965s$. However, the two-stage approach ignores satellite capacities, considers only one depot and just one vehicle type can be used.

Nikbakhsh and Zegordi (2010) is able to solve two-echelon location routing problems with time windows (2E-LRP-TW) for instances with 10 possible depot locations, 50 possible satellite locations and 100 customers in $271s$. They developed a two-phase heuristic, location-first, allocation-routing second for initial solution construction and a neighbourhood search for an initial solution improvement.

Mirhedayatian et al. (2019) claim to be the first to study a two-echelon location routing problem with time windows and synchronisation (2E-LRPTW-SS). They propose a decomposition-based heuristic solution approach, which is done in three stages. First, a configuration of satellite locations is chosen, then, customers are assigned for this configuration and lastly, the routes of the echelons are established. Feedback loops between the stages ensure working towards the best solution. Different sets of instances are tested and solved for at most 40 nodes. The average computation time for the instances was $2993s$.

Escobar-Vargas et al. (2021) presents two mixed-integer programming formulations and an exact solution framework by a dynamic time discretisation scheme for a two-echelon location routing problem with time windows and satellite synchronisation. They formulate the problem as a Two-Echelon Multi-Attribute Location-Routing Problem with fleet synchronisation at intermediate facilities (2E-MALRPS), which results in a 2E-LRPTW-SS by the definitions used in this paper. The two mixed-integer programming formulations used are a compact formulation and a time-space formulation. Because of the temporal dimension of the time-space formulation, the model is more realistic but also less scalable. They propose a dynamic discretisation discovery (DDD) framework to improve the scalability. The DDD solution framework is able to solve instances of 6 depots, 4 satellites and 10 customers optimally in $4936s$ and find feasible solutions for all instances up to 6 depots, 4 satellites and 30 customers in $36000s$.

Table 2.3: Overview solution methods two-echelon location routing problem

Paper	Attributes				Depot		Problem size		
	TW[1]	**MT**[2]	**SS**[3]	**SC**[4]	**Single**	**Multi**	**d/s/c** [5]	**Exact**	**Heuristic**
Nikbakhsh and Zegordi (2010)							10/50/100		
Boccia et al. (2011)							3/10/25		
Contardo et al. (2012)							5/20/200		
Hemmelmayr et al. (2012)							0/20/200		
Winkenbach et al. (2016)							1/225/900		
Mirhedayatian et al. (2019)							1/5/34		
Escobar-Vargas et al. (2021)							6/4/30		

[1] Time-Windows, [2] Multi-Trip, [3] Satellite Synchronisation, [4] Satellite Capacity, [5] Number of depots/satellites/customers

Table 2.3 gives an overview of the most promising research on two-echelon location routing problems and some details about the solution approaches. It is clearly visible that introducing synchronisation reduces the size of the solvable problems. All considered problems include vehicle capacities and a homogeneous fleet for both echelons, except for Winkenbach et al. (2016), where multiple second-echelon vehicles can be assessed. All problems that include multiple depots also consider location decisions for both echelons, so for the depots and the satellites.

As can be seen in Table 2.3, no research has been conducted on two-echelon vehicle routing problems that include time-windows, multi-trip and satellite synchronisation or satellite capacity. This is presumably because of the large computation capacity it takes to tackle such a problem. However, it is important to develop solution methods for problems that include all these attributes, since they make the problem a better representation of the real world.

2.3.3. Summary & Gaps

This chapter aims to answer the question *What decision models for multi-modal transportation systems exist?*. From the research evaluated in this chapter, it can be concluded that a lot of work is carried out researching suitable solution algorithms for location and routing problems and much more research is currently being conducted. However, at this moment, more realistic formulations are often not applicable on the scale for real-life problems. Furthermore, not all attributes have been studied together. More research has to be conducted on these variants and better solution methods should be developed to tackle larger problem instances.

Suitable solution methods for the preferred IWLT system can be found by first connecting the system through Figure 2.2 to the problem formulations. For most system options and their problem formulations multiple solution methods exist, as can be seen in Table 2.2 and Table 2.3. Therefore, the most promising methods are selected, which is based on the size of the problem they can solve within a reasonable time and the additional options they provide. Figure 2.3 gives an overview of the selected best methods for specific systems.

Second, Figure 2.3 can be used to connect the problem formulation to the available solution methods on the right in blue. By following the row of the problem that needs solving, it can be seen which solution methods are available. To illustrate this, following the row for a system with predetermined transfer locations, a single depot, direct transfers and not require multiple trips, this problem can be solved using methods from Anderluh et al. (2021), Li et al. (2020) and Jia et al. (2022). Which of these methods to use depends on additional factors, which will be explained below.

Figure 2.3 is useful to determine which solution methods can be used, but sometimes multiple methods exist. Which method to choose depends on several factors, whether multiple system options are still considered, the size of the problem and the wish for additional attributes.

To show how this connects with which solution method to use, the following example is used. A system is considered with predetermined transfer locations, a single depot and it is decided not to unload loaded vehicles. The options of direct transfers, capacitated storage, or both, and multiple trips are still open.

First, the factor of considering multiple system options is explored. As in this case, there are some remaining design choices. Modelling the different systems and comparing the results helps the system designers to make better decisions on service network design. To model these different systems, multiple solution methods can be used. More specific, Anderluh et al. (2021) for direct transfers with multiple trips, Li et al. (2020) for direct transfers, capacitated storage and multiple trips and lastly, Jia et al. (2022) for direct transfers and capacitated storage. It is most efficient to use one solution method that covers most system options, so only one is needed to test the options. In this case, it would be most efficient to use the method suggested by Li et al. (2020), since it takes into account all of the system options considered and will search for the best option within these system options.

The second factor in choosing a solution method is the size of the problem to be tackled. When the system is already chosen, the choice in solution method can be made based on the problem size they can tackle. In this case, Li et al. (2020) covers the largest problem instances.

Then, the last factor that helps in choosing a solution method, is the wish for additional attributes. Multiple trips are not required, but it might be useful to include the option and improve the system, this would push in the direction of using Li et al. (2020) or Anderluh et al. (2021). Moreover, the system designer might be interested in emissions or other external effects. Anderluh et al. (2021) gives to option to use these external effects as objectives. Furthermore, Li et al. (2020) enables assessing different vehicle capacities, which might be interesting to the system designer.

To put it more generally, it might be interesting to use a solution method that covers multiple options. A solution method that solves the system while taking into account more options will directly determine the more efficient solutions and help to make the remaining design choices, without the need for multiple models. The downside of a solution method that covers this many aspects is that it will have a longer computation time and the problems it can solve may be smaller.

As can be seen in Figure 2.3, not all problems do have a known solution method yet. Solution methods are missing for the two-echelon vehicle routing problem with multiple depots that includes satellite capacity (2E-VRP-SC) and for the two-echelon location routing problem that includes satellite synchronisation and multiple trips (2E-MTLRP-SS). For some of these problems, it is possible to approach them by using one of the other solution methods.

For the two-echelon location routing problem with satellite synchronisation and multiple trips, it might be possible to tackle the problem by a two-echelon vehicle routing problem that includes satellite synchronisation and multiple trips, with some more side constraints as mentioned in the introduction of this section. However, this is an approximation and will most likely not result in the optimal solution. The missing solution methods for problems involving multiple depots and satellite capacity could be replaced by using solution methods with satellite synchronisation, since this synchronisation adds more constraints and will at least result in feasible solutions. However, these solutions do not use all available resources and will therefore not be optimal. A potential direction is to use the existing models by integrating additional attributes within the solution framework for the feasibility as well as cost reduction.

Altogether, for many options of the integrated water- and land-based inland transportation systems discussed in this paper, solution methods are available. However, the size of the problems that can be solved differs a lot and is not always sufficient. Next to this, solution methods that cover a broader range of system options still have to be developed, to help make design choices for IWLT systems.

Transfer locations		Depot	Transfer type		Multiple trips		Studies with appropriate methodologies
Predetermined	2E-VRP	Multi	Direct and capacitated storage	SS + SC	Required	MT	
					Optional	(MT)	Jia et al. (2022)
			Direct	SS	Required	MT	Marques et al. (2020a)
					Optional	(MT)	
			Capacitated storage	SC	Required	MT	
					Optional	(MT)	
			Unload loaded	-	-		
		Single	Direct and capacitated storage	SS + SC	Required	MT	Li et al. (2020) [1]
					Optional	(MT)	
			Direct	SS	Required	MT	Anderluh et al. (2021) [2]
					Optional	(MT)	
			Capacitated storage	SC	Required	MT	
					Optional	(MT)	
			Unload loaded	-	-		
To determine	2E-LRP	Multi	Direct and capacitated storage	SS + SC	Required	MT	
					Optional	(MT)	Escobar-Vargas et al. (2021)
			Direct	SS	Required	MT	
					Optional	(MT)	
			Capacitated storage	SC	Required	MT	Boccia et al. (2011)
					Optional	(MT)	Nikbaksh et al. (2010) Contardo et al. (2012)
			Unload loaded	-	-		
		Single	Direct and capacitated storage	SS + SC	Required	MT	
					Optional	(MT)	
			Direct	SS	Required	MT	
					Optional	(MT)	Mirhedayatain et al. (2019)
			Capacitated storage	SC	Required	MT	Winkenbach et al. (2016) [3]
					Optional	(MT)	
			Unload loaded	-	-		

[1] Enables assessing different vehicle capacities, [2] Enables assessing multiple external objectives, [3] Enables assessing different vehicle capacities

Figure 2.3: Solution methods to use for specific problem type

3 Modelling Methodology

The previous chapter describes the various design choices important for developing an IWLT system, and how these connect to problem formulations and solving methods. This provides a start for modelling the IWLT system, but many strategies exist. This chapter focuses on the third sub-question: *How to develop decision models for integrated water- and land-based transportation systems that allow to solve full-scale realistic problems?*. Multiple strategies found in the literature are investigated for their suitability. The biggest challenge is to develop a model that is able to tackle large instances with many attributes, which is needed to apply the model to the real-life problem in the city of Amsterdam.

In this chapter, first the scope and goals are highlighted. After that, the problem classification is determined. With the problem classification in mind, the modelling approach is established.

3.1. Scope and Goals

This section outlines the scope and goals of the decision model for IWLT systems designed in this research. The key design choices, objectives, and the intended outcomes of the decision model are detailed.

The IWLT system investigated in this research is established in collaboration with the municipality of Amsterdam. The goal is to investigate the feasibility of supplying Horeca in the busy city centre through an IWLT system. The municipality aims to reduce road congestion by shifting part of the transportation to waterways, which is expected to alleviate the pressure on the crowded urban streets.

The broader goal of this research is to develop a decision model that aids system designers or policymakers, in implementing IWLT systems. This model must encompass various system options and scenarios, bridging the decision-making process with real-life applications. It is designed to explore trade-offs, system requirements, and critical design choices, thereby guiding the development and optimisation of IWLT systems.

The key design choices to be explored in this research include the number and locations of satellites, the number and type of vehicles for both water and road modalities, the time span for deliveries, and the storage capacities at satellites. Determining satellite placements is crucial for effective distribution. Determining the size of the fleet equipped for the tasks is important for evaluating the feasibility of the system in terms of implementation costs. Establishing a feasible and efficient time window for delivery operations is essential since regulations on operating times can be installed. Additionally, assessing the need and extent of storage capacity at satellites can significantly impact operational efficiency.

The decision model aims to provide insights into realistic bounds of these design choices and their impact on the overall system objectives. The primary objective for the municipality is to reduce road kilometres in the city centre. However, achieving this goal inevitably affects other city components, since part of the transportation burden is shifted to the waterways. Therefore, a sub-objective is reducing kilometres travelled on waterways. Furthermore, to make sure the system is feasible for real-life application, objectives regarding the number of vehicles for both vessels and road vehicles are included.

Given these interconnected variables and objectives, the model must facilitate experiments to identify feasible bounds and trade-offs among different design choices. Rather than seeking the ”best” system, the research focuses on confirming the feasibility of the IWLT system and providing insights into system requirements under various scenarios. By evaluating and analysing different configurations and bounds, the model will help identify effective strategies for implementing the IWLT system in Amsterdam.

3.2. Problem Definition

Before defining the modeling approach, it is essential to establish a formal problem definition. This allows for a precise classification of the problem and helps in assessing the applicability of current state-of-the-art research.

The problem is to supply customers using multi-modal transportation. Cargo originates from a depot of set DC_w, with unlimited storage and loading capacity, allowing simultaneous loading of multiple vehicles. Transshipment at the depot takes t^{DC} minutes per vessel.

The cargo is then transported by vessels of set F from a depot to satellites. Vessels have a capacity of $q^{\mathrm{W}}[m^3]$ and a speed $v^{\mathrm{W}}[m/s]$. They can perform multiple trips of set W and visit multiple satellites in one trip, if those trips and satellites are assigned to the same depot.

The satellite locations have to be selected from a set S of potential location, of which N^{S} can be opened. Satellites in the standard configuration have no storage capacity, $q^{\mathrm{S}} = 0$, necessitating direct transshipment from vessels to road vehicles, a process taking t^{S} minutes. Vessels might have to wait at a satellite until the cargo is picked up and transshipment activities can only be performed on one vessel and one road vehicle at a satellite simultaneously. However, the satellite capacity can be adjusted for specific cases by changing parameter q^{S}.

Road vehicles of set R transport the cargo from satellites to customers in set C, with a demand of $q_c[m^3]$ per customer and the demand of all customers has to be satisfied. Each road vehicle can perform multiple trips of set V and can visit multiple customers in a trip, as long as their load does not exceed their capacity of $q^{\mathrm{V}}[m^3]$. Road vehicles have a speed of $v^{\mathrm{V}}[m/s]$, and transshipment at a customer takes a fixed t^{C} minutes. Road vehicles start their first trip and end their last trip at a road vehicle depot, DC_v.

Routes are established for both modalities: waterways for first echelon vehicles and roads for second echelon vehicles. Distances between depot, satellites, and customers are given by Δ_{ij}.

All transshipment activities must occur within a maximum time span, $t^{\max}$ minutes. Vessels can start their trip before the beginning of the time span and exceed this time window when travelling back to the depot. Road vehicles can still perform deliveries of the last trip.

This problem is defined as a two-echelon multi-trip location routing problem with satellite synchronisation (2E-MTLRP-SS), incorporating capacitated vehicles, multiple depots and a global time window, with a possibility of satellite storage. Both echelons have a homogeneous fleet. The primary objective is to minimise road burden while ensuring real-life feasibility in terms of costs and time. This involves minimising the number of vehicles required and the distance travelled on the roads while adhering to all time constraints. Additionally, minimising the distance travelled on the waterways is a sub-objective to ensure that reducing road traffic does not result in excessive congestion on the waterways.

Key decision variables include satellite locations, the number of satellites to open, and vehicle numbers for both modalities. Vehicle characteristics are governed by regulations and system requirements and are represented as parameters. The routes of the vehicles are an important factor for the objectives, which are evaluated by kilometres on the roads and waterways.

For this problem classification, it is crucial to assess the size of the problem that the model aims to address. The decision model is designed to explore IWLT system scenarios for real-life applications, so it must be capable of solving problems of considerable size. The case study for the municipality of Amsterdam serves as an excellent representation of a real-world scenario. This case involves 1635 Horeca locations, 56 potential satellite locations and 3 depots. Further details about this case will be introduced in Section 5.1.

3.3. Modelling Approach

Given the previously determined problem classification, it becomes evident that none of the current state-of-the-art solution methods are suitable for this specific type of problem. As illustrated in Figure 2.3, the combination of requirements for the two-echelon location routing problem with multiple depots, satellite synchronisation and multiple trips necessitates a tailored approach.

Since none of the current state-of-the-art solution methods is suitable for the large-scale IWLT system with all its attributes for the city of Amsterdam, a new strategy is developed in this research. The new strategy is developed with inspiration from the decomposition approaches used in literature, adding extra decomposition steps to tackle the large-scale problem. The decomposition approach is used to model different optimisation problems linked via synchronisation in time, space and load, therefore enabling tractable models for realistic-sized problems.

To address the complexity of the two-echelon multi-trip location routing problem with satellite synchronisation for large problem instances, it is essential to decompose the problem effectively while ensuring integration and synchronisation between different stages. First, a review of some of the decomposition approaches from existing literature will be conducted to evaluate their relevance to this research problem. Following this, the integration of these decomposition approaches into the chosen strategy for this study is explained. Finally, the specific decomposition approach adopted in this research will be detailed.

As discussed in Section 2.3, many models exist for 2E-VRPs and 2E-LRPs. However, most only tackle small instances, or only address part of the attributes. Especially for 2E-LRPs, the problem instances that can be solved are small and not applicable to most real-life problems. Only Winkenbach et al. (2016) tackle large instances, but only address satellite capacity and load synchronisation, but no spatial and temporal synchronisation is included.
Li et al. (2020) solve large 2E-VRP problems including synchronisation. Their approach involves creating an initial solution by first constructing second-echelon routes and then constructing routes for the first-echelon that respect the synchronisation constraints. The approach has promising results, but the facility location problem is not included. However, their method of splitting the routing problem of the two echelons is relevant for the system considered in this research.
Contardo et al. (2012) implement a similar decomposition for the 2E-LRP. The problem is split into two LRPs. Decomposing the problem in sub-problems for the two echelons is a commonly used method in literature. Mirhedayatian et al. (2019) approaches the 2E-LRP with a different decomposition. The problem is solved in three stages, first, an FLP for the satellite locations, next, the customers are assigned to the satellites and lastly, the routes of the echelons are established. No decomposition is applied to the routing of the echelons, which is viable for the small problem instances they tackle. However, their decomposition of the FLP and routing is relevant for this research.

With these decomposition approaches considered, the following strategy is formulated for the previously defined problem.
The facility location problem for satellites is treated separately from the routing decisions, as done by Mirhedayatian et al. (2019), to reduce the computational complexity involved in simultaneously determining both location and routing. This approach streamlines the problem into more manageable sub-problems. When tackling the facility location problem, decisions are based on the distances over existing road networks between customers and potential satellite sites. This ensures that the locations selected are strategically viable in terms of proximity to customer locations.

The routing tasks of first- and second-echelon vehicles are also addressed separately, as seen in Contardo et al. (2012) and Li et al. (2020). Initially, by determining the routes for second-echelon vehicles, the trip demand and duration of each trip are established, providing input for the first-echelon routing. The first-echelon vehicles must meet the demands set by the second-echelon routes, but the specific paths of the second-echelon vehicles do not affect the routing decisions of the first-echelon. Time and synchronisation constraints are included in the first-echelon routing problem to ensure integration between the two echelons. This approach ensures that the first-echelon vehicles are effectively coordinated with the second-echelon operations while reducing the model complexity.

For real-life applications, it is essential that vehicles from both echelons perform multiple trips. It is impractical to have a dedicated vehicle for each trip. It is important to note that most literature does not cover multiple trips for both echelons. Incorporating multiple trips for both echelons into the routing problems can make the problem excessively large and complex. Therefore, the problem is further decomposed by treating the multiple trips in a separate scheduling problem.

Given the high number of trips required to meet customer demands, large vehicle sets are necessary for effective scheduling. By splitting the scheduling problem into separate sub-problems for each echelon, the decision variables per problem and the size of the vehicle sets are significantly reduced. This reduction in complexity allows for more efficient scheduling and better resource allocation. Furthermore, with the reduced vehicle sets, it becomes feasible to enhance the schedule by solving decision variables for both echelons within an integrated problem.

All in all, the strategy used in this research is to decompose the problem in an FLP, two separate VRPs for water and street level while incorporating integration and synchronisation, and a scheduling problem. For the two VRPs, using only exact methods reduces the problem variations and instances that can be tackled. Using only heuristic methods can result in sub-optimal results. Therefore, to achieve high-quality results, both heuristic and exact methods are developed and combined. The scheduling problem is added to enable multiple trips and reduce the required number of vehicles.

Figure 3.1 shows the problem decomposition. The problem is decomposed into four problems, indicated in the figure by yellow boxes: the facility location problem, the second-echelon trip generation, the first-echelon trip generation and the scheduling problem. The trip generations and scheduling problem each consist of multiple sub-problems. Below, an overview of the (sub-)problems is given and each of the problems is further explained in the indicated section in Chapter 4:

- **Facility location problem** (Section 4.1)**:**
  MILP model to determine the satellite locations to open and assign customers to those satellites
- **Second-echelon trip generation** (Section 4.2)**:**
  - VRP road initial:
    Heuristic method to establish initial routes for the road vehicles
  - VRP road improvement:
    MILP model to improve the initial road vehicle routes
- **First-echelon trip generation** (Section 4.3)**:**
  - VRP water initial:
    Heuristic method to establish initial routes for the vessels
  - VRP water improvement + synchronisation:
    MILP model to improve the initial vessel routes and implement synchronisation between the two echelons
- **Scheduling problem** (Section 4.4)**:**
  - Scheduling road vehicles initial:
    Heuristic method to create an initial schedule for the road vehicle trips
  - Scheduling road vehicles:
    MILP model to schedule the road vehicle trips and determine the required number of road vehicles while respecting synchronisation constraints to vessels
  - Scheduling vessels:
    MILP model to schedule the vessel trips and determine the required number of vessels while respecting synchronisation constraints to road vehicles
  - Scheduling integrated system:
    MILP model to improve the schedules for both echelons while respecting synchronisation constraints

Concluding, a decomposition approach for the two-echelon multi-trip location routing problem with synchronisation is developed. The decomposition exists out of four main problems with additional sub-problems. This decomposition enables evaluating large-scale problem instances for different system scenarios while incorporating synchronisation.

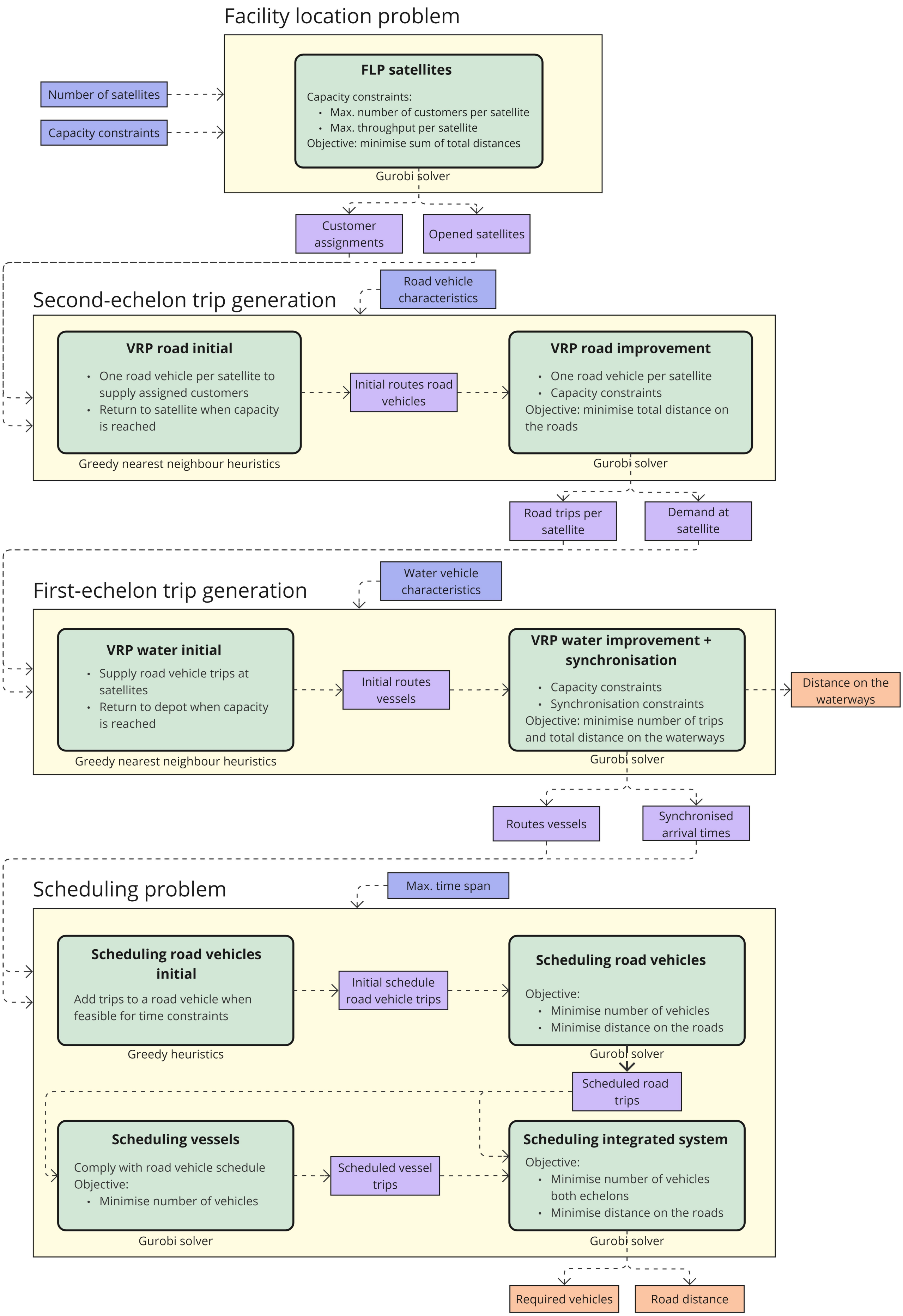


Figure 3.1: Decomposed model approach

4

Mathematical Models

This chapter provides a more elaborate explanation of the sub-problems, including the mathematical formulations. The outputs of each sub-problem are used as inputs for the following sub-problems, as shown previously in Figure 3.1. First, the facility location problem and its variants are described, followed by the second-echelon vehicle routing problem, with its sub-problems. Next, the first-echelon vehicle routing problem and its initial solution are given. Lastly, the separate sub-problems of the scheduling problem are explained.

Section 3.2 introduces parameters and some of the sets used for the models. For clarity, an overview of these parameters and sets is provided below.

Sets

F	set of vessels
W	set of vessel trips
R	set of road vehicles
V	set of road vehicle trips
DC_w	set of vessel depots assigned to vessel trip w in set W
DC_v	set of road vehicle depots assigned to road vehicle trip v in set V
S	set of potential satellite locations
C	set of customers

Parameters

t^{DC}	transshipment time at vessel depot	$[\mathrm{min}]$
q^{W}	capacity of vessels	$[\mathrm{m}^3]$
v^{W}	speed of vessels	$[\mathrm{m/s}]$
N^{S}	number of satellites to open	$[-]$
q^{S}	storage capacity of satellites	$[\mathrm{m}^3]$
t^{S}	transshipment time at satellites	$[\mathrm{min}]$
q_c	demand of customer c in set C	$[\mathrm{m}^3]$
q^{V}	capacity of road vehicles	$[\mathrm{m}^3]$
v^{V}	speed of road vehicles	$[\mathrm{m/s}]$
t^{C}	transshipment time at customers	$[\mathrm{min}]$
$t^{\max}$	maximum time span	$[\mathrm{min}]$
Δ_{ij}	distance between node i and j in sets DC_w, DC_v, S and C	$[\mathrm{m}]$

For modelling purposes, additional sets and parameters are introduced for some of the models. While these sets and parameters can be used in subsequent models, they are not reintroduced to avoid redundancy and keep the text concise.

4.1. Facility Location Problem

The first sub-problem determines the satellite locations. The basic version determines the most suitable satellite locations based on the objective of minimising the total distance on the roads from satellites to customers. Other variants of the FLP are investigated, adding constraints to limit the number of customers assigned to a satellite, since assigning too many customers to one satellite is not desirable. Because of the transshipment time at satellites, it might not be possible to serve all these locations within a reasonable time.

Two options are considered to limit the number of customers assigned to a satellite. The obvious method is to allow a maximum number of customers to be assigned to a satellite. The second option is to limit the throughput allowed at a satellite. The throughput is the units of load transferred through one satellite. Below, the mathematical model of the basic FLP is given first, and then the additions to the mathematical model for the variants are described.

4.1.1. Basic FLP

Variables

U_{ij} if customer $j \in C$ is assigned to satellite $i \in S$: $U_{ij} = 1$, else: $U_{ij} = 0$

O_i if satellite $i \in S$ is open $O_i = 1$, else $O_i = 0$

Objective Function

The objective is to minimise the sum of the distances between customers and satellites:

$$\min \sum_{i \in S} \sum_{j \in C} U_{ij} \Delta_{ij}$$

Functional constraints

1. Each customer must be assigned to one satellite:

$$\sum_{i \in S} U_{ij} = 1 \qquad \forall j \in C \tag{4.1.1}$$

2. Facility opening constraint, a satellite can only be used if it is open:

$$U_{ij} \leq O_i \qquad \forall i \in S, j \in C \tag{4.1.2}$$

3. The number of satellites that are opened is less than or equal to the maximum number of satellites:

$$\sum_{i \in S} O_i \leq N^{\mathrm{S}} \tag{4.1.3}$$

Additional constraints The binary variables can have either a value of 0 or of 1:

$Y_{ij} \in \{0, 1\} \qquad \forall i \in S, j \in C$

$O_i \in \{0, 1\} \qquad \forall i \in S$

4.1.2. Variants of the FLP

The first method is to set a maximum number of customers that can be assigned to a satellite, a constraint is added to the model, the number of the customers assigned to the satellite is maximum B:

$$\sum_{j \in C} U_{ij} \leq B \qquad \forall i \in S \tag{4.1.4}$$

The second method to limit the number of customers to be assigned to a satellite is to implement a maximum satellite throughput. The following constraint is added to the model. The maximum throughput constraint for satellites, the demand of the customers assigned to the satellite is maximum the throughput capacity A:

$$\sum_{j \in C} q_j U_{ij} \leq A \qquad \forall i \in S \tag{4.1.5}$$

The values of A and B can be constants, or dependent on the number of open satellites. These values determine the tightness of the constraints. When set to zero, the customers are evenly distributed over the satellites, when set to a high value, some satellites might be unused. For these constraints to have a positive impact on the system, tests have to be conducted to determine the right value.

4.2. Second-echelon trip generation

The second problem is the second-echelon trip generation. This problem is split up into two sub-problems, first, an initial solution for the routes is created, second, a MIP model is used to improve the vehicle routes. By post-processing, the vehicle routes are split into separate trips and the duration of the trips is calculated.

4.2.1. VRP road initial

The first sub-problem is creating an initial solution for the road vehicle routing problem, which are the vehicles of the second echelon (VRP-E2). The initial solution is created as an input for the second sub-problem, which uses an MIP solver as an exact method for the VRP road. An initial solution is provided to help the solver improve the solutions faster.

The initial routes of the road vehicles are created using simple heuristics, inspired by Greedy and Nearest Neighbour heuristics. For each satellite one vehicle is created that has to supply all customers assigned to that satellite. Customers are greedily added to a vehicle trip until the vehicle capacity is reached, upon which the vehicle returns to the satellite and starts a new trip. This process is repeated for each satellite with its assigned customers. The heuristics create an initial route from each satellite as one long trip. Of course, this is not feasible in real life, since it would take a long time to perform this trip. The long trip is split up in the third sub-problem. The output of the first sub-problem is an initial route per vehicle, as well as the quantity delivered to each customer in this route.

4.2.2. VRP road improvement

The second sub-problem is the road vehicle routing problem improvement. The VRP is modelled using Gurobi, an exact solver. The output of the first sub-problem is used as an initial solution for this model, to reduce the computation time. The VRP improves the routes of the vehicles and has as output the improved routes, still as one long trip per satellite. The output includes the total kilometres on the road, as well as the quantity delivered to each customer, which is important for the routing of the first-echelon vehicles.

From the FLP, sets are created with the customers per satellite, C_s. Each satellite has one road vehicle r assigned to it which serves the customers assigned to the satellite. For modelling purposes, sets are created per road vehicle with its assigned customers and/or satellite, and per satellite with its assigned vehicle or customers.

Algorithm 1 Initial VRP road heuristics

```
for r in road vehicles do
    while customers left to visit by vehicle r do
        for c in customers to visit do
            determine closest customer c
        load of r += demand[c]
        if capacity of r ≥ load of r then
            visit customer c
            remove customer c from customers to visit
        else if capacity of r < load of r then
            return to satellite from previous customer
            set load of r = 0
    return routes of road vehicle per satellite
```

Some variables and parameters have a superscript, the superscript indicates which set it applies to, since some notations are re-used in separate sub-problems.

The initial solution does not contain split deliveries, in this sub-problem split deliveries are allowed, which can improve the solutions further.

Sets

$\bar{S}$	set of opened satellites obtained from the FLP
S_r	satellite to which road vehicle r is assigned
C_r	set of customers assigned to road vehicle $r \in R$
SC_r	combination of customers and satellites for vehicle $r \in R$, $SC_r = S_r \cup C_r$
R_s	road vehicle assigned to satellite $s \in \bar{S}$
C_s	set of customers assigned to satellite $s \in \bar{S}$
SC_s	set of customers assigned to satellite $s \in \bar{S}$ including the satellite itself

Variables

X^{R}_{ijr}	if road vehicle $r \in R$ travels from node $i \in SC_r$ to $j \in SC_r$: $X^{\mathrm{R}}_{ijr} = 1$, else: $X^{\mathrm{R}}_{ijr} = 0$
Q^{R}_{ir}	quantity delivered to customer $i \in C_r$ by road vehicle $r \in R$
Z^{R}_{ir}	if node $i \in SC_r$ is visited by road vehicle $r \in R$: $Z^{\mathrm{R}}_{ir} = 1$, else: $Z^{\mathrm{R}}_{ir} = 0$
L^{R}_{ir}	accumulated load of vehicle $r \in R$ at node $i \in C_r$

Objective Function

The objective is to minimise the sum of the distances travelled by vehicles r:

$$\min \sum_{r \in R} \sum_{i \in SC_r} \sum_{j \in SC_r} \Delta_{ij} X^{\mathrm{R}}_{ijr}$$

Functional constraints

1. Vehicles never travel from node i to node i:

$$X^{\mathrm{R}}_{iir} = 0 \qquad \forall r \in R, i \in SC_r \tag{4.2.1}$$

2. Each customer must be visited by at least one road vehicle:

$$\sum_{j\in SC_s}\sum_{r\in R_s} X^{\mathrm{R}}_{ijr} \geq 1 \qquad \forall i \in C_s \tag{4.2.2}$$

3. Each satellite must be visited at least the number of times needed for the demand of the assigned customers based on vehicle capacity:

$$\sum_{j\in SC_s}\sum_{r\in R_s} X^{\mathrm{R}}_{ijr} \geq \frac{\sum_{i\in C_s} q_i}{q^{\mathrm{R}}} \qquad \forall i \in \bar{S} \tag{4.2.3}$$

4. Arriving and departing road vehicles for a satellite or customer must be the same:

$$\sum_{j\in SC_r} X^{\mathrm{R}}_{ijr} = \sum_{j\in SC_r} X^{\mathrm{R}}_{jir} \qquad \forall i \in SC_r, r \in R \tag{4.2.4}$$

5. $Z^{\mathrm{R}}_{ir} = 1$ if node i is visited by road vehicle r:

$$X^{\mathrm{R}}_{ijr} = 1 \Rightarrow \qquad Z^{\mathrm{R}}_{ir} = 1 \qquad \forall i,j \in SC_r, r \in R \tag{4.2.5}$$

Capacity and demand road vehicles

6. The demand delivered to i by vehicle r is zero if vehicle r does not visit i:

$$Z^{\mathrm{R}}_{ir} = 0 \Rightarrow \qquad Q^{\mathrm{R}}_{ir} = 0 \qquad \forall i \in SC_r, r \in R \tag{4.2.6}$$

7. Demand satisfaction constraint, the sum of the load delivered by all road vehicles to a customer equals the demand of that customer:

$$\sum_{r\in R_s} Q^{\mathrm{R}}_{ir} = q_i \qquad \forall i \in C_s \tag{4.2.7}$$

8. No load is delivered to satellites:

$$Q^{\mathrm{R}}_{ir} = 0 \qquad \forall i \in S_r, r \in R \tag{4.2.8}$$

9. The accumulated load is zero at satellites:

$$L^{\mathrm{R}}_{ir} = 0 \qquad \forall i \in S_r, r \in R \tag{4.2.9}$$

Maximum capacity constraints for road vehicles:

10. The accumulated load delivered by vehicle r for visits from customer i to j:

$$X^{\mathrm{R}}_{ijr} = 1 \Rightarrow \qquad L^{\mathrm{R}}_{jr} - L^{\mathrm{R}}_{ir} - Q^{\mathrm{R}}_{jr} = 0 \qquad \forall i \in SC_r, j \in C_r, r \in R \tag{4.2.10a}$$

The accumulated load of vehicle r is zero at node i if that node is not visited by r:

$$Z^{\mathrm{R}}_{ir} = 0 \Rightarrow \qquad L^{\mathrm{R}}_{ir} = 0 \qquad \forall i \in SC_r, r \in R \tag{4.2.10b}$$

The load delivered to customer i by vehicle r is always less than or equal to the accumulated load of r at customer i:

$$Q^{\mathrm{R}}_{ir} \leq L^{\mathrm{R}}_{ir} \qquad \forall i \in C_r, r \in R \tag{4.2.10c}$$

The accumulated load of vehicle r at customer i is always less than or equal to the maximum capacity of vehicle r:

$$L^{\mathrm{R}}_{ir} \leq q^{\mathrm{V}} \qquad \forall i \in C_r, r \in R \tag{4.2.10d}$$

Additional constraints

11. Binary variables can have either a value of 0 or 1:

$$X_{ijr}^{\mathrm{R}} \in \{0,1\} \qquad \forall i \in SC_r, j \in SC_r, r \in R \tag{4.2.11a}$$

$$Z_{ir}^{\mathrm{R}} \in \{0,1\} \qquad \forall i \in SC, r \in R_s \tag{4.2.11b}$$

Post-processing

The routes of the road vehicles are previously determined as one long trip per satellite. The road vehicle returns to the satellite when its capacity is reached and repeats this until all customers assigned to the satellite are served. By post-processing the results, the route of a road vehicle is split into a separate trip each time the road vehicle visits the satellite. The split trips are further used for synchronisation with the water vehicles and later scheduled to road vehicles based on the number of road vehicles or time period available. The outputs of this sub-problem are road vehicle trips, with their duration and demand at a satellite.

To be able to schedule the trips to road vehicles in a later step, the duration of each trip including the time it would take to arrive at the start of the next trip is calculated. If a road vehicle performs multiple trips from different satellites, it has to travel from the last customer of one trip to the satellite for the next trip. The duration of each trip is the total distance of the trip divided by the vehicle speed plus the number of customers visited in the trip times the transshipment time at a customer. The total distance of a trip is the distance travelled on the road to visit the customers in the trip, until the last customer, plus the distance to the satellite of the next potential trip. This gives p_{kl}, which is the time it takes to perform trip k and get to the start of trip l.

4.3. First-echelon trip generation

The next problem is the trip generation for the vessels. The trips of the road vehicles are used as input by assigning the demand of road vehicles to satellites, which the vessel trips have to satisfy. The vehicle routing problem for vessels is split into two sub-problems. First, a heuristic solution is determined, which is used as an initial solution for the VRP in Gurobi.

4.3.1. VRP water initial

The first sub-problem is the initial solution for the water vehicle routing problem. With a heuristic algorithm based on Greedy and Nearest Neighbour heuristics with capacity constraints, an initial solution for the routes of the vessels is found based on the demand at satellites per road vehicle trip. The output of this sub-problem gives routes for the vessel trips and their load.

Algorithm 2 Initial VRP water heuristics

```
for w in vessels do
    while satellites left to visit do
        for s in satellites to visit in neighourhood do
            determine closest satellite s
        for v in vehicle trips left to supply from satellite s do
            load of w += demand[v, s]
            if capacity of w ≥ load of w then
                w visits satellite c
                w delivers demand[v, s]
                remove vehicle v from vehicle trips left to supply
                set arrival of w before vehicle trip v
            else if capacity of w < load of w then
                return to depot from last visited satellite
        if no demand left at satellite s then
            remove s from satellites to visit
  return Initial routes of vessels, quantity delivered by vessels
```

4.3.2. VRP water improvement + synchronisation

The second sub-problem is the water vehicle routing problem improvement plus synchronisation of the two echelons, solved with Gurobi. In this problem, the outputs of the previous sub-problems are integrated to find an improved solution with synchronisation at the satellites by introducing time constraints. Furthermore, it is implemented that only one street-level vehicle can be loaded at a satellite at the same time.

The outputs of the heuristics in the first sub-problem of the water VRP are used as an initial solution for the MIP solver for the water VRP improvement. The road vehicle trips and their demands determined by Section 4.2 are used as input parameters for the model.

The outputs are the final routes of the water level trips, the kilometres on the water, and the required number of vessel trips. Next, the synchronised arrivals and departures of the water and road vehicle trips are determined. At this moment, the trips still resemble individual vehicles. In the next problem, the trips will be scheduled to vehicles.

The superscript W indicates the parameter or variable is for vessel trips, V for road vehicle trips and WV for both water and road vehicle trips.

Added sets

WV	set of water and road vehicle trips
$WV0$	set of water and road vehicle trips, plus trip 'zero'
DS	combined set of vessel depots and satellites
S_d	set of satellites assigned to vessel depot $d \in DC$
V_s	set of road vehicle trips for satellite $s \in \bar{S}$

Added parameters

L_{sv}	demand at satellite $s \in \bar{S}$ by vehicle trip $v \in V$
Z^{V}_{iv}	nodes visited by vehicle v
$t^{\max D}$	maximum departure time
ζ	importance value for distance in objective
γ	importance value for number of vessel (trips) in objective

Variables

X^{W}_{ijw}	if vessel trip w travels from node i to node j: $X^{\mathrm{W}}_{ijw} = 1$, else: $X^{\mathrm{W}}_{ijw} = 0$
Z^{WV}_{ik}	if node i is visited by vehicle k: $Z^{\mathrm{WV}}_{ik} = 1$, else: $Z^{\mathrm{WV}}_{ik} = 0$
Y_{ikl}	binary variable, $Y_{ikl} = 1$ if both vehicle k and vehicle l visit node i, else: $Y_{ikl} = 0$
A_{ik}	arrival time of vehicle k at node i
dA_{ikl}	absolute difference between arrival times of vehicle k and l at node i
Q^{W}_{iw}	quantity delivered to satellite i by vessel w
L^{W}_{iw}	accumulated load of vessel w at node i
LS_{ik}	accumulated load delivered to satellite i by vessels after arrival of vehicle k
B_{ikl}	binary variable, $B_{ikl} = 1$ if trip k arrives at satellite i after trip l
S_{ik}	stock at satellite i after arrival of vehicle k

N_w^{W} binary variable, $N_w^{\mathrm{W}} = 1$ if trip w is required

G_{ikl} binary variable, $G_{ikl} = 1$ if trip k leaves satellite i after trip l

D_{ik} departure time of trip k from satellite i

I_{iw} idle/waiting time for trip w at satellite i

Objective Function
The objective is to minimise the sum of the distances travelled in trips W times factor ζ plus the number of trips that are performed:

$$\min \zeta \sum_{w\in W} \sum_{i\in DS} \sum_{j\in DS} \Delta_{ij} X_{ijw}^{\mathrm{W}} + \gamma \sum_{w\in W} N_w^{\mathrm{W}}$$

Functional constraints

1. Vessel trips never travel from node i to node i:

$$X_{iiw}^{\mathrm{W}} = 0 \qquad \forall w \in W, i \in DS \tag{4.3.1}$$

2. Arriving and departing vessel trips for a satellite or depot must be the same:

$$\sum_{j\in DS} X_{ijw}^{\mathrm{W}} = \sum_{j\in DS} X_{jiw}^{\mathrm{W}} \qquad \forall i \in DS, w \in W \tag{4.3.2}$$

3. Vessel trips can only visit satellites that are assigned to the same depot:

$$\sum_{i\in S} \sum_{j\notin S_d} X_{ijw}^{\mathrm{W}} = 0 \qquad \forall w \in W, d = DC_w \tag{4.3.3}$$

4. Nodes that are visited in vessel trip w:

$$Z_{iw}^{\mathrm{WV}} = \sum_{j\in DS} X_{ijw}^{\mathrm{W}} \qquad \forall i \in DS, w \in W \tag{4.3.4}$$

5. Nodes that are visited by road vehicle v:

$$Z_{iv}^{\mathrm{WV}} = Z_{iv}^{V} \qquad \forall i \in DS, v \in V \tag{4.3.5}$$

Capacity and demand vessels

6. The demand delivered to i by vehicle w is zero if vehicle w does not visit i:

$$Z_{iw}^{\mathrm{WV}} = 0 \Rightarrow \qquad Q_{iw}^{\mathrm{W}} = 0 \qquad \forall i \in DS, w \in W \tag{4.3.6}$$

7. Demand satisfaction constraint, the sum of the load delivered by all vessels to a satellite equals the demand of at that satellite by road vehicles:

$$\sum_{w\in W} Q_{iw}^{\mathrm{W}} = \sum_{v\in V_s} L_{iv} \qquad \forall i \in \bar{S} \tag{4.3.7}$$

8. No load is delivered to the depot:

$$Q_{iw}^{\mathrm{W}} = 0 \qquad \forall i \in D, w \in W \tag{4.3.8}$$

9. The accumulated load is zero at the depot:

$$L_{iw}^{\mathrm{W}} = 0 \qquad \forall i \in D, w \in W \tag{4.3.9}$$

Maximum capacity constraints for vessels:

10. The accumulated load delivered by vehicle w for visits from satellite i to j:

$$X^{\mathrm{W}}_{ijw} = 1 \Rightarrow \qquad L^{\mathrm{W}}_{jw} - L^{\mathrm{W}}_{iw} - Q^{\mathrm{W}}_{jw} = 0 \qquad \forall i \in DS, j \in \bar{S}, w \in W \tag{4.3.10a}$$

The accumulated load of vehicle w is zero at node i if that node is not visited by w:

$$Z^{\mathrm{WV}}_{iw} = 0 \Rightarrow \qquad L^{\mathrm{W}}_{iw} = 0 \qquad \forall i \in DS, w \in W \tag{4.3.10b}$$

The load delivered to satellite i by vehicle w is always less than or equal to the accumulated load of w at satellite i:

$$Q^{\mathrm{W}}_{iw} \leq L^{\mathrm{W}}_{iw} \qquad \forall i \in \bar{S}, w \in W \tag{4.3.10c}$$

The accumulated load of vehicle w at satellite i is always less than or equal to the maximum capacity of vehicle w:

$$L^{\mathrm{W}}_{iw} \leq q^{\mathrm{W}} \qquad \forall i \in \bar{S}, w \in W \tag{4.3.10d}$$

Time constraints

11. Sequential visits of vessels to satellites:

$$X^{\mathrm{W}}_{ijw} = 1 \Rightarrow \qquad A_{jw} \geq A_{iw} + \frac{\Delta_{ij}}{v^{\mathrm{W}}} + I_{iw} \qquad \forall i \in DS, j \in \bar{S}, w \in W \tag{4.3.11}$$

12. The arrival time at the first satellite in trip w is the start time of trip w plus the travel time plus the loading time at the depot of that trip d:

$$X^{\mathrm{W}}_{djw} = 1 \Rightarrow \qquad A_{jw} \geq A_{dw} + \frac{\Delta_{dj}}{v^{\mathrm{W}}} + t^{DC} \qquad \forall j \in \bar{S}, w \in W, d = DC_w \tag{4.3.12}$$

13. Binary variable $Y_{ikl} = 1$ if both vehicle k and l visit node i:

$$Y_{ikl} = Z^{\mathrm{WV}}_{ik} \cdot Z^{\mathrm{WV}}_{il} \qquad \forall k, l \in WV, i \in \bar{S}, k \neq l \tag{4.3.13}$$

14. Absolute difference between arrival times of vehicle k and l at node i:

$$dA_{ikl} = |A_{ik} - A_{il}| \qquad \forall k, l \in WV, i \in \bar{S} \tag{4.3.14}$$

15. Road vehicles cannot be loaded at one satellite at the same time. The arrival time of road vehicles have to be at least the transshipment time apart:

$$Y_{ikl} = 1 \Rightarrow \qquad dA_{ikl} \geq t^{\mathrm{S}} \qquad \forall k, l \in V, i \in \bar{S} \tag{4.3.15a}$$

Vessels cannot be unloaded at one satellite at the same time. The arrival time of vessels have to be at least the waiting time apart:

$$B_{ikl} = 1 \Rightarrow \qquad dA_{ikl} \geq I_{il} \qquad \forall k, l \in W, i \in \bar{S} \tag{4.3.15b}$$

16. The departure time of a road vehicle trip at a satellite is the arrival time of that trip plus the transshipment time:

$$Z^{\mathrm{WV}}_{iv} = 1 \Rightarrow \qquad D_{iv} = A_{iv} + t^{\mathrm{S}} \qquad \forall v \in V, i \in \bar{S} \tag{4.3.16}$$

17. The departure time of a vessel trip at a satellite is the arrival time of that trip plus the waiting time:

$$Z^{\mathrm{WV}}_{iw} = 1 \Rightarrow \qquad D_{iw} = A_{iw} + I_{iw} \qquad \forall w \in W, i \in \bar{S} \tag{4.3.17}$$

18. The arrival time of vehicles at satellites is equal to or larger than zero:

$$A_{ik} \geq 0 \qquad \forall i \in S, k \in WV \tag{4.3.18}$$

19. The departure time of vehicles from satellites cannot be later than the maximum time span:

$$D_{ik} \leq t^{\max} \qquad \forall i \in S, k \in WV \tag{4.3.19}$$

Satellite synchronisation

20. Binary variable, $B_{ikl} = 1$ if vehicle k arrives at satellite i after vehicle l:

$$Y_{ikl} = 1 \Rightarrow \qquad A_{ik} - K * B_{ikl} - A_{il} \leq 0 \qquad \forall k, l \in WV, i \in \bar{S} \tag{4.3.20a}$$

$$Y_{ikl} = 1 \Rightarrow \qquad B_{ikl} + B_{ilk} = 1 \qquad \forall k, l \in WV, i \in \bar{S} \tag{4.3.20b}$$

$$B_{ikl} + B_{ilk} \leq 1 \qquad \forall k, l \in WV, i \in \bar{S} \tag{4.3.20c}$$

$$Z_{ik}^{\mathrm{WV}} = 0 \Rightarrow \qquad B_{ikl} = B_{ilk} = 0 \qquad \forall k, l \in WV, i \in \bar{S} \tag{4.3.20d}$$

21. Accumulated load delivered to satellite i by vessels after arrival of vehicle k:

$$B_{ikl} = 1 \Rightarrow \qquad LS_{ik} - LS_{il} - Q_{ik}^{\mathrm{W}} \geq 0 \qquad \forall k, l \in WV0, i \in \bar{S} \tag{4.3.21a}$$

$$LS_{ik} \leq \sum_{w \in W} Q_{iw}^{\mathrm{W}} \qquad \forall k \in WV, i \in \bar{S} \tag{4.3.21b}$$

$$Z_{ik}^{\mathrm{WV}} = 0 \Rightarrow \qquad LS_{ik} = 0 \qquad \forall k \in WV, i \in \bar{S} \tag{4.3.21c}$$

22. Stock at satellite i after arrival of vehicle k, the stock is always equal to or greater than zero and always equal to or less than the capacity of vehicle k plus the storage capacity at satellite i:

$$Z_{ik}^{\mathrm{WV}} = 1 \Rightarrow \qquad S_{ik} = -\sum_{l \in V} (L_{il}^{V} * B_{ikl}) - L_{ik}^{V} + LS_{ik} \qquad \forall k \in WV, i \in \bar{S} \tag{4.3.22a}$$

$$S_{ik} \geq 0 \qquad \forall k \in WV, i \in \bar{S} \tag{4.3.22b}$$

$$S_{ik} \leq q^{\mathrm{W}} + q_i^{\mathrm{S}} \qquad \forall k \in WV, i \in \bar{S} \tag{4.3.22c}$$

23. Binary variable, $G_{ikl} = 1$ if vehicle trip k leaves satellite i after vehicle trip l:

$$Y_{ikl} = 1 \Rightarrow \qquad D_{ik} - K * G_{ikl} - D_{il} \leq 0 \qquad \forall k, l \in WV, i \in \bar{S} \tag{4.3.23a}$$

$$Y_{ikl} = 1 \Rightarrow \qquad G_{ikl} + G_{ilk} = 1 \qquad \forall k, l \in WV, i \in \bar{S} \tag{4.3.23b}$$

$$B_{ikl} = 1 \Rightarrow \qquad G_{ikl} = 1 \qquad \forall k, l \in V0, i \in \bar{S} \tag{4.3.23c}$$

$$B_{ikl} = 1 \Rightarrow \qquad G_{ikl} = 1 \qquad \forall k, l \in W0, i \in \bar{S} \tag{4.3.23d}$$

$$Z_{ik}^{\mathrm{WV}} = 0 \Rightarrow \qquad \sum_{l \in WV} G_{ikl} + \sum_{l \in WV} G_{ilk} = 0 \qquad \forall i \in \bar{S}, k \in WV \tag{4.3.23e}$$

24. When a vessel departs from a satellite, the load at the satellite is greater than or equal to zero and less than or equal to the storage capacity at satellite i:

$$Z^{\mathrm{WV}}_{ik} = 1 \Rightarrow \qquad 0 \leq \sum_{l \in V} L^{V}_{il} * G_{ikl} + L^{V}_{ik} - LS_{ik} \leq q^{\mathrm{S}}_{i} \qquad \forall i \in \bar{S}, k \in W \tag{4.3.24}$$

Vehicle zero constraints

25. No demand is delivered by vehicle zero:

$$Q^{\mathrm{W}}_{i0} = 0 \qquad \forall i \in \bar{S} \tag{4.3.25}$$

26. All vehicles that visit a satellite i arrive after vehicle zero:

$$Z^{\mathrm{WV}}_{ik} = 1 \Rightarrow \qquad B_{ik0} = 1 \qquad \forall k \in WV, i \in \bar{S} \tag{4.3.26}$$

Constraints for objective

27. Binary constraint $N^{\mathrm{W}}_{w} = 1$ if vessel trip w visits at least one satellite:

$$Z^{\mathrm{WV}}_{ik} = 1 \Rightarrow \qquad N^{\mathrm{W}}_{w} = 1 \qquad \forall i \in \bar{S}, w \in W \tag{4.3.27}$$

Additional constraints

28. Binary variables can have either a value of 0 or 1:

$$X^{\mathrm{W}}_{ijw} \in \{0, 1\} \qquad \forall i \in DS, j \in DS, w \in W \tag{4.3.28a}$$

$$Z^{\mathrm{WV}}_{ik} \in \{0, 1\} \qquad \forall i \in DS, k \in WV0 \tag{4.3.28b}$$

$$Y_{ikl} \in \{0, 1\} \qquad \forall i \in S, k \in WV, l \in WV \tag{4.3.28c}$$

$$B_{ikl} \in \{0, 1\} \qquad \forall i \in S, k \in WV, l \in WV \tag{4.3.28d}$$

$$N^{\mathrm{W}}_{w} \in \{0, 1\} \qquad \forall w \in W \tag{4.3.28e}$$

$$G_{ikl} \in \{0, 1\} \qquad \forall i \in S, k \in WV, l \in WV \tag{4.3.28f}$$

4.4. Scheduling problem

The last problem is the **scheduling problem of vehicle trips**, which schedules the found trips for the road vehicles and vessels. This problem is split into three sub-problems: MIP optimisations for first the road vehicle schedule; second, the vessel schedule; and lastly, the total schedule for all vehicles. The scheduling problem is split up to reduce the problem instance for MIP optimisation. The outputs of the separate scheduling problems are used as initial solutions for the next scheduling problem, with smaller vehicle sets, adjusted to the found solutions.

Scheduling the trips is necessary to determine the number of vehicles required for performing all deliveries within a specified time span. With unlimited vehicles, each vehicle could perform one trip and the time span would be minimal. However, vehicles are expensive, so this is not desirable. Also, if unlimited time is available, all deliveries could be made by just one vehicle per echelon. Again, this is not desirable. Each day, new orders are made, and with such a system, the orders will pile up. Therefore, a balance has to be found between the time span and the number of vehicles.

Each scheduling model is an addition to the water vehicle routing problem, the constraints given in Subsection 4.3.2 are still valid, with extra constraints added for each scheduling problem. To reduce the solution space, the decision variables for the vessel trip routes and their loads are now fixed to the solutions found in the water vehicle routing problem, $\bar{X}^{\mathrm{W}}_{ijw}$ and $\bar{Q}^{\mathrm{W}}_{iw}$.

4.4.1. Scheduling road vehicles initial

The first sub-problem for scheduling is the initial road vehicle scheduling. A basic initial schedule for the road vehicle trips is determined, by greedily adding a trip to a road vehicle if the start time of that trip is later than the completion time of the previous trip. This initial schedule is created to reduce the size of the problem for the MIP solver, the schedule reduces the required number of road vehicles by approximately 25%.

Algorithm 3 Initial road vehicle schedule heuristics

```
for r in road vehicles do
    while road vehicle trips left to perform do
        for k in trips left do
            add trip k to vehicle r
            for l in trips left do
                if trip l starts from the same satellite as trip k then
                    if start time of trip l is later than the completion time of trip k then
                        add trip l to vehicle r
            break
    return Initial schedule of road vehicle trips
```

4.4.2. Scheduling road vehicles

Next, road vehicle trips are further scheduled to road vehicles using an MIP solver. Below, the mathematical formulation of the model for road vehicle scheduling is given. The objective is to minimise the number of road vehicles and the total distance travelled on the roads. A constraint to ensure a minimum number of road vehicles is implemented, so the schedule is not too tight and leaves room for improvement in the vessel scheduling. The last sub-problem improves the total system schedule without a lower limit on vehicles.

Added parameters

p_{kl} time it takes to perform trip k and get to the start of trip l

d^{R}_{kl} total distance travelled in trip k plus the distance to the start of trip l

$n^{\min R}$ minimum number of road vehicles to use

λ importance value for number of road vehicles in objective

Added variables

T^{V}_{klr} binary variable, $T^{\mathrm{V}}_{klr} = 1$ if vehicle r first performs trip k and then trip l

A^{R}_{vr} start time of trip v by vehicle r

N^{R}_{r} binary variable, $N^{\mathrm{R}}_{r} = 1$ if vehicle r is used

Z^{R}_{vr} binary variable, $Z^{\mathrm{R}}_{vr} = 1$ if vehicle r performs trip v

New objective function

The objective is to minimise the sum of the distances travelled by vehicles r times factor ζ plus the number of road vehicles used times factor λ:

$$\min \zeta \sum_{r\in R} \sum_{k\in V0} \sum_{l\in V0} T^{\mathrm{V}}_{klr} d^{\mathrm{R}}_{kl} + \lambda \sum_{r\in R} N^{\mathrm{R}}_{r}$$

Added functional constraints

1. Each vehicle can only leave the vehicle depot once:

$$\sum_{l \in V0} T^{\mathrm{V}}_{0lr} \leq 0 \qquad \forall r \in R \tag{4.4.1}$$

2. Each trip is performed once:

$$\sum_{r \in R} \sum_{k \in V0} T^{\mathrm{V}}_{klr} = 1 \qquad \forall l \in V \tag{4.4.2}$$

3. Trip l can only be performed by vehicle r if the start time of trip l is later than the end of trip k:

$$T^{\mathrm{V}}_{klr} = 1 \Rightarrow \qquad A^{\mathrm{R}}_{lr} \geq A^{\mathrm{R}}_{kr} + p_{kl} \qquad \forall r \in R, k \in V0, l \in V \tag{4.4.3}$$

4. A trip can never be performed after itself:

$$T^{\mathrm{V}}_{llr} = 0 \qquad \forall r \in R, l \in V0 \tag{4.4.4}$$

5. Vehicle r can only end trip l if it also started it:

$$\sum_{k \in V0} T^{\mathrm{V}}_{klr} = \sum_{k \in V0} T^{\mathrm{V}}_{lkr} \qquad \forall r \in R, l \in V0 \tag{4.4.5}$$

6. Binary variable $N^{\mathrm{R}}_r = 1$ if road vehicle r performs at least one trip:

$$T^{V}_{0lr} = 1 \Rightarrow \qquad N^{\mathrm{R}}_r = 1 \qquad \forall r \in R, l \in V \tag{4.4.6}$$

7. The number of road vehicles used is greater than or equal to the minimum number of road vehicles:

$$\sum_{r \in R} N^{\mathrm{R}}_r \geq n^{\min R} \tag{4.4.7}$$

8. Binary variable $Z^{\mathrm{R}}_{lr} = 1$ if vehicle r performs trip l:

$$Z^{\mathrm{R}}_{lr} = \sum_{k \in V0} T^{\mathrm{V}}_{klr} \qquad \forall r \in R, l \in V \tag{4.4.8}$$

9. The start time of trip v by vehicle r is the start time of trip v at its satellite:

$$A_{V_s[v]v} = \sum_{r \in R} A^{\mathrm{R}}_{vr} \qquad \forall v \in V \tag{4.4.9}$$

Additional constraints

10. Binary variables can have either a value of 0 or 1:

$$T^{\mathrm{V}}_{klr} \in \{0, 1\} \qquad \forall k \in V0, l \in V, r \in R \tag{4.4.10a}$$

$$Z^{\mathrm{R}}_{vr} \in \{0, 1\} \qquad \forall v \in V, r \in R \tag{4.4.10b}$$

$$N^{\mathrm{R}}_r \in \{0, 1\} \qquad \forall r \in R \tag{4.4.10c}$$

4.4.3. Scheduling vessels

The third sub-problem schedules the vessel trips with Gurobi. The road vehicle schedule is used as an input, given as $\bar{A}^{\mathrm{R}}_{iv}$, but the arrival times can be adjusted. The objective is to minimise the number of vessels required to perform the trips determined by the VRP for vessels in Section 4.3.

The model is again an extension of the water vehicle routing improvement, with extra variables and constraints to schedule the vessel trips. Below, the new variables and constraints are given.

First, the inputs determined by previous models and the constraints that integrate the solutions of the road vehicle schedule into this model are given. The arrival times of road vehicles at satellites are given as input, but the constraints allow some adjustments to schedule the vessels.

Added parameter

$n^{\min F}$ minimum number of vessels to use

Added variables

T^{W}_{klf} binary variable, $T^{\mathrm{W}}_{klf} = 1$ if vessel f first performs trip k and then trip l

N^{F}_{f} binary variable, $N^{\mathrm{F}}_{f} = 1$ if vessel f is used

Z^{F}_{wf} binary variable, $Z^{\mathrm{F}}_{wf} = 1$ if vessel f performs trip w

A^{F}_{wf} start time of trip w by vehicle f

New objective function

The objective is to minimise the number of vessels used:

$$\min \sum_{f \in F} N^{\mathrm{F}}_{f}$$

Added functional constraint to implement the road vehicle schedule

1. Trip l can only be performed by vehicle r if the start time of trip l is later than the end of trip k:

$$T^{\mathrm{V}}_{klr} = 1 \Rightarrow \qquad A^{\mathrm{R}}_{lr} \geq A^{\mathrm{R}}_{kr} + p_{kl} \qquad \forall r \in R, k \in V0, l \in V \tag{4.4.11}$$

Added functional constraints to schedule vessels

1. Each vessel trip is performed once:

$$\sum_{f \in F} \sum_{k \in W0} T^{\mathrm{W}}_{klf} = 1 \qquad \forall l \in W \tag{4.4.12}$$

2. Trip l can only be performed by vessel f if the start time of trip l is later than the end of trip k, the end of trip k is the latest departure time from a satellite in trip k plus the time it takes to travel back to the depot:

$$T^{\mathrm{W}}_{klf} = 1 \Rightarrow \qquad A^{\mathrm{F}}_{lf} \geq \max_{i \in S}(D_{ik}) + \frac{\sum_{i \in S}(\Delta_{id} * \bar{X}^{\mathrm{W}}_{idk})}{v^{\mathrm{W}}} \qquad \forall f \in F, k \in W, l \in W, d = D_k \tag{4.4.13}$$

3. A trip can never be performed after itself:

$$T^{\mathrm{W}}_{llf} = 0 \qquad \forall f \in F, l \in W0 \tag{4.4.14}$$

4. Vehicle f can only end trip l if it also started it:

$$\sum_{k \in W0} T^{\mathrm{W}}_{klf} = \sum_{k \in W0} T^{\mathrm{W}}_{lkf} \qquad \forall f \in F, l \in W0 \tag{4.4.15}$$

5. Vehicle f can only perform trips that start from the same depot:

$$T^{\mathrm{W}}_{klf} = 0 \qquad \forall k, l \in W, f \in F, D_l \neq D_k \tag{4.4.16}$$

6. Binary variable $N^{\mathrm{F}}_f = 1$ if road vehicle r performs at least one trip:

$$T^{W}_{0lf} = 1 \Rightarrow \qquad N^{\mathrm{F}}_f = 1 \qquad \forall f \in F, l \in W \tag{4.4.17}$$

7. The number of vessels used is greater than or equal to the minimum number of vessels:

$$\sum_{f \in F} N^{\mathrm{F}}_f \geq n^{\min F} \tag{4.4.18}$$

8. Binary variable $Z^{\mathrm{F}}_{lf} = 1$ if vehicle f performs trip l:

$$Z^{\mathrm{F}}_{lf} = \sum_{k \in W0} T^{\mathrm{W}}_{klf} \qquad \forall f \in F, l \in W \tag{4.4.19}$$

9. Vehicle f can only perform trips if it has started from trip 0:

$$Z^{\mathrm{F}}_{kf} = 1 \Rightarrow \qquad \sum_{l \in W} T^{\mathrm{W}}_{0lf} = 1 \qquad \forall f \in F, k \in W \tag{4.4.20}$$

10. The start time of trip w by vehicle f is the start time of trip w at the depot:

$$A_{dw} = \sum_{f \in F} \sum_{k \in W0} T^{\mathrm{W}}_{kwf} A^{\mathrm{F}}_{wf} \qquad \forall w \in W, d = DC_w \tag{4.4.21}$$

Additional constraints

11. Binary variables can have either a value of 0 or 1:

$$T^{\mathrm{W}}_{klf} \in \{0, 1\} \qquad \forall k \in W0, l \in W, f \in F \tag{4.4.22a}$$

$$Z^{\mathrm{F}}_{wf} \in \{0, 1\} \qquad \forall w \in W, f \in F \tag{4.4.22b}$$

$$N^{\mathrm{F}}_f \in \{0, 1\} \qquad \forall f \in F \tag{4.4.22c}$$

4.4.4. Scheduling integrated system

The last sub-problem combines the decisions for road and vessel scheduling to improve the integrated schedule. The models of the water vehicle routing problem, the road scheduling problem and the vessel scheduling problem are integrated, except for the added constraints for implementing the road vehicle schedule in the water scheduling problem. No minimum is set to the required number of vehicles, $n^{\min R} = n^{\min F} = 0$ and the constraints to implement the road vehicle schedule. The solution found in the previous sub-problem is used as an initial solution for the Gurobi model. By integrating the road and vessel scheduling decisions, improvements can be made while considering the synchronisation. The objectives are to minimise the distance travelled on the roads and the required number of vehicles for both the road and water levels, with importance values ζ, λ and γ, respectively. The objective function is:

$$\min \zeta \sum_{r \in R} \sum_{k \in V0} \sum_{l \in V0} T^{\mathrm{V}}_{klr} d^{\mathrm{R}}_{kl} + \lambda \sum_{r \in R} N^{\mathrm{R}}_r + \gamma \sum_{f \in F} N^{\mathrm{F}}_f \tag{4.4.23}$$

The outputs of this model are the final solutions for the total problem, these solutions allow for evaluating the decision variables and inspecting trade-offs. The most important outputs are the numbers of vehicles used (N^{R}_r, N^{F}_f) and the total distances travelled on the waterways and roads.

5 Experiments

This chapter provides experimental results based on the developed decision models. Experiments are performed to investigate the system requirements under different scenarios. The results help answer the question *What is the performance of the proposed IWLT system under different scenarios of interest?*. Next to this, sensitivity analyses are concluded for some of the input parameters and the demand sets. Furthermore, parameter settings for the Gurobi models are investigated.

The experiments are performed on the Delft High Performance Computing Centre (DHPC), 2024, with 2x Intel Xeon E5-6248R 24C 3.0GHz and 192 GB memory. The models are solved using Gurobi Optimizer, version 11.0.1, implemented in Python 3.12.2.

Before the experiments are discussed, the problem instance for the city of Amsterdam, with its network, data, and parameters, is introduced in Section 5.1. Starting from Section 5.2, experiments are conducted on this problem instance, which allows for the investigation of decision variables and validating the modelling approach used. Experiments and tests are performed for model settings, system scenarios and sensitivity analyses.

5.1. Case Study

This research is conducted in collaboration with the municipality of Amsterdam. The specific IWLT system for the city centre of Amsterdam is solved with the model to provide the municipality with insights for implementation, while simultaneously verifying the modelling approach developed in this research. Data about the demand is collected, parameter values determined and possible satellite locations, customer (Horeca) locations and the network are specified. This section elaborates on those specifications for the case study.

5.1.1. Network and Locations

The model requires an infrastructure network to perform calculations and determine the routes. This infrastructure network can be altered to apply the model to different cities. The focus of the case is the city centre of Amsterdam, for which the canal and road network need to be specified. Some canals restrict vessel sizes, and the road network contains one-way streets.

The canal and road network are obtained from previous research on IWLT systems done between Delft University of Technology and the municipality of Amsterdam. These networks are connected by satellites, of which the nodes are included in both networks. For each of the networks, a distance matrix between each pair of nodes is determined. More information about the constructions of the networks can be found in the research by Bijvoet (2023).

Next to the network, the locations of potential satellites and customers have to be determined. The customer (Horeca) locations can be obtained through public data from the municipality of Amsterdam. The city centre counts 1635 Horeca locations. Furthermore, the potential satellite locations are determined by selecting existing transfer sites in the city centre, 56 in total. The locations used in this research are equal to those in Bijvoet (2023).

Figure 5.1 shows the infrastructure network, the potential satellites, and customer locations. Figure 5.2 gives the depot locations, for both water and road vehicles.

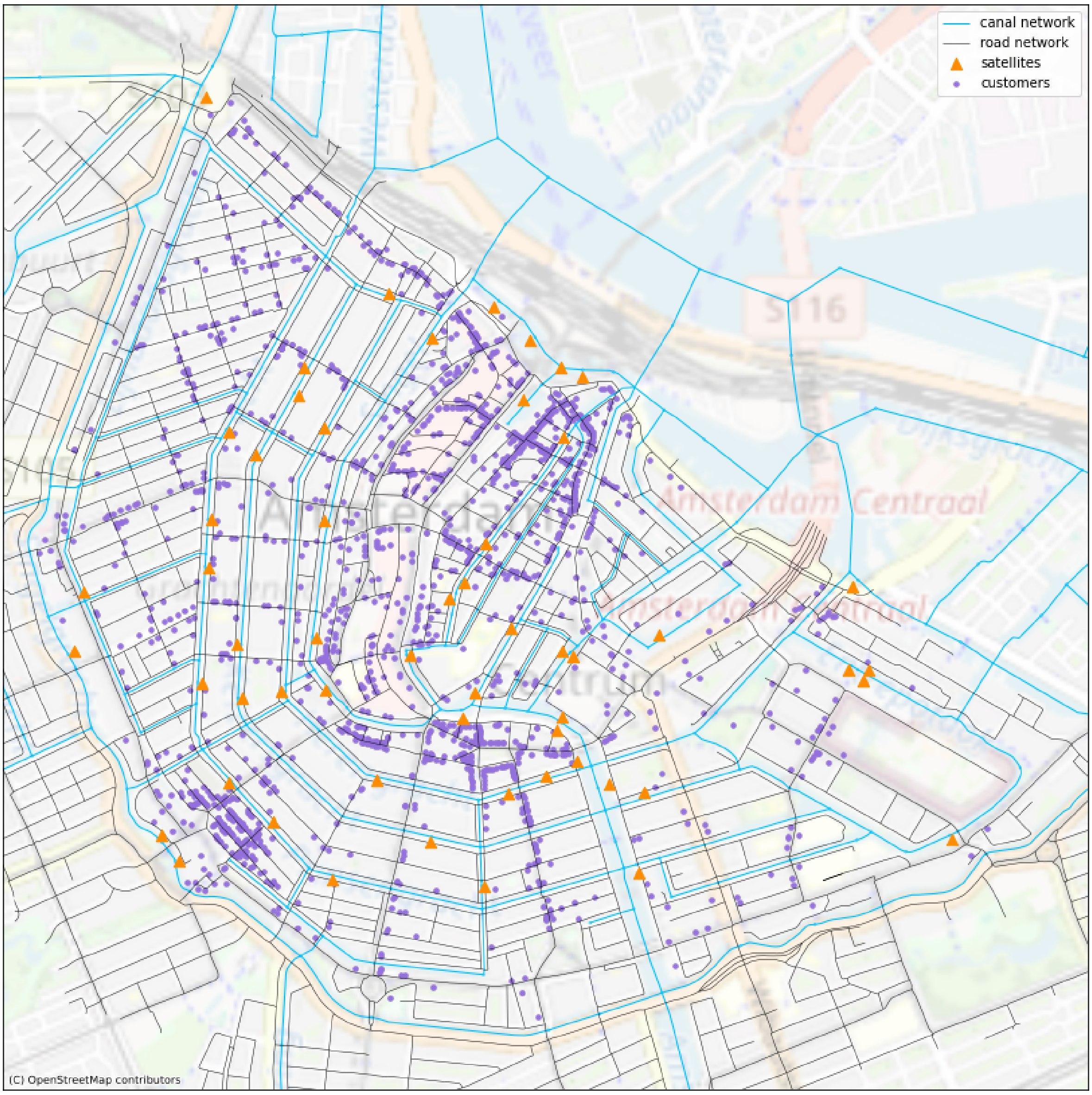


Figure 5.1: Network, satellites and customers, Amsterdam case

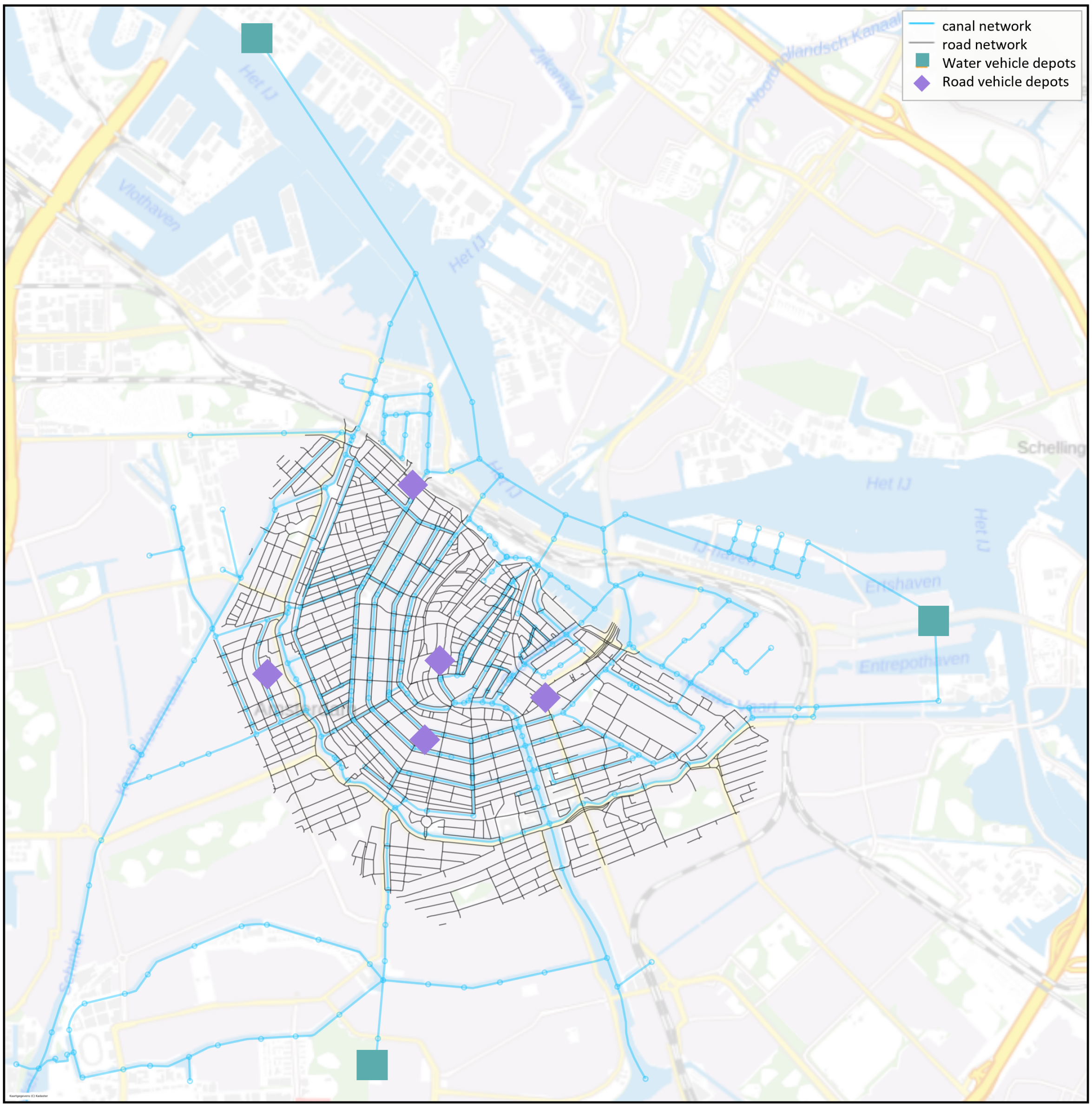


Figure 5.2: Water and road vehicle depots, Amsterdam case

5.1.2. Demand data

The research of Bijvoet (2023) provides 10 demand sets for the Horeca locations. These demand sets are created in consultation with the municipality of Amsterdam. Each set represents one simulated day. The demand is based on the probability of 45% that a location has a demand per day. The demand can be one, two or three units. In the work of Bijvoet (2023), a unit is specified as one rolling container, which is 0.8m in length, 0.64m in width and 1.6m in height, resulting in 0.8192m^3. Table 5.1 shows the demand probability distribution.

Table 5.1: Demand probability distribution Horeca locations (Bijvoet, 2023)

Demand	0	1	2	3
Probability	55%	15%	15%	15%

The Horeca locations with demand can differ each day, however, the sets are all quite similar, with the number of locations with demand between 696 and 758 per day, and the total demand between 1416 and 1520 units. Some basic tests are computed with the demand sets to determine if the difference is significant, and from these, it is concluded that the impact is negligible. Therefore, the experiments in the next chapter are performed for only one of these demand sets.

It is, however, important to investigate the effect on the system requirements when demand changes significantly. Extra demand sets with more extreme values are created to test the adaptability of the system. These sets are shown in Table 5.2. Demand set 2 is the first day of the sets provided by Bijvoet (2023), demand set 1 has lower demand, while the demand increases for set 3 and 4.

The demand units were determined as 0.8192m^3, but in the rest of this research one demand unit is equal to one cubic meter. This makes calculations more clear and accounts for sub-optimal use of vehicle capacity.

Table 5.2: Demand probability distribution per demand set

Demand set						Total demand $[m^3]$	Customers with demand
1	Demand $[m^3]$	0	1	2	3	988	506
	Probability	70%	10%	10%	10%		
2	Demand $[m^3]$	0	1	2	3	1498	750
	Probability	55%	15%	15%	15%		
3	Demand $[m^3]$	0	1	2	3	1952	971
	Probability	40%	20%	20%	20%		
4	Demand $[m^3]$	0	1	2	3	2502	1240
	Probability	25%	25%	25%	25%		

5.1.3. Parameter values

Some input parameter values have to be determined, namely, the vehicle capacities and speeds. Because of city regulations, some bounds are placed on the vehicle characteristics. This section investigates the possible parameter values.

In the city of Amsterdam, tight restrictions for vehicle weight are in place because of the damage to the quay walls. A road vehicle can have a maximum weight of 7500 kilograms, which limits the capacity of the vehicle. Through internet research, it is found that vehicles below 7500 kilograms can transport between 15 and 30 cubic meters. The maximum speed in the city centre is 30 kilometers per hour, however, on average this speed will not be achieved, because of other traffic, turns and traffic lights. The average speed is set to 18 kilometres per hour (5 meters per second).

Because of limited space in the city centre, no storage capacity is enabled for the satellites. By deeper investigation of the satellite locations, it might be possible to assign certain locations with limited storage.

The canals do not have one clear maximum for the vessel size, each canal is characterised by a passage profile, which indicates the maximum size for that canal. To make sure each satellite can be reached, a smaller vessel size is chosen that can access all canals to which satellites are connected. Such a vessel has a maximum width of 4.5 meters and a maximum length of 20 meters. A vessel of this size should be able to transport a maximum of 100 cubic meters of load. The speed of a vessel in the canals is approximately 1.6 meters per second.

Parameter values for the transshipment times are obtained from Bijvoet (2023). Below, an overview of the parameter values used for the experiments is given. These values are the baseline for all experiments unless otherwise stated in the experiment description.

$q^{\mathrm{V}} = 15m^3$ capacity of road vehicles

$v^{\mathrm{V}} = 5m/s$ speed of road vehicles

$q^{\mathrm{W}} = 50m^3$ capacity of vessels

$v^{\mathrm{W}} = 1.6m/s$ speed of vessels

$t^{\mathrm{DC}} = 25min$ transshipment time at the depot

$t^{\mathrm{S}} = 3min$ transshipment time at satellites

$t^{\mathrm{C}} = 1.5min$ transshipment time at customers

$t^{\max} = 480min$ maximum time span

$q^{\mathrm{S}} = 0$ storage capacity of satellites

5.1.4. Problem Instances

The entire case study contains 3 vessel depots, 5 road vehicle depots, 56 potential satellite locations and 1635 Horeca locations, of which the number of locations with demand varies per demand set.

Since this is a large problem, it is useful to create a smaller test case to quickly investigate some scenarios and analyse the model's sensitivity. This set consists of the Horeca locations in a busy city area, the Wallen. This area contains 345 Horeca locations, which is approximately 21% of the Horeca locations in the entire city centre. Figure 5.3 shows the selected Horeca locations.

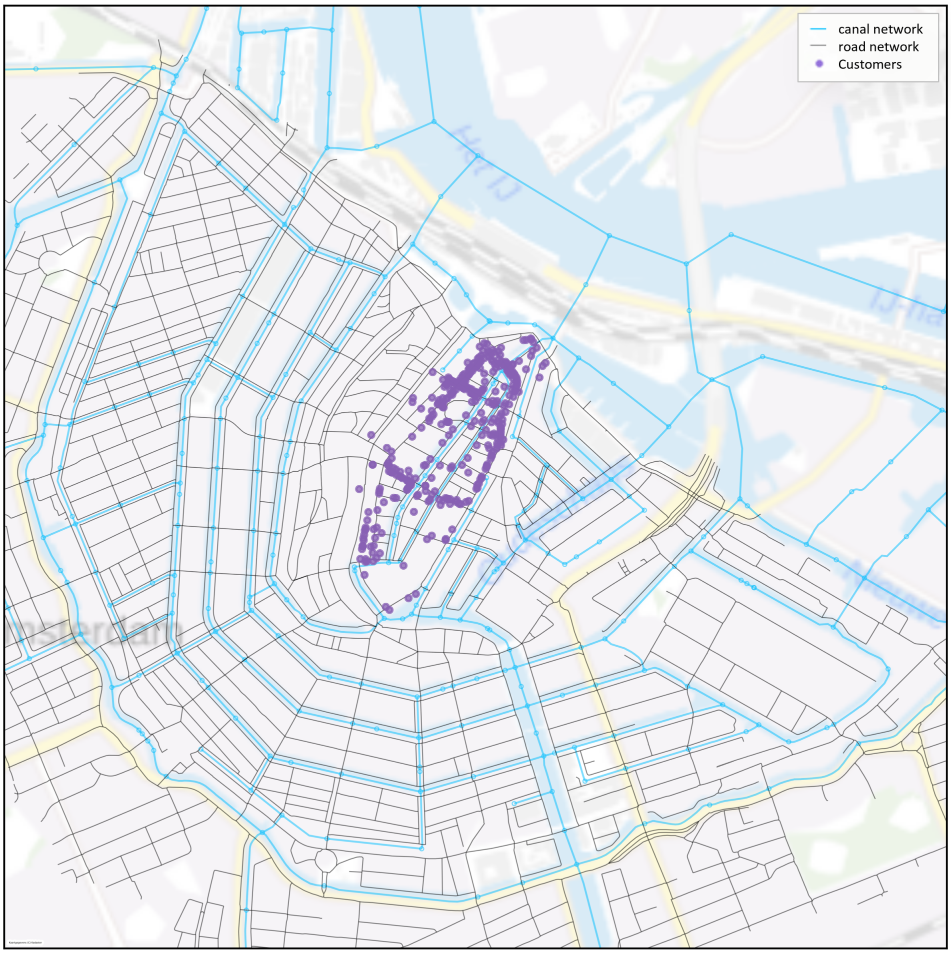


Figure 5.3: Network and customers for the Wallen neighbourhood

5.2. Model settings

Before conducting experiments with different system scenarios, different model settings are evaluated. First, time limits for solving the sub-problems are investigated. Then, the strategies for limiting the number of customers assigned to satellites are tested.

5.2.1. Time limits

The time limit parameter specifies the maximum computation time allowed for the solver to find a solution. It is essential to strike a balance between computation time and solution quality, particularly in the context of large IWLT systems. While the optimisation model should produce results within a reasonable time frame, the definition of ”reasonable time” in this application is nuanced.

Unlike operational decision-making processes that require real-time or near-real-time solutions, the optimisation models developed for the IWLT system are used in the development and design phases. These models assist in determining system requirements and making design choices rather than solving ad hoc operational problems daily. Therefore, the concept of reasonable time can be stretched.

However, after a certain time period, the results of the Gurobi models often do not improve much further. Therefore, tests are conducted for each of the MILP sub-problems, the road vehicle routing problem, the vessel routing problem and all three scheduling problems, to find a balance between the computation time and solution quality. Since the models in this research are all connected through initial solutions, the computation time and solution quality of one model influence the solution quality of all subsequent models. To investigate the impact of changing the time limit of one model, the computation time of that model is varied, while the time limits of the other problems remain at 7200s. Computation times up to 10800s are tested. The FLP finds optimal solutions within 200s, so no additional tests are performed for this model.

For these tests, the number of opened satellites is set to $N^{\mathrm{S}} = 15$. The rest of the parameters are specified in Subsection 5.1.3.

Second-Echelon Vehicle Routing Problem

The first model investigated is the second-echelon vehicle routing problem. To evaluate the performance, the distance on the roads determined by this VRP is investigated, shown in Figure 5.4. The left axis shows the distance on the roads, the right axis its corresponding optimality gap. Increasing the time limit from 100s to 1000s reduces the distance on the roads substantially and up to 3600s there is still some reduction visible. Increasing the time limit further results in small decreases of the optimality gap, but does not improve the solution significantly.

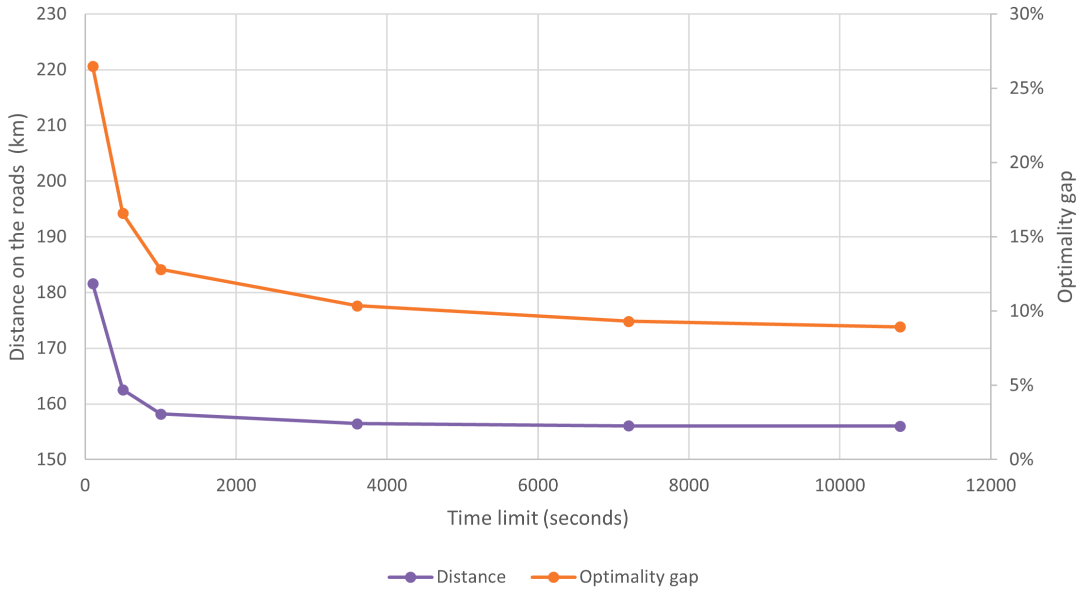


Figure 5.4: Distance travelled on the roads after the second-echelon vehicle routing problem for different time limits of this problem with corresponding optimality gaps

First-Echelon Vehicle Routing Problem and Synchronisation
The water vehicle routing problem combined with the synchronisation is a complex model. Increasing the computation time does not have any visible effect up to 7200s. At 7200s, the distances on the roads and waterways decrease. The results do not change when increasing the computation time further up to 10800s.

Road Vehicle Scheduling
Increasing the time limit for the road scheduling problem has a large impact on both the number of road vehicles required and the distance travelled on the roads. This is to be expected, since the number of road vehicles required directly impacts the distance travelled on the roads, through the added distance from a road vehicle depot for each used vehicle. Figure 5.5 shows the distance on the roads on the left axis and the required number of road vehicles on the right axis. As can be seen they follow the same trend, but are not exactly related. This is due to the trip assignment to road vehicles, which also influences the distance travelled. The results keep improving for increased computation times, but the effect is less significant for higher time limits. This convergence is best visible in Figure 5.6, which shows the optimality gaps for the different computation times. The optimality gap converges to approximately 6%.

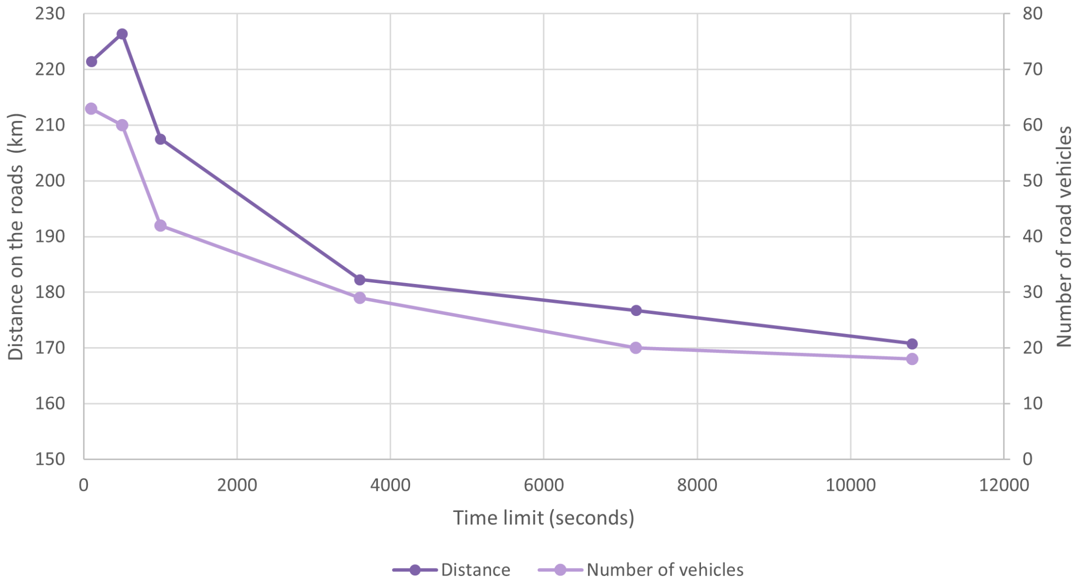


Figure 5.5: Distance travelled on the roads and required number of road vehicles for different time limits of the road vehicle scheduling problem

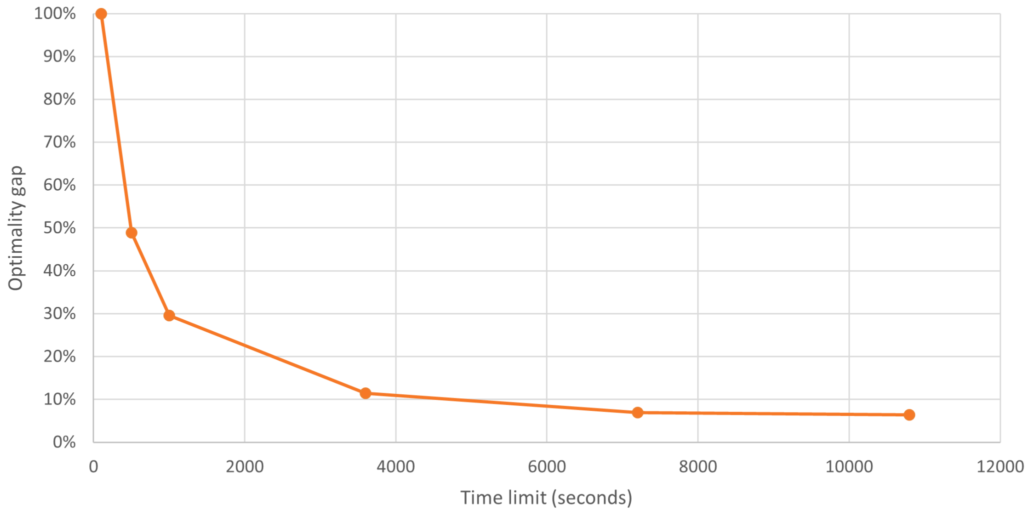


Figure 5.6: Optimality gaps for different computation times of the road vehicle scheduling problem

Vessel Scheduling

The vessel scheduling model only affects the number of vessels required to perform the trips found by the water vehicle routing model. No extra distance is added, since the vessels depart from the depot where the load is stored and they can only perform trips that depart from the same depot. Figure 5.7 shows the number of required vessels, which decreases significantly for increased computation times. For a computation time of 3600s, the number of required vehicles decreases more than 50%, and a reduction of 62% is found after 10800s. The optimality gap converges to approximately 40%, which is quite high, but this gap is highly dependent on the lower bound implemented on the number of vessels. Setting a higher lower bound results in better optimality gaps and even ”optimal” solutions, but the objective is to determine the lowest number of vehicles possible, so the lower bound is set to a value that might not be feasible but forces the model to search for better solutions. Therefore, the large optimality gaps are acceptable.

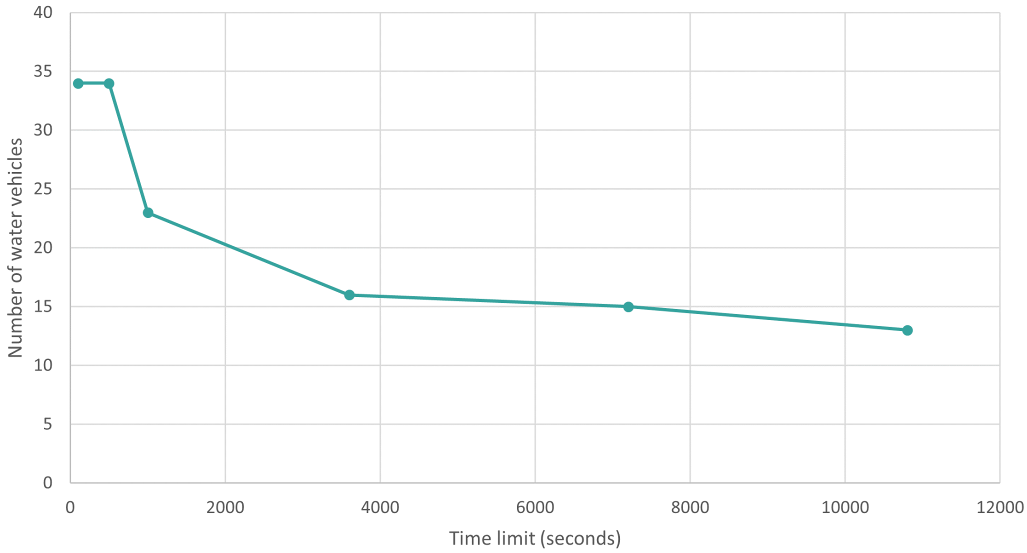


Figure 5.7: Number of required vessels for different time limits of the vessel scheduling problem

Integrated Scheduling

Increasing the computation time for the integrated scheduling problem influences the distance travelled on the roads, the number of road vehicles and the number of vessels required. Since all three objectives are improved by this model, tests are extended to 14400s for this model. However, a computation time of 14400s does not result in better solutions compared with a computation time of 10800s. All three objectives follow the same trend for different computation times, as can be seen in Figure 5.8. Improvements start at 1000s and continue up to 10800s.

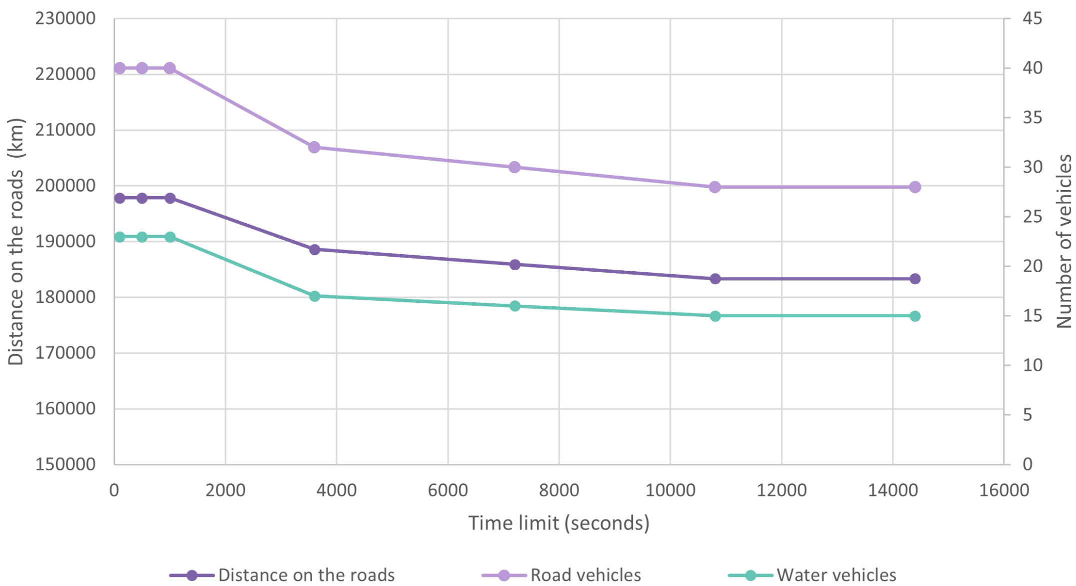


Figure 5.8: Number of required vehicles and distance travelled on the roads for different time limits of the integrated scheduling problem

With the results evaluated in this section, it can be concluded that higher computation times significantly improve the results for the scheduling problems. For the road vehicle problem, a time limit of 1000s already provides good quality solutions, but 3600s ensures most improvements are found. The vessel scheduling problem only improves the solutions at a computation time of 7200s. All three scheduling problems seem to converge at 10800s. Therefore, for the next experiments on the full case study, the time limits are set to:

Second-echelon vehicle routing problem:	3600s
First-echelon vehicle routing problem:	7200s
Road vehicle scheduling problem:	10800s
Vessel scheduling problem:	10800s
Integrated scheduling problem:	10800s

Some tests were performed for different computation times on the smaller instance for the Wallen neighbourhood. All models converged or found solutions with a 0% optimality gap within 3600s. Therefore, the time limits are all set to 3600s for the case test instance.

5.2.2. FLP strategies

Two variants to limit the customers assigned to satellites in the FLP are given in Subsection 4.1.1. For both of these constraints, many possible equations can be used that change the tightness of the constraint. It is possible to precisely even out the number of customers so each satellite has the same number of customers assigned, but this might not have the best results since some customers will be assigned to satellites further away. Some freedom can be implemented, allowing the assignment of more customers to satellites when that is more favourable for the distance travelled on the roads. How much freedom is necessary for the best results is investigated.

Tests are conducted to investigate the constraints' impact on the most important decision variables and to observe the system's behaviour regarding satellite utilisation.

The first method is to assign a maximum of B customers to a satellite, implemented by Equation 4.1.4. The value of B is further defined as:

$$B = \frac{|C|}{N^S} \cdot b \tag{5.2.1}$$

Using this equation, the maximum number of customers per satellite depends on the total number of customers, $|C|$, the number of opened satellites, N^S and the factor b. By including the number of customers and opened satellites in the equation, the constraint is applicable to different system scenarios. The factor b has to be larger than 1, to ensure all customers can be assigned to a satellite.

The second method to limit the number of customers assigned to a satellite is to implement a maximum satellite throughput as shown in Equation 4.1.5. A is defined as:

$$A = \frac{\sum_{i \in C} q_i}{N^S} \cdot a \tag{5.2.2}$$

The maximum throughput of satellites depends on the total customer demand, $\sum_{i \in C} q_i$, the number of opened satellites, N^S and a. Again, a has to be larger than 1, to ensure all customers can be assigned to a satellite.

Experiments with the constraints are performed on the Wallen neighbourhood defined in Subsection 5.1.4 for the demand distribution provided by Bijvoet (2023), resulting in 151 Horeca locations with a total demand of 290m^3 in the Wallen neighbourhood. The factors a and b are varied from 1 to 2.5 and the number of opened satellites N^S is set to 2 or 3.

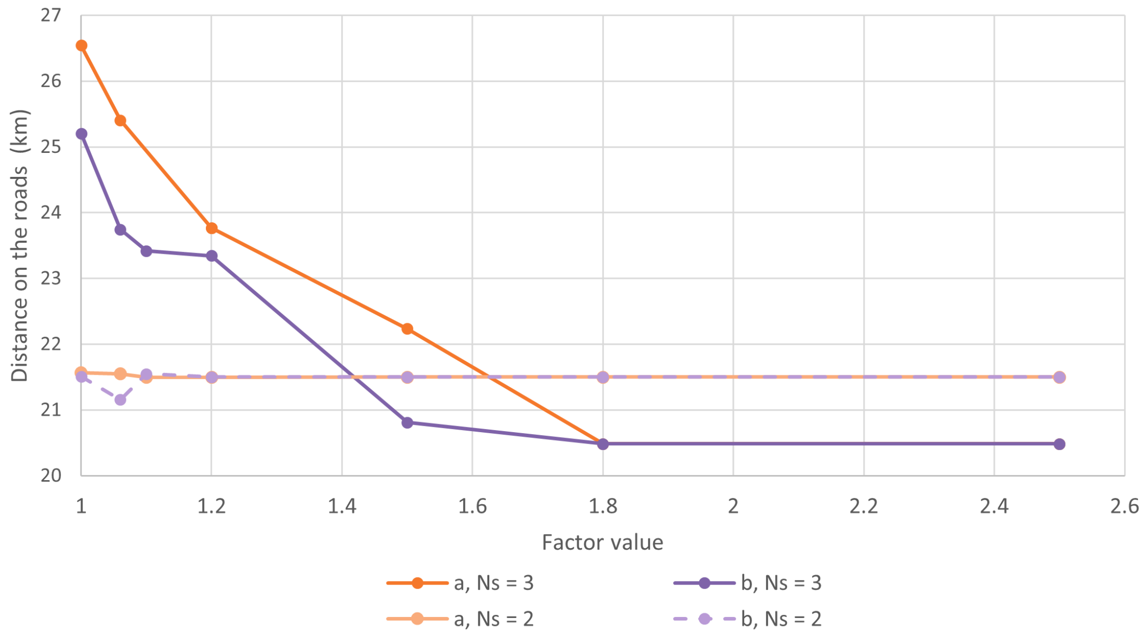


Figure 5.9: Total distance travelled on the roads under different FLP constraints for 2 and 3 opened satellites (Ns=2,3), factor a limits the throughput, factor b limits the number of customers

In Figure 5.9 the distances travelled on the roads under different FLP constraints are shown. It is interesting to see the difference in the effect of changing the tightness of the FLP constraints between 2 and 3 opened satellites. With 3 opened satellites, loosening the constraint reduces the distance on the roads, while for 2 opened satellites the distance is fairly stable. Customers are assigned to minimise the sum of the distances to the satellites, while respecting the limit on the customer assignment per satellite. With a tight constraint for 3 satellites, customers have to be distributed over those 3 satellites, which can result in sub-optimal customer assignment. When the constraint is loosened, customers can be assigned to their closest satellite, resulting in less distance on the roads. When the constraint on the number of customers is implemented, 3 opened satellites perform better than 2 satellites for a factor of $b \geq 1.5$, while constraining the throughput of a satellite performs better for 3 satellites starting at a factor of $a = 1.8$. The small dip at $b = 1.06$ for 2 opened satellites can be ascribed to small differences in the customer assignment. The locations of the road vehicle depots, which might be closer to one of the opened satellites. A tighter constraint forces customers to be assigned to that satellite, reducing the distance travelled from the depot to the satellite.

In such systems where vehicles are utilising shared resources at the satellites, the transshipment processing capacity becomes significant. The impact of adjusting the maximum throughput constraint appears to have a larger negative impact on the road distance, compared to tightening the maximum customer constraint. This can be attributed to the need to assign customers with higher demand to more distant satellites under throughput constraints. While the constraint on the maximum number of customers allows for more favourable assignments by selecting the customer with the least additional distance, the throughput constraint might necessitate less optimal assignments.

The distribution of the satellite utilisation for three satellites under different constraint factors to limit the number of customers is visualised in Figure 5.10. When the constraint is loosened, a large difference in utilisation between satellites is visible. It is important to note that it is expected the distance on the roads increases when the FLP constraint is tightened. However, for practical applications it is still relevant to limit the number of customers supplied from one satellite. For the scenarios investigated here, the uneven satellite utilisation is not necessarily a problem, however, for a shorter maximum time span and a larger number of customers, the system can become infeasible. Furthermore, evenly distributing the satellite utilisation will minimise inconveniences for city residents.

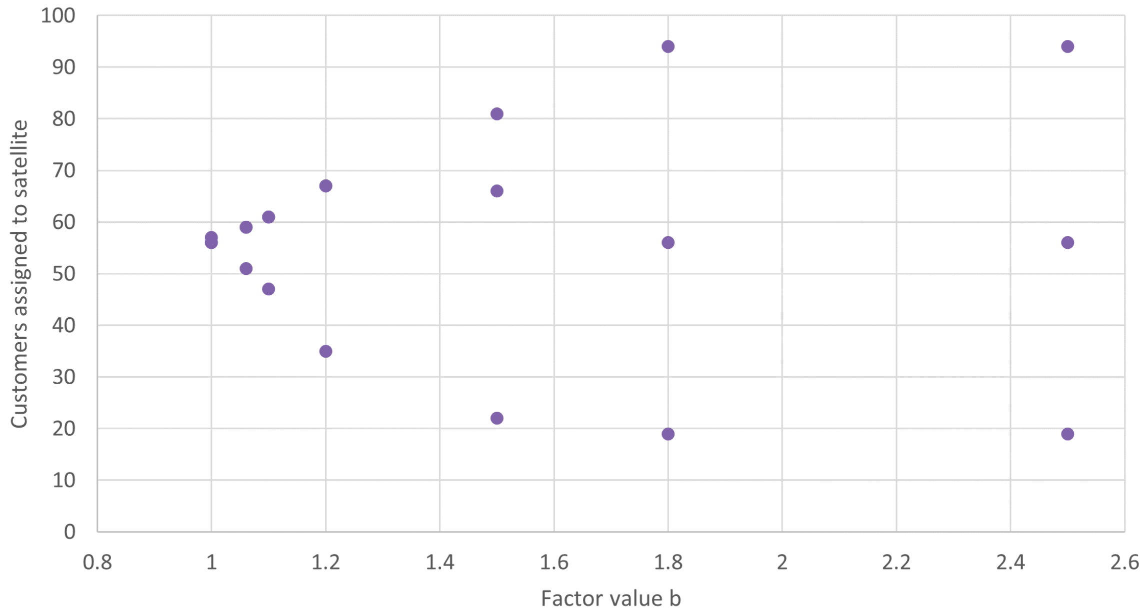


Figure 5.10: Number of customers assigned to satellites under different FLP constraints on the number of customers (factor b) for 3 opened satellites

Allowing 1.5 times the evenly divided number of customers to be assigned to a satellite provides enough flexibility for near-optimal customer assignment to satellites while distributing the utilisation more evenly.

5.2.3. Objectives Ratios

Each of the MIP models aims to minimise its respective objective. The facility location problem, road vehicle routing problem, and vessel scheduling problem each have a single objective. However, the water vehicle routing problem, road vehicle scheduling problem, and integrated scheduling problem involve multiple objectives.

For the water vehicle routing problem and the road vehicle scheduling problem, these objectives complement each other. The water vehicle routing problem aims to minimise both the distance travelled on waterways and the number of trips. These goals are aligned, as fewer trips generally result in less distance travelled to and from depots. Similarly, the road vehicle scheduling problem seeks to minimise the number of road vehicles and the distance travelled on roads. Although these objectives are complementary, a balance must be established. For instance, it would be undesirable to add an extra road vehicle merely to reduce the distance by a few kilometres.

The integrated scheduling problem is more complex, as it combines multiple objectives: minimising the number of vessels, the number of road vehicles, and the distance travelled on roads. These objectives can be conflicting. Reducing the number of vessels might limit the flexibility in arrival times for road vehicles, potentially increasing the number of road vehicles required and the distance travelled on roads. The objective function is specified as:

$$\min \zeta \sum_{r\in R} \sum_{k\in V0} \sum_{l\in V0} T^{\mathrm{V}}_{klr} d^{\mathrm{R}}_{kl} + \lambda \sum_{r\in R} N^{\mathrm{R}}_{r} + \gamma \sum_{f\in F} N^{\mathrm{F}}_{f} \tag{5.2.3}$$

With the importance values; ζ for the distances on the roads, λ for the number of road vehicles and γ for the number of vessels. To investigate the balance between these objectives, experiments are conducted with varying importance ratios between the number of road vehicles (λ) and vessels (γ). These experiments explore a range of ratios from an extreme case where the importance of reducing road vehicles is 1/500 the importance of reducing vessels, to more balanced ratios of λ/γ =1/5, up to λ/γ =2/1. This helps to understand the sensitivity and trade-offs between the different objectives within the integrated scheduling problem. The importance of the distance on the roads is set to a constant low value of $\zeta = 0.0001$, which does not affect the objective so much but prevents road vehicles from travelling to satellites at the other side of the city centre.

The experiments are performed on demand set 1, as specified in Table 5.1. Some additional experiments were conducted on demand set 2 provided by Bijvoet (2023), to validate the results. For both demand sets, 12 satellites are used and the parameters are equal to those specified in Subsection 5.1.3.

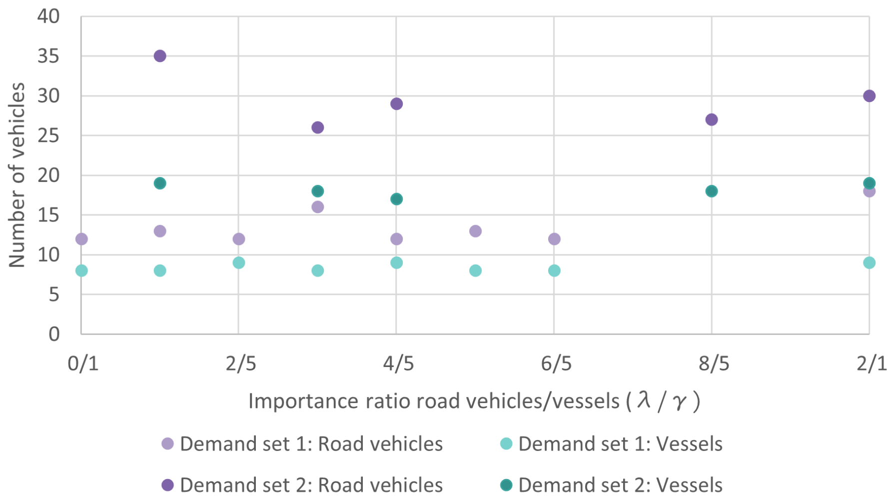


Figure 5.11: Number of customers vehicles for demand set 1 and 2 in the entire city centre, with varying importance ratios in the objective

The results from varying the importance ratio between the number of road vehicles and vessels in the objective function do not show a clear trend in the number of vehicles used. As depicted in the Figure 5.11, different ratios result in varied numbers of road vehicles and vessels without a consistent pattern. This lack of trend can be attributed to the complex interdependencies within the system. The number of road vehicles and vessels required are interdependent and influenced by numerous factors, such as delivery routes and synchronisation requirements. Simply adjusting the importance ratio might not capture these complex interactions. Other factors influencing the lack of trend are the discrete nature of vehicle counts and local optima in the optimisation process. These factors collectively contribute to the absence of a straightforward relationship between the importance ratio and the number of vehicles used.

Still, a decision on the importance ratio must be made, and this can be done by evaluating the practical significance of minimising the number of vehicles. While vessels are more expensive to purchase, the primary objective is to reduce busyness on the roads. This consideration leads to the selection of a ratio that balances these factors. The chosen ratio of 4 road vehicles to 5 vessels aims to achieve a balance between cost and road usage. This ratio acknowledges the higher financial cost of vessels, but it also emphasises the importance of minimising road vehicles to alleviate congestion and reduce the distance travelled on the roads.

5.3. Scenarios

It is important to evaluate the results of different system scenarios for practical application. In this section, the effect of changing the number of opened satellites on the system performance is investigated, which is valuable knowledge for developing the IWLT system. The system is also evaluated for different maximum time spans to provide insights into the system requirements when limited time is available. Additionally, experiments with varying storage capacities at satellites are conducted.

5.3.1. Number of Satellites

One of the most important design choices for developing an IWLT system is the number of satellites to open. Having a small number of satellites in the city centre means these satellites are used intensively, which can create nuisance under city residents. However, a large number of satellites might also not be desirable since satellites require blockage of parking spaces and can congest the waterways when transshipment is taking place. Therefore, it is important to have insights into the effect of the number of satellites on the road and water kilometres, so these factors can be weighted and decisions can be made.

First, experiments are performed for the Wallen neighbourhood, since it is valuable to examine the behaviour of the system with smaller customer sets to explore the possibility of initiating a smaller-scale pilot program. To do so, experiments with 1 to 10 satellites are performed on the Wallen case. First, the same demand distribution of set 2 is used for the 345 Horeca locations in the Wallen neighbourhood. Second, demand set 3, as specified in Table 5.2 is implemented on these locations in the Wallen neighbourhood. Figure 5.12 shows the distances travelled on the roads for these experiments.

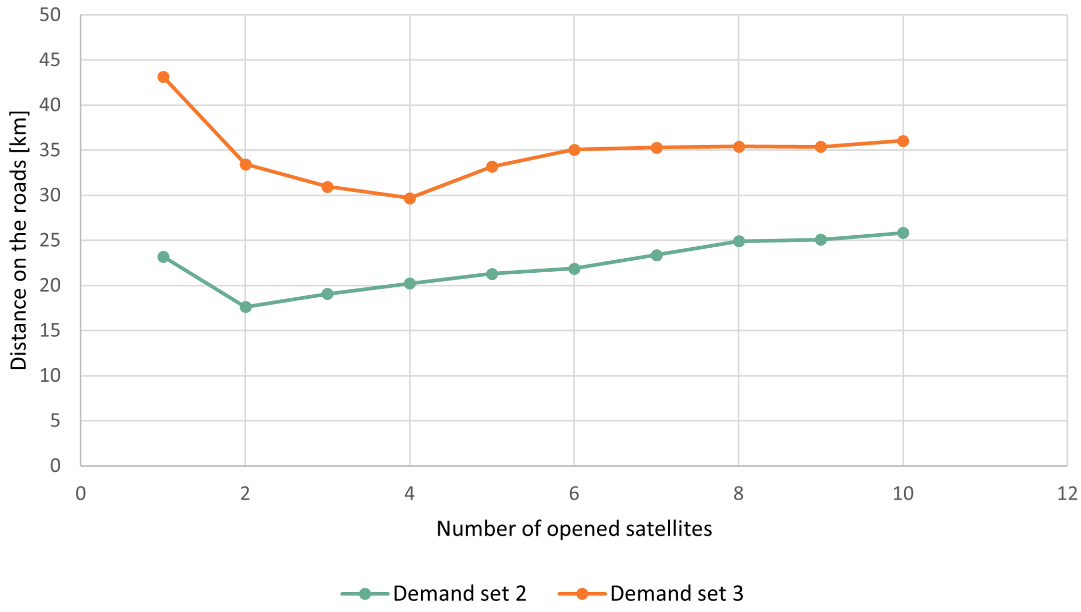


Figure 5.12: Distances travelled on the roads after integrated scheduling in the Wallen neighbourhood, for demand set 2 and 3 with 1 to 10 satellites

Demand set 2 in Figure 5.12 supplies 151 Horeca locations and provides a total demand of $290m^3$, while demand set 3 serves 263 customers and delivers a total demand of $541m^3$. Demand set 2 performs best for 2 satellites, while demand set 3 has better results for 4 satellites. These results indicate a relation between the demand set and the system's performance for different numbers of satellites. This relation is investigated further after results for the entire city centre are analysed.

The same experiment is conducted for supplying the entire city centre. The model is run for 3 to 25 opened satellites to investigate the effect of the number of satellites, with the timelimits specified in Subsection 5.2.1 per sub-problem and the FLP constraint on the number of customers with $b = 1.5$. The customer demand is specified in demand set 2 of Table 5.2 provided by Bijvoet (2023).

It is interesting to analyse the systems performance for the results of the road scheduling problem first, since all scenarios use the same number of road vehicles after this scheduling problem because of the lower bound on this. Therefore, the results are not yet dependent on the extra distance travelled from and to the road vehicle depots by added vehicles and can be easily compared. The optimality gaps determined by Gurobi for these scenarios are approximately equal to the Optimality gaps for fewer opened satellites and the same number of road vehicles is used. Figure 5.13 shows the distances travelled on the roads found by the road vehicle scheduling problem and found after the integrated scheduling problem. Looking at the distances after the road scheduling problem, it can be seen that the distance reduces substantially for each extra opened satellite for up to 9 satellites, is at a minimum for 12 opened satellites and starts to increase for extra opened satellites. This indicates the systems performance is better for 9 to 13 opened satellites, which can have three causes, first: the FLP constraint forces customers to be assigned to the extra opened satellites, even if these locations are less favourable, second: vehicles might have to travel more between satellites, third: the road vehicle depots might be located further away from some satellites. Investigating the results of the distance travelled after the integrated scheduling model, the same trend is visible. Noteworthy is that no improvements on the distance is found in the integrated scheduling problem for 16 or more opened satellites. The optimality gaps of the integrated scheduling model determined by Gurobi for these scenarios are approximately equal to the optimality gaps for fewer opened satellites.

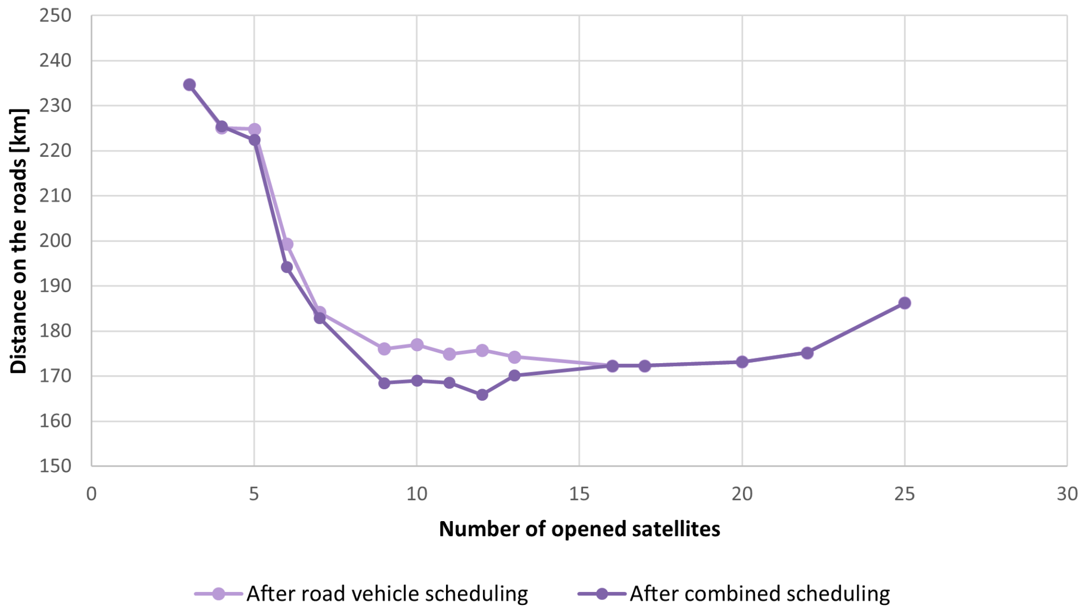
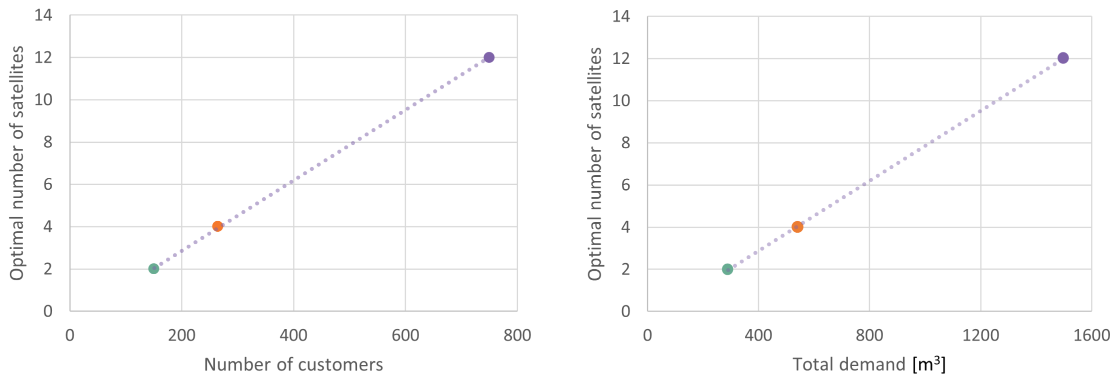


Figure 5.13: Distances travelled on the roads after road vehicle scheduling and integrated scheduling, for 3 to 25 satellites

The results shown in Figure 5.13 are based on the system that serves 750 Horeca locations with a total demand of $1498m^3$, which gives 12 satellites for the best performing system scenario. Figure 5.14 shows the trend between the best performing number of satellites and the specifics of the case studied, with in Figure 5.14a the number of customers on the x-axis and Figure 5.14b the total demand on the x-axis. These results indicate a linear relation between the demand sets and the number of satellites to open. It is important to note these demand sets all assume an evenly distributed demand of 1 to $3m^3$ per Horeca location, the difference is in the number of locations with demand. In Subsection 5.4.1 variations to these sets are further explored.



(a) (b)

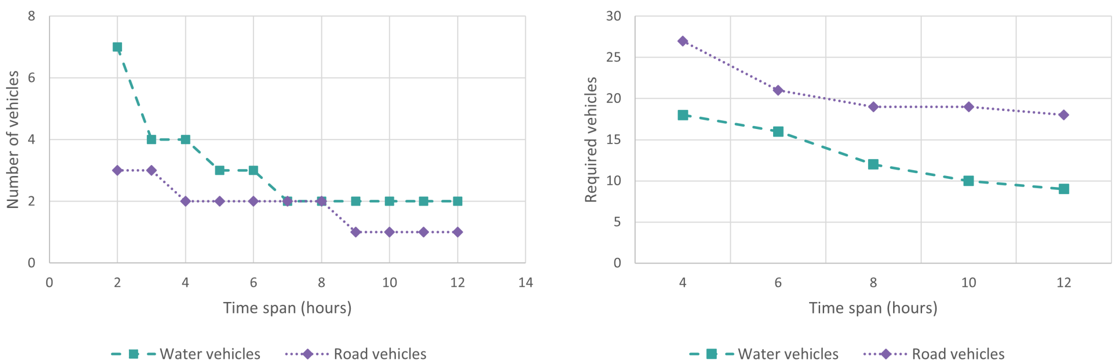
Figure 5.14: Best performing number of satellites plotted against the demand characteristics

Concluding, opening between 9 and 13 satellites is recommended to effectively supply the entire Horeca sector in Amsterdam. If a smaller city area is supplied, the number of satellites seems to decrease linearly with the total demand of that area. For the remaining full case experiments, 12 satellites are opened. For the experiments on the Wallen neighbourhood, 2 satellites are used.

5.3.2. Maximum time span

The time span in which the deliveries are performed is crucial for the IWLT system to be feasible in real-life applications. For example, the time span can be restricted due to city regulations against noise pollution. Next to this, busyness in the city centre during peak hours is best avoided, which also limits the time span. The available time impacts the system requirements to serve all customers. To see the effect on these requirements, different maximum time spans are tested and the results investigated.

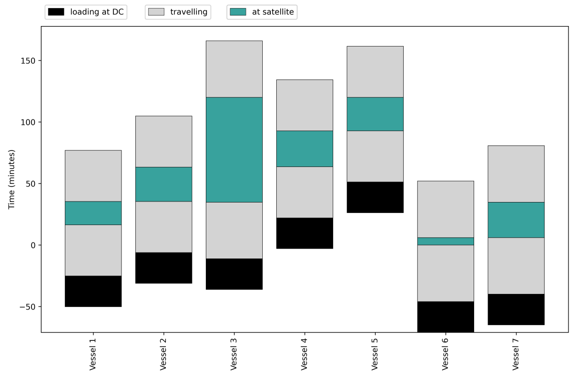
The maximum time spans ($t^{\max}$) evaluated are 4, 6, 8, 10, and 12 hours, with 12 satellites opened for the entire city center of Amsterdam, using the demand data provided by Bijvoet (2023), specified in Table 5.2 set 2. Additional experiments are conducted for the Wallen neighborhood with time spans ranging from 2 to 12 hours, in increments of 1 hour, utilising two satellites. The demand distribution for these experiments is also based on set 2 but is limited to the Horeca locations in the Wallen neighbourhood.



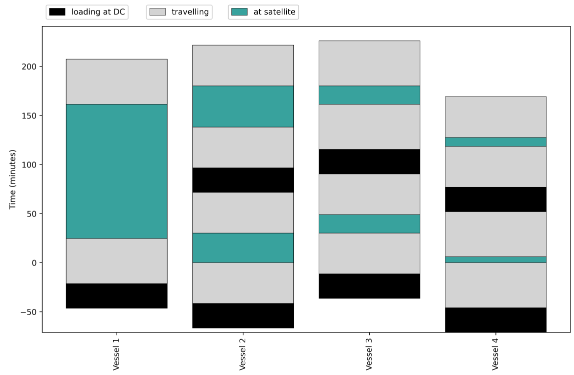
(a) Wallen neighbourhood (b) Entire city centre

Figure 5.15: Required number of vehicles for varying time spans $t^{\max}$

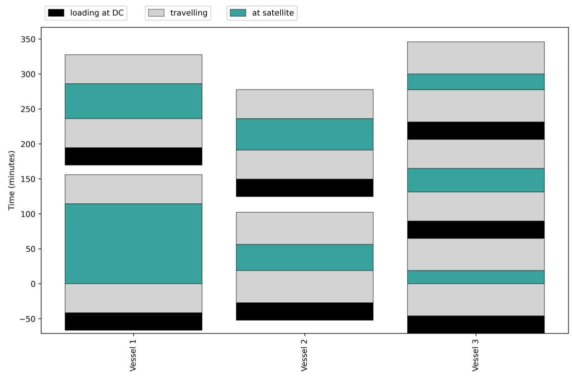
The impact of increasing the time span can best be shown through the number of vehicles required, as shown in Figure 5.15, especially for vessels. Half of the vessels are required when extending the time span from 4 to 12 hours, which is expected since vessel trips have long completion times, so with a shorter time span, vehicles are not always able to perform multiple trips. Figure 5.16 shows the vessel schedules in the Wallen neighbourhood for the time spans where the number of vessels decreases, so for 2,3,5 and 7 hours. These figures provide a clear image of the number of trips a vessel can make within the time span. In a time span of 2 hours, the vehicles can only perform one trip, while at 3 hours it is possible to perform two trips. At 5 hours, this increases to three trips, and at a time span of 7 hours, a vessel can perform four trips.



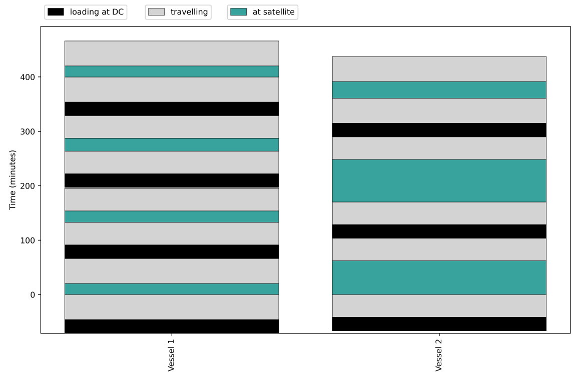
(a) $t^{\max}$ =2 hours



(b) $t^{\max}$ =3 hours



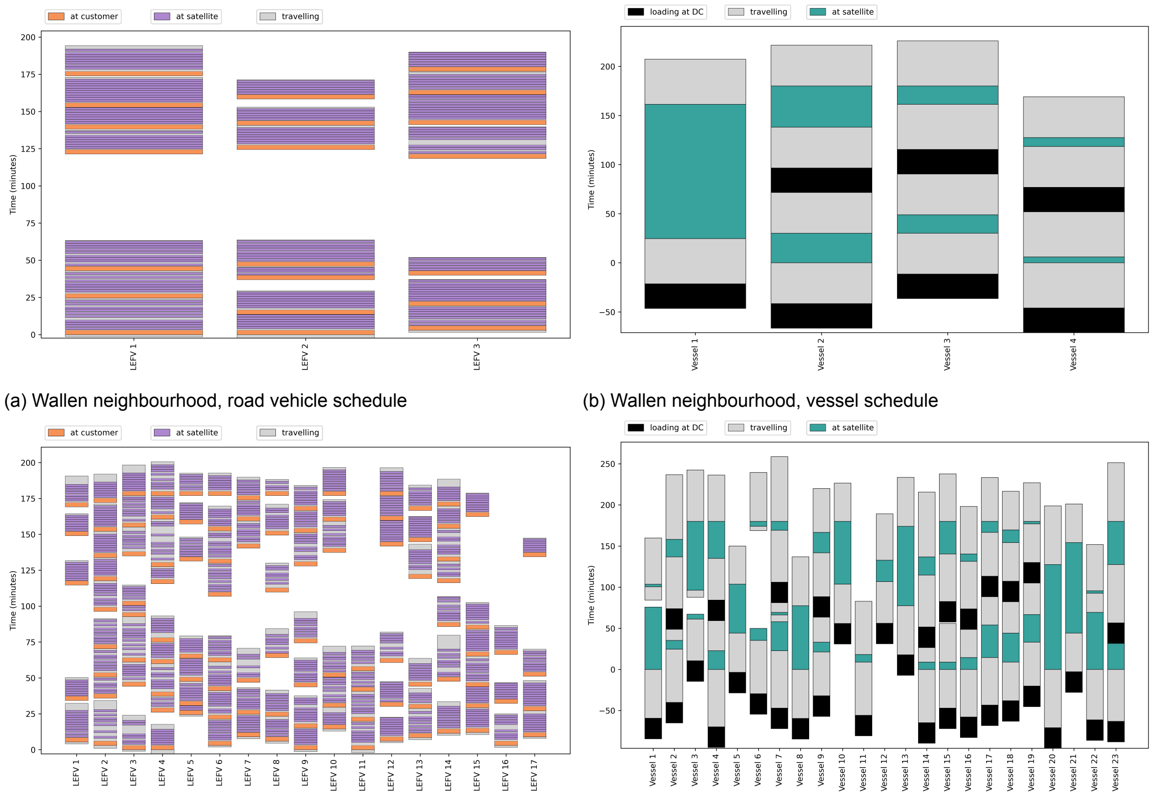
(c) $t^{\max}$ =5 hours



(d) $t^{\max}$ =7 hours

Figure 5.16: Vessel schedules Wallen case for different time spans

The decrease is also visible for road vehicles. However, the decrease is less significant. Increasing the time span from 4 to 12 hours for the full case results in 33% fewer required road vehicles. This phenomenon can be linked to the vessel schedule. Most of the vessels arrive at approximately the same time at satellites, so at that moment, many road vehicles are required as well. Figure 5.17 shows the vehicle schedules for the Wallen neighbourhood and the entire city centre with a time span of 3 hours. In this time span, vessels are able to perform two trips. The road vehicle schedules are clearly dependent on the approximately simultaneous arrival times of the vessels. All road vehicles are required at the same moments, at the start of the time period and at the arrival time of the second vessel trip. Still, when the time span increases and the vessels perform multiple trips, fewer road vehicles are required at the same moment.



(a) Wallen neighbourhood, road vehicle schedule

(b) Wallen neighbourhood, vessel schedule

(c) Entire city centre, road vehicle schedule

(d) Entire city centre, vessel schedule

Figure 5.17: Vehicle schedules for a time span of $t^{\max}$ =3 hours

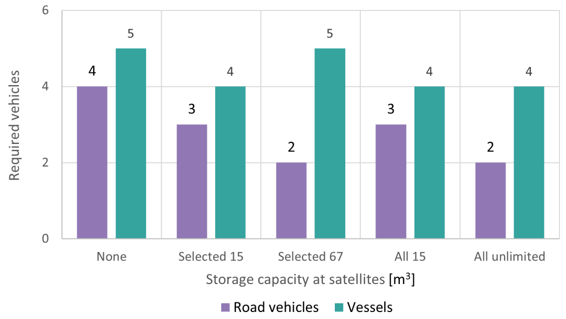
As fewer road vehicles are required to serve the customer demand, the overall distance travelled on roads decreases. This reduction is due to the inclusion of the distance from the vehicle depot to the satellites in the total distance calculation. The distance on the water does not change, since all vessel trips depart from the same depot.

5.3.3. Storage Capacity Satellites

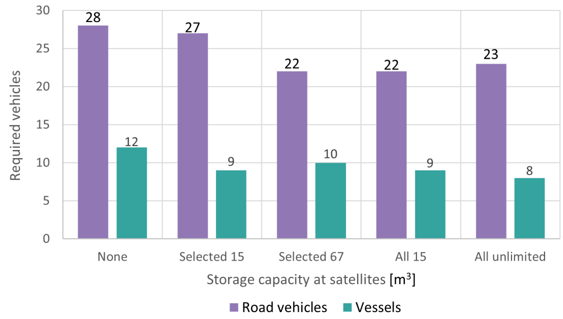
Since space is scarce in most city centres, the basic scenario investigated assumes no storage capacity at satellites. However, at certain locations, some storage might be feasible, potentially enhancing system performance, which would make it worthwhile to consider allocating storage space in city centres. To understand the impact of satellite storage on the system behaviour, various storage scenarios are evaluated.

Through field research, satellite locations with potential for storage are identified. These satellites are strategically positioned at larger waterways or docks equipped with jetties. To supply the entire city centre with 12 satellites, four of the locations show significant potential to incorporate storage facilities. In the Wallen neighbourhoods with four satellites, two of the satellites are feasible for storage.

Experiments are conducted to assess various storage capacities at these satellites: 15m^3 and 67m^3, which correspond with containers of 10ft and 40ft (2.8m and 12m). Additionally, it is interesting to see the effect on the system's performance if all satellites have storage available. For this scenario, a capacity of 15m^3 is considered since this is most viable for real-life applications. Finally, an analysis is performed under the hypothetical scenario of unlimited storage capacity at all satellites, offering insights into potential operational bottlenecks despite its infeasibility in practical implementation.



(a) Wallen neighbourhood



(b) Entire city centre

Figure 5.18: Required number of vehicles for different storage scenarios at satellites

Figure 5.18 shows the required vehicles for different storage scenarios at satellites for the Wallen neighbourhood and the entire city centre. As can be seen, having 15m^3 storage capacity at the selected satellites lowers the number of vessels, from 5 to 4 for the Wallen neighbourhood and from 12 to 9 for the entire city centre, which are significant improvements. However, for the entire city centre, it only slightly decreases the number of road vehicles from 28 to 27.

Increasing the storage capacity at selected satellites to 67m^3 results in a more efficient road vehicle schedule, but this improvement comes with a trade-off. Specifically, it increases the number of vessels required for both the Wallen neighbourhood and the entire city centre. This effect is likely due to the storage capacity of 67m^3 at the selected satellites exceeding the vessel capacity of 50m^3. Consequently, when the larger storage is utilised by the road vehicle schedule, it might necessitate more complex movements of the vessels to accommodate this utilisation.

When all satellites are equipped with a storage capacity of 15m^3, the results for the Wallen neighbourhood are identical to the scenario of 15m^3 storage at selected satellites. However, for the entire city centre, this scenario shows an improvement by reducing the required road vehicles while maintaining the same number of vessels, compared to the storage scenario of 15m^3 at selected satellites.

A further improvement is observed under the hypothetical scenario of unlimited storage capacity at all satellites, requiring only 23 road vehicles and 8 vessels. This scenario highlights the substantial impact of satellite storage capacity on the logistics network, demonstrating significant performance gains with storage. However, the most significant improvement in required vessels for the entire city centre is made when increasing the storage at the selected satellites from zero to 15m^3, indicating that having some storage available provides enough flexibility for the system to operate more efficiently.

It is important to note the potential for further improving the road vehicle schedule when storage is available at satellites. Enabling storage capacity while also allowing direct transfers significantly increases the solution space of the model. This is particularly impactful for the road vehicle schedule, given the greater number of road vehicles with smaller capacities performing numerous trips compared to vessels. With storage available, vessels must still ensure the load arrives before road vehicles pick it up. However, road vehicles can collect the load at any convenient time afterwards, greatly increasing the flexibility in scheduling. This expanded scheduling flexibility can lead to more efficient logistics operations, but also increases the solution space and, therefore, the computational complexity of the model.

5.3.4. Depot locations

The vessel depot locations are selected based on their accessibility for road transportation, as cargo is transported to the depots by trucks. These locations are informed by the work of Bijvoet (2023), conversations with municipality workers and research from the municipality. However, these depots are situated quite far from the city centre, resulting in significant travel distances on the waterways. In the scenario where the entire city centre is supplied using 12 satellites, 97% of the waterway distance is attributed to travel to and from the depots. This high percentage is due to many trips only visiting one satellite, so all of the travel distance of that trip is the journey to and from the depot. It is worthwhile to investigate the potential reduction in waterway travel distance if depots were positioned closer to the city centre.

To investigate the effect of depot placement, a scenario with depots closer to the city centre is created. Figure 5.19 shows the depot locations for this experiment. The scenario investigated is to supply the full city centre with the demand as provided by Bijvoet (2023), specified in Table 5.2 set 2.

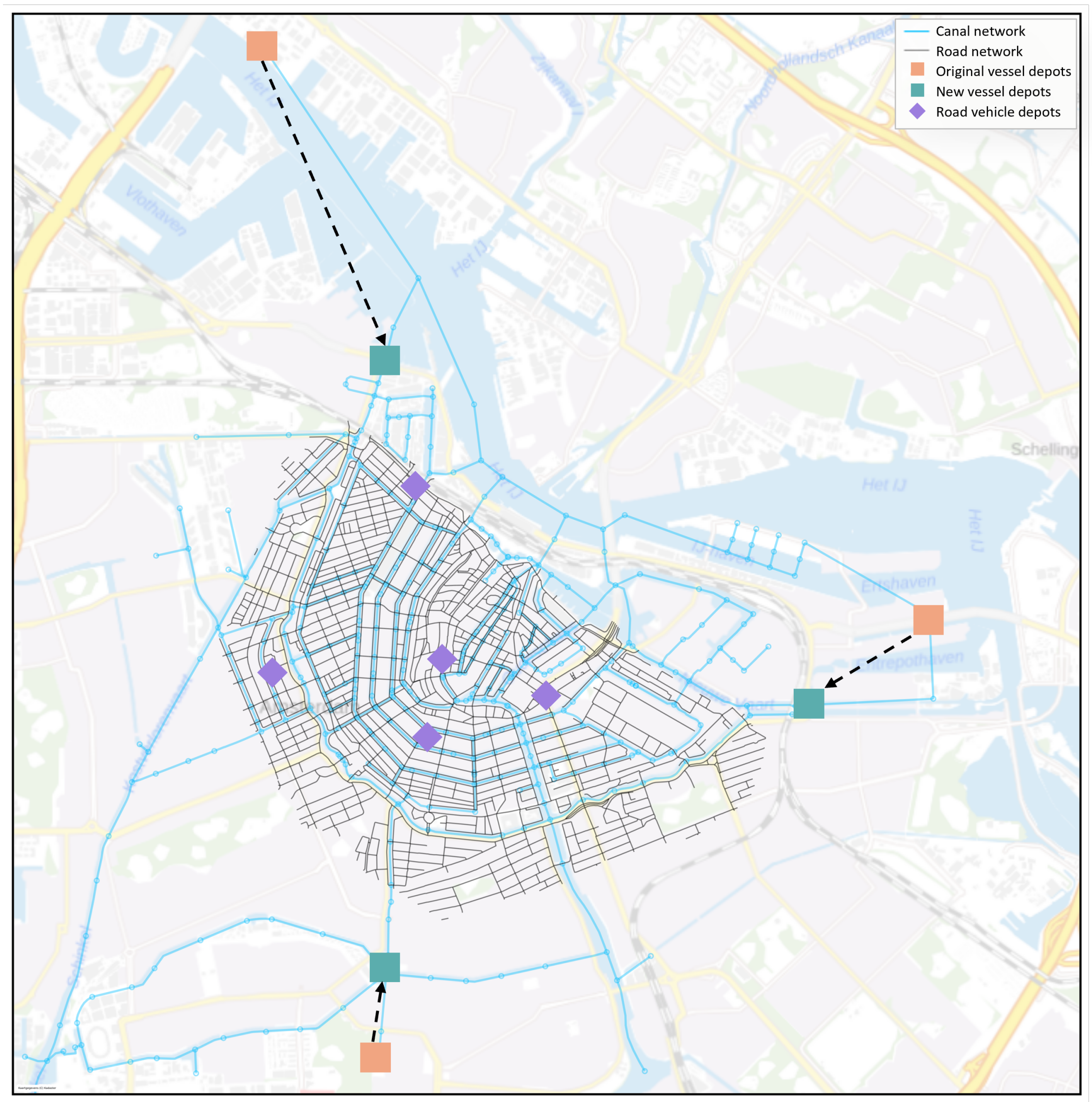
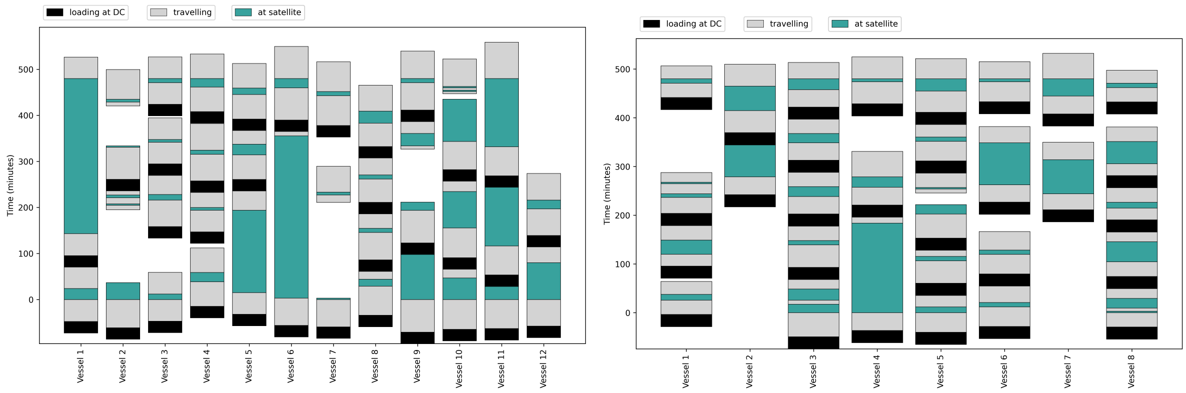


Figure 5.19: Depot locations for scenario to investigate

Figure 5.20 shows the vessel schedules for the different depot locations. The grey travelling parts directly after and before the black loading at DC parts, indicate travelling from and to the depots. As expected, these travel times are significantly smaller for the new depot locations.



(a) Original depot locations

(b) New depot locations

Figure 5.20: Vessel schedules for the original depot locations and the new depot locations

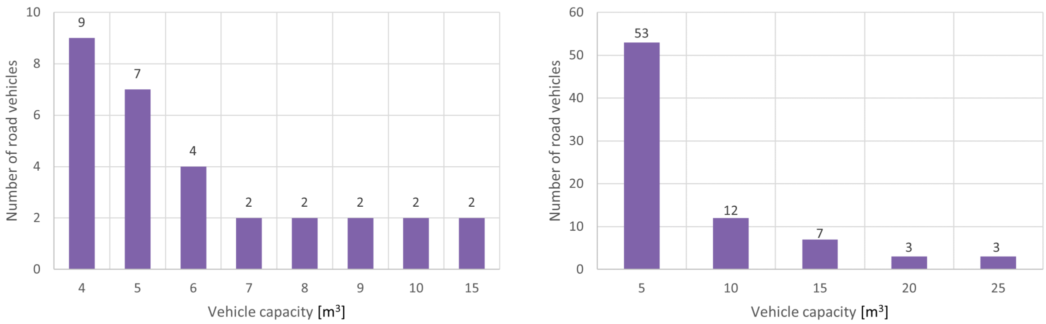
The distance travelled on the waterways is reduced from 273km to 199km, which is a reduction of 27%. The contribution of the distance travelled from and to the new depot locations is 86% of the total travel distance. On top of that, due to the reduced travel time, the number of vessels required reduces from 12 to 8, resulting in a reduction of 33%. These are significant reductions, making it worthwhile to investigate the possibility of placing depots closer to the city centre. However, the difficulties of supplying depots closer to the city centre have to be investigated, in terms of added distances and travel time for trucks.

For road vehicle depots, their location has a smaller impact on the total distance travelled. The distance travelled from and to the depots contributes approximately 10% to the total distance on the roads. This contribution is much smaller than that of the vessel depots because road vehicles do not need to return to their depot between each trip. Nonetheless, if road vehicles could park on streets adjacent to the satellites, a reduction of 10% in travel distance could be achieved.

5.3.5. Road Vehicle Characteristics

The capacities of the road vehicles are determined by internet research for vehicles that comply with the regulations in the city centre of Amsterdam. It is interesting to see how the system behaves for different vehicle capacities since it might be desired to have a different fleet composition and regulations can change. The standard capacity used is $q^{\mathrm{V}} = 15m^3$. The experiments for the Wallen neighbourhood examine road vehicle capacities of 4 to 15m^3. For the entire city centre, capacities of 5, 10, 15, 20 and 25m^3, are tested. For the Wallen neighbourhood, demand set 2 is used, while for the entire city centre, demand set 1 is employed. This is due to the fact that smaller road vehicle capacities significantly increase the vehicle set, which increases the computational complexity. Demand set 1 for the entire city centre serves 506 Horeca locations with a demand of $988\mathrm{m}^3$.

Figure 5.21a shows the required number of road vehicles for the Wallen neighbourhood with different vehicle capacities and Figure 5.21a gives these results for the full case.



(a) Wallen neighbourhood, demand set 2

(b) Entire city centre, demand set 1

Figure 5.21: Required number of road vehicles for varying road vehicle capacities

From these bar charts, it can be seen that for both the Wallen neighbourhood and the entire city centre, logically, a clear trend reveals; as the capacity of the road vehicles increases, the number of required road vehicles decreases. For the Wallen neighbourhood, a capacity of $7\mathrm{m}^3$ is enough to achieve the minimum number of 2 road vehicles.

When considering the entire city centre, initially, with a capacity of $5\mathrm{m}^3$, 53 vehicles are necessary. Doubling the capacity to $10\mathrm{m}^3$ results in a substantial reduction, with only 12 vehicles required. Further increases to $15\mathrm{m}^3$, $20\mathrm{m}^3$, and $25\mathrm{m}^3$ continue to decrease the vehicle count to seven, three, and three, respectively. This diminishing return beyond $15\mathrm{m}^3$ suggests that larger capacities significantly alleviate the need for more vehicles, but additional capacity beyond this point offers less of a reduction.

From a strategic planning perspective, these insights are valuable. They suggest that investing in vehicles with capacities around $10\mathrm{m}^3$ to $15\mathrm{m}^3$ may offer the best balance between reducing vehicle numbers and maintaining operational efficiency. In densely packed areas like the Wallen neighbourhood, even modest increases in vehicle capacity can have a notable impact on the fleet size required, reducing operational costs. For the Wallen neighbourhood, it would be advantageous to use the smallest vehicles that achieve the minimum number of two required vehicles, specifically those with a capacity of $7\mathrm{m}^3$. Smaller vehicles are better suited for the city centre due to their improved maneuverability and ease of navigating narrow streets and tight spaces, which are common in densely populated urban. This approach balances efficiency with practicality, ensuring that deliveries are conducted smoothly while minimising traffic congestion.

5.4. Sensitivity Analyses

Understanding how the system responds to different parameter values or demand sets is crucial. Sensitivity analyses are performed to explore these variations. Examining how the system behaves under different conditions provides insights into design choices for implementation, the system's limitations and can help identify areas for improvement.

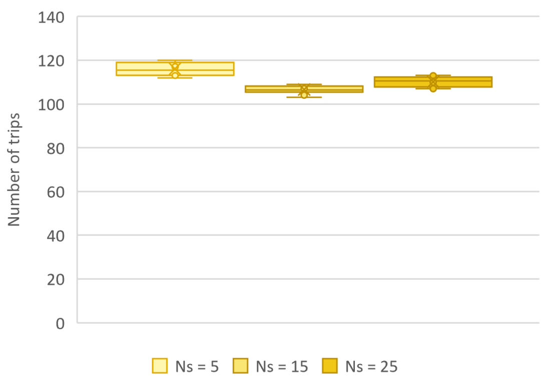
5.4.1. Demand sets

As discussed in Subsection 5.1.2, the ten basic demand sets are fairly similar. Some runs are performed for the ten basic demand sets, to determine if the results differ significantly. It is most important to investigate the required number of vehicles and the time period to perform the deliveries. Next to this, the kilometres on the road and canals are examined. The results of the ten demand sets are evaluated for 5, 15 and 25 satellites.

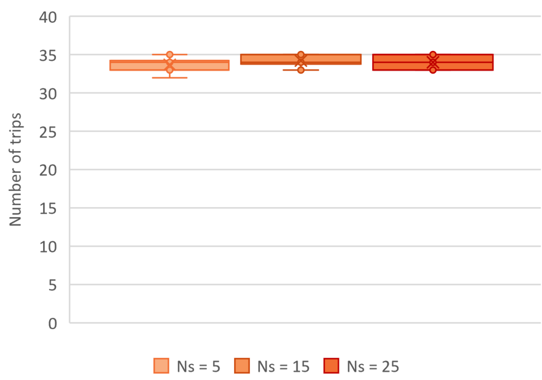
The kilometres travelled on the road have on average a 1.5% deviation per demand set, for the kilometres on the canals this is on average 2.6%. The number of trips performed by road vehicles have an average deviation of 2.1 trips, which is a 1.9% deviation. For the number of trips performed by vessels, an average deviation of 0.67 trips is found, which is a 2.0% deviation. The maximum difference in road vehicle trips is 8 trips, which is 6.9% of the average number of road vehicle trips required with

5 satellites.

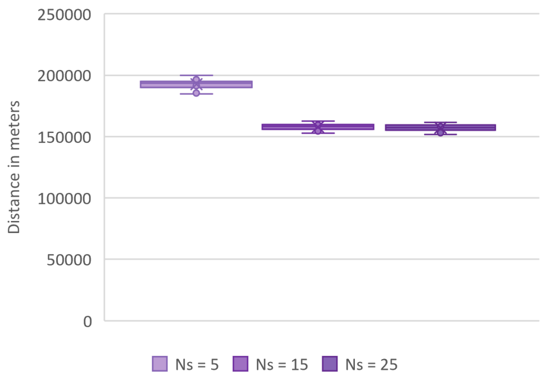
Figure 5.22 shows the distributions of the results per number of satellites for the different demand sets. These results do not show significant differences for the demand sets, since the added number of trips are small and will not result in more required vehicles.



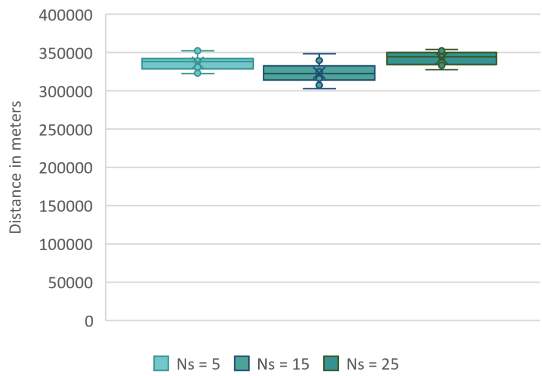
(a) Road vehicle trips



(b) Vessel trips



(c) Distance on the roads



(d) Distance on the waterways

Figure 5.22: Distributions of results for different demand sets

In addition to the basic demand sets, some more extreme demand sets are created and tested to evaluate the system's adaptability, as explained in Subsection 5.1.2, a quick overview of the sets is given in Table 5.3.

Table 5.3: Overview of the demand sets

Demand set	Total demand $[m^3]$	Customers with demand
1	988	506
2	1498	750
3	1952	971
4	2502	1240

The required number of vehicles for each demand set are shown in Figure 5.23. The number of water and road vehicles increases approximately linearly with the size of the demand sets.

Table 5.4: Demand probability distribution per demand set

	Demand $[m^3]$					
Demand set	0	1	2	3	4	5
5		33%	33%	33%		
6	20%	20%	20%	20%	20%	
7		100%				
8	50%					50%

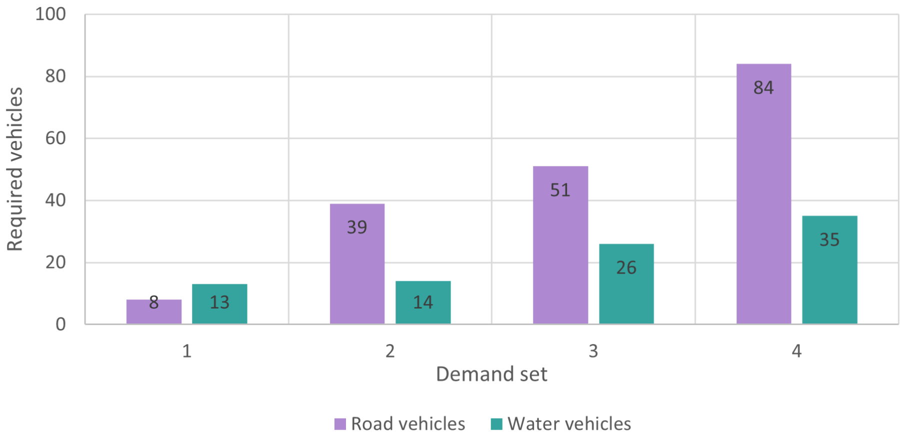


Figure 5.23: Number of required road and vessels for different demand sets

To get better insight in the influence of the demand and number of customers with demand, additional experiments are performed on the Wallen case. The additional demand sets are defined in Table 5.4.

The demand sets defined in Subsection 5.1.2 and in Table 5.4 result in the total demand and customers with demand given in Table 5.5 for the Wallen case.

Table 5.5: Overview of the demand sets

Demand set	Total demand $[m^3]$	Customers with demand
1	220	114
2	290	151
3	428	212
4	541	263
5	679	345
6	687	279
7	345	345
8	870	174

Figure 5.24a and Figure 5.24c show the results for the different demand sets with on the x-axis the number of customers with demand. Figure 5.24b and Figure 5.24d show the same results, but with the total demand on the x-axis. As can be seen, the relation between the total demand and the results for the distances and number of vehicles is almost linear, while the number of customers has a less visible relation with the results.

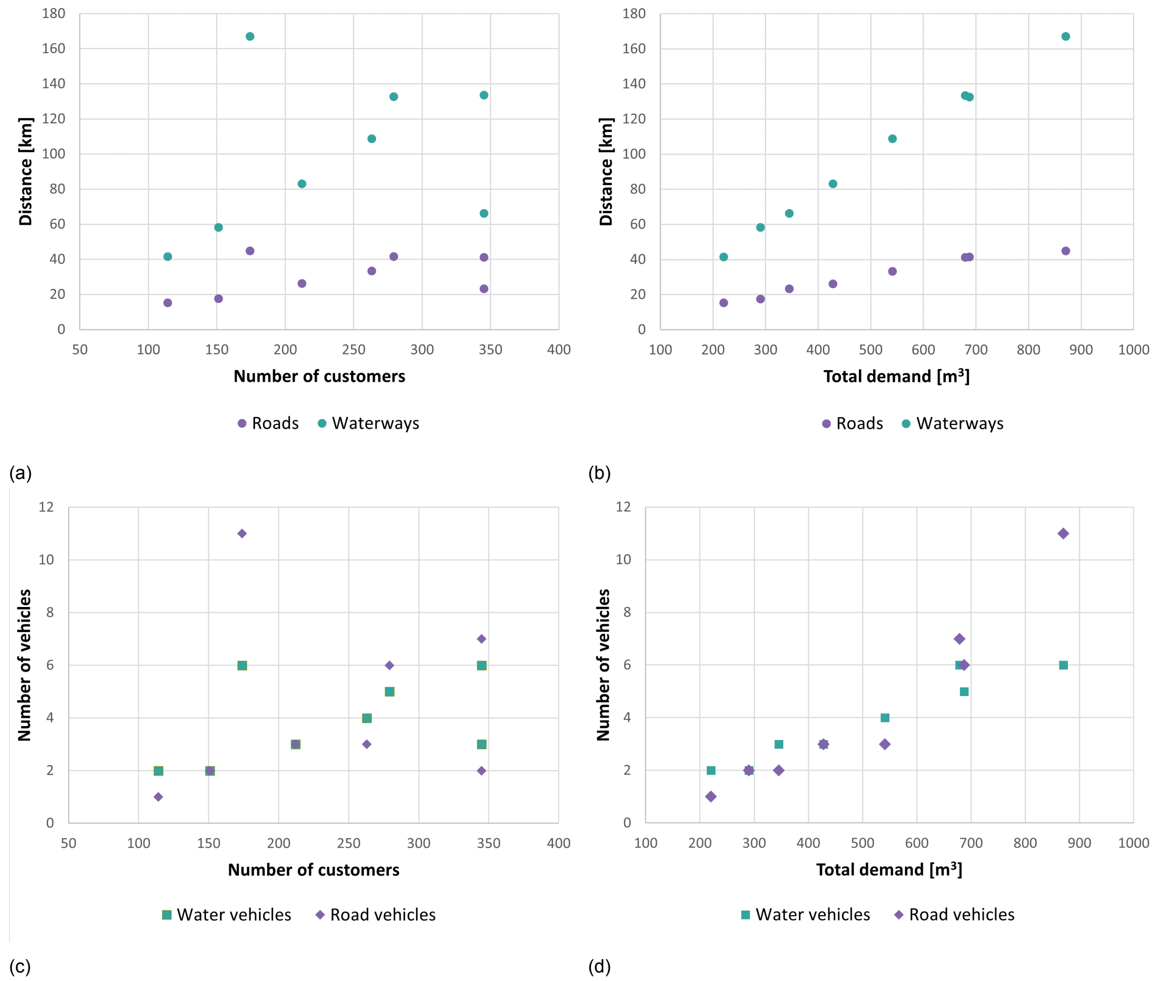
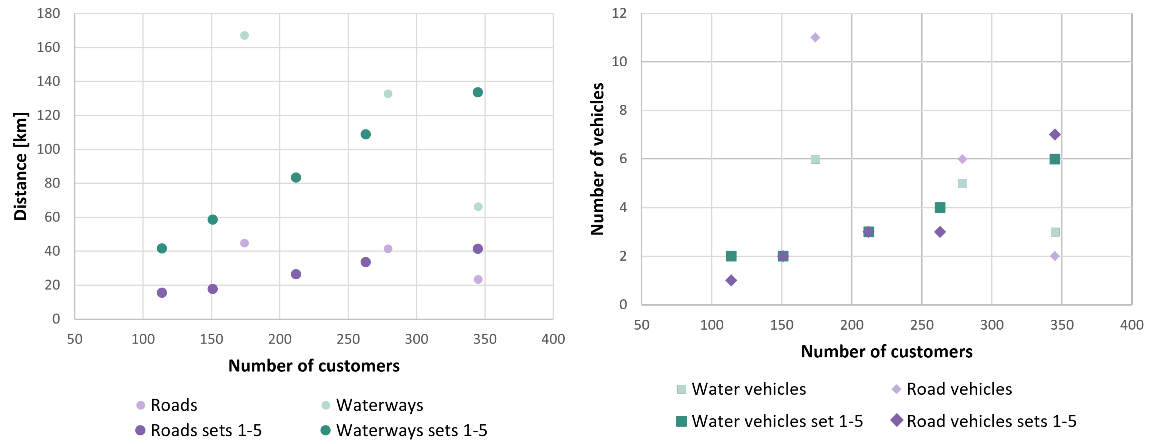


Figure 5.24: Results for the demand sets in the Wallen neighbourhood

Demand sets 1 to 5 have a consistent distribution of demand, ranging from 1 to 3 m^3 per Horeca location, with varying probabilities of zero demand. Exploring the relationship between the number of customers and the outcomes for these sets yields valuable insights.Figure 5.25 mirrors the results of Figure 5.24 plotting the results against the number of Horeca locations with demand and highlighting the results for demand sets 1-5. These findings do suggest a linear relation between the number of Horeca locations and the outcomes. However, it's essential to note that this conclusion may not hold true for demand sets with dissimilar distributions. In Subsection 5.3.1 a linear relation was found between both the number of customers and the total demand concerning the outcomes for the number of satellites. The linear relation for the number of customers can now be attributed to the uniform distribution of demand across Horeca locations within the sets evaluated for the number of satellites. However, the relation between the total demand and the results is further validated.



(a) (b)

Figure 5.25: Results for the demand sets in the Wallen neighbourhood plotted against the number of customers, with demand sets 1-5 highlighted

The linear relationship between the total demand and the distances travelled indicates a predictable pattern in how demand affects distances. It suggests that the model is robust and reacts predictably to changes in demand, which is a desirable property for any decision-making tool. This robustness builds confidence in the model's use for real-life applications.

5.4.2. Transshipment Times

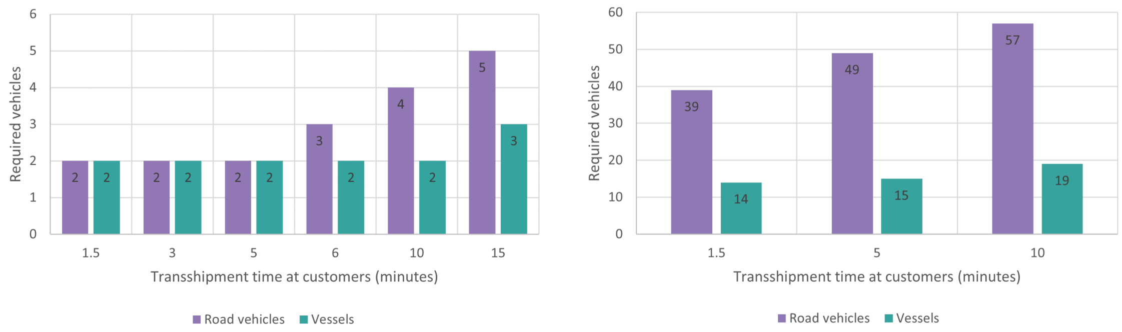
The transshipment time at customers constitutes a significant portion of the road vehicle schedule and often exceeds travel time in terms of duration. The transshipment times used in the other experiments are based on the work of Bijvoet (2023). However, it is worth noting that these times seem to be optimistic and may not accurately reflect real-world scenarios.

Given the potential for variability and uncertainty in transshipment processes, conducting sensitivity analysis is important. This sensitivity analysis involves testing the IWLT system requirements under different transshipment times at customers. The analysis evaluates the system's sensitivity to changes in transshipment times and helps identify thresholds where performance may be significantly impacted.

Figure 5.26 shows the required vehicles for supplying the Wallen neighbourhood and the entire city centre for transshipment times at customers. The number of required road vehicles increases for longer transshipment times, however, more than tripling the transshipment time of $t^{\mathrm{C}} = 1.5min$ in the entire city centre only requires 26% more road vehicles. Furthermore, doubling the transshipment time of $t^{\mathrm{C}} = 5min$ only increases the road vehicle requirement by 16%. For the Wallen case, no increase in the number of vehicles is required for transshipment times up to 5 minutes.

This phenomenon can be explained by investigating Figure 5.27, which shows the road vehicle schedules for the Wallen case with transshipment times $t^{\mathrm{C}} = 1.5min$ and $t^{\mathrm{C}} = 5min$. Road vehicles have a lot of idle time when waiting for vessels to arrive. Therefore, the extra transshipment time can be added to the road vehicle trips without requiring extra vehicles. However, for the customer transshipment time of 5 minutes, the road vehicles are almost fully utilised. Therefore, a transshipment time at the customers of $t^{\mathrm{C}} = 6min$ does require an extra road vehicle.

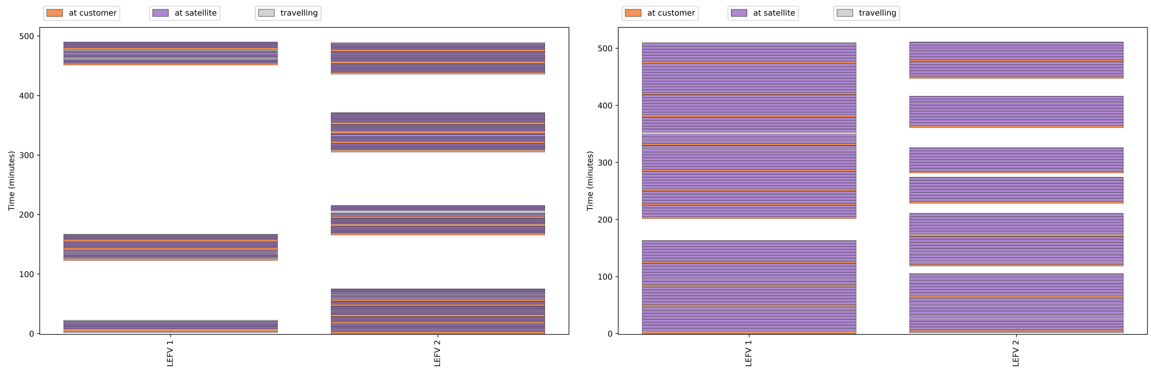
Increasing the transshipment time at customers does also affect the number of vessels required, however less significant. Since road vehicles have longer trip times, vessels might have to wait longer at satellites, which can ultimately results in more required vessels.



(a) Wallen neighbourhood

(b) Entire city centre

Figure 5.26: Required vehicles for different transshipment times at customers



(a) Road vehicle schedule for $t^{C} = 1.5min$

(b) Road vehicle schedule for $t^{C} = 5min$

Figure 5.27: Road vehicle schedules for different transshipment times at customers

With these results, the system does not appear to be overly sensitive to variability in customer transshipment times. When the road vehicles are not fully utilised, the increased transshipment times can be accommodated. When the transshipment time is increased further, a linear relation between the required number of road vehicles and increased time seems to exist.

5.5. Overall system performance

Based on the experimental analysis, it is essential to evaluate how the IWLT system performs compared to the current situation. Leveraging insights from the experiments, four system scenarios are selected to assess performance, identify bottlenecks, and compare the results with the current state. The scenarios represent various combinations of the key design choices.

The scenarios for the number of satellites are selected based on bounds that show efficient coverage for the entire city centre. The system's performance improves significantly with up to 9 satellites. From 9 to 13 satellites, performance remains relatively stable, with peak efficiency at 12 satellites. The selected scenarios include 9 and 12 satellites: 9 for being the minimum number with strong performance and 12 for achieving the highest efficiency.

Regarding the time span, a clear decreasing trend is observed in the number of vehicles needed as the time span increases. This indicates that a 12-hour time span scenario will perform better than a 4-hour one. However, since the trend is continuous, no specific range of time spans can be identified as optimal. Therefore, scenarios with varying time spans are chosen: 4, 8, and 12 hours.

Having storage at the satellites enhances system performance. However, for practical applications, it is valuable to compare realistic storage scenarios. The selected storage capacity scenarios are: no storage, $15\mathrm{m}^3$ at selected satellites, and $15\mathrm{m}^3$ at all satellites.

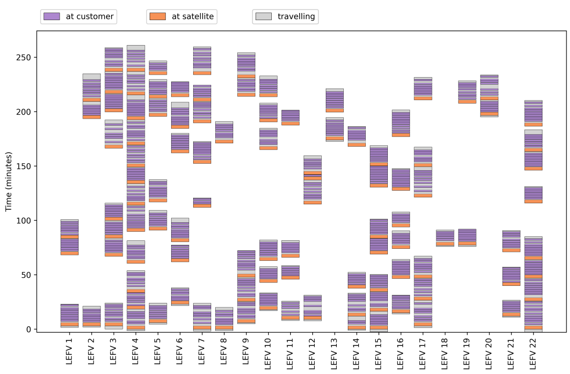
The scenarios for these design choices are combined to create four distinct scenarios: the expected lowest-performing plausible scenario (A), a baseline realistic scenario (B), an enhanced realistic scenario (C), and the expected best-performing scenario (D). These combinations are shown in Table 5.6.

Table 5.6: Selected scenarios for performance evaluation

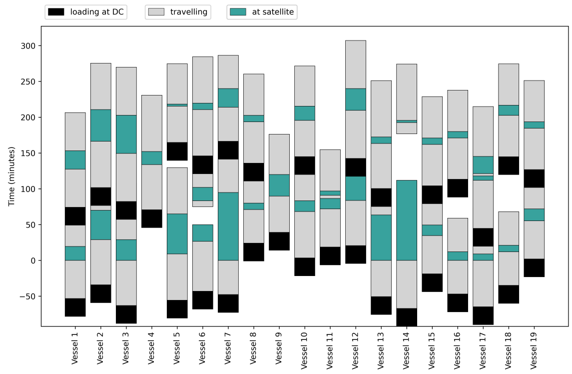
Scenario	Number of satellites	Time span (hours)	Satellite storage
A	9	4	None
B	12	8	None
C	12	8	$15\mathrm{m}^3$ selected four
D	12	12	$15\mathrm{m}^3$ all

For these scenarios, the model is solved with more allocated computational resources, specifically by allocating more CPUs and tasks, to ensure a comprehensive and accurate comparison of the IWLT system with the current situation. The FLP strategy used is to limit the number of customers with parameter $b = 1.5$, and the demand follows the distribution of set 2 Table 5.2. The parameters defined in Subsection 5.1.3 do not change unless specified in Table 5.6.

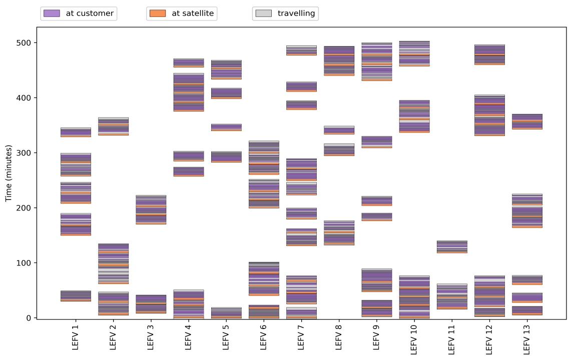
To obtain insights into the performance and bottlenecks for each scenario, it is useful to explore the vehicle schedules. Figure 5.28 shows the schedules for the selected scenarios.



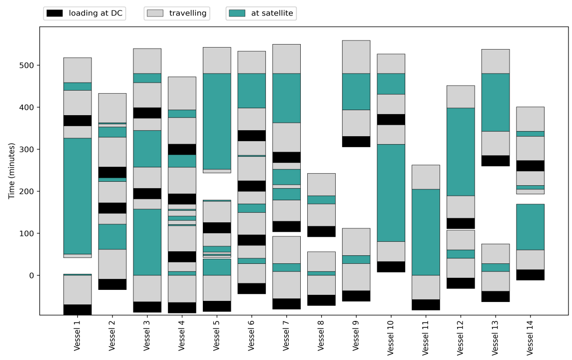
(a) Scenario A: road vehicle schedule



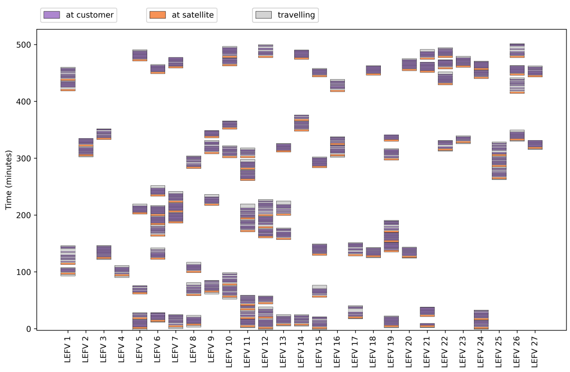
(b) Scenario A: vessel schedule



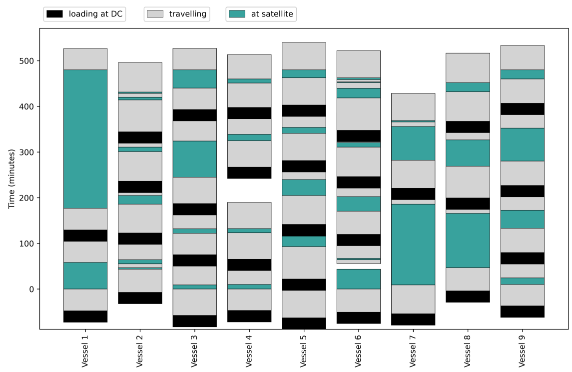
(c) Scenario B: road vehicle schedule



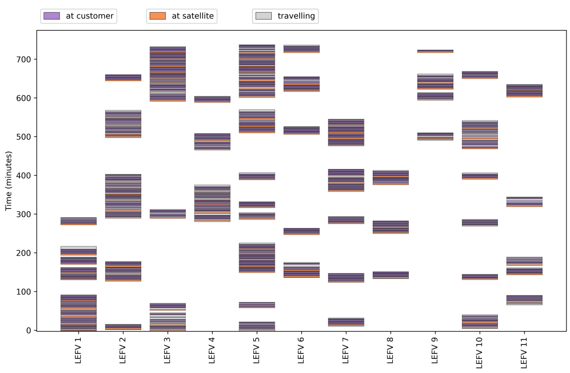
(d) Scenario B: vessel schedule



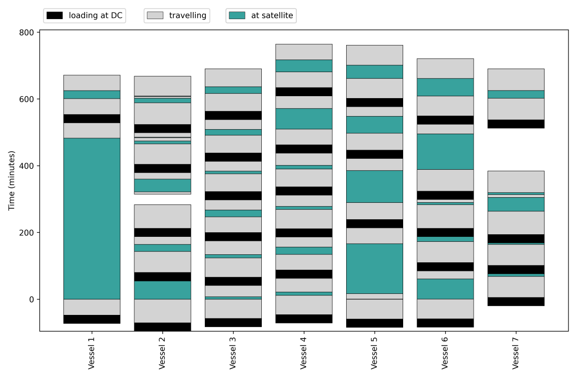
(e) Scenario C: road vehicle schedule



(f) Scenario C: vessel schedule



(g) Scenario D: road vehicle schedule



(h) Scenario D: vessel schedule

Figure 5.28: Vehicles schedules for the selected system scenarios

Table 5.7 shows the results for the selected scenarios. As expected, scenario C has the best performance, with only 11 road vehicles and 7 vessels required to supply the Horeca in the entire city centre. The distance travelled on the roads is equal for scenarios B and C, which can be attributed to the equal number of satellites. The waterway distance reduces for scenarios with storage capacities, as expected.

Table 5.7: Results for selected scenarios

Scenario	Road kilometres	Water kilometres	Road vehicles	Vessels
A	172	278	22	19
B	166	273	13	14
C	163	260	27	9
D	163	252	11	7

For scenario A, the simultaneous arrival of vessels at satellites creates a bottleneck, hindering the reduction of road vehicles. Nearly all road vehicles are required at the beginning of the time span, as displayed in Figure 5.28a. Introducing storage capacity at satellites could alleviate this bottleneck. Another option is to impose constraints on the departure times of vessels, though this may negatively impact the vessel schedule.

Scenario B performs significantly better than scenario A, decreasing the number of road vehicles from 22 to 13 and the number of vessels from 19 to 14. The difference between scenario A and B is the time span, which is increased from 4 to 8 hours. The experiments in Subsection 5.3.2 demonstrate a 33% reduction in required vessels and a 30% reduction in road vehicles when the time span is increased from 4 to 8 hours. For the investigated scenarios A and B, the increased time span results in a 26% reduction of vessels and 41% of road vehicles.

Scenario C performs significantly better than scenario A and B in terms of the required number of vessels. This improvement compared to scenario A is partly due to the increased time span. Additionally, introducing storage capacity at satellites contributes to reducing the number of required vessels, as detailed in Subsection 5.3.3. Implementing a storage capacity of $15\mathrm{m}^3$ at 4 out of the 12 satellites resulted in a 25% reduction in the required number of vessels. These factors combined lead to a significant reduction in the number of required vessels in scenario B compared to scenario A, amounting to an overall reduction of 53%.

On the contrary, the number of road vehicles increases for scenario C compared to both scenarios A and B. As explained in Subsection 5.3.3, this can be attributed to the increased solution space when storage is introduced at satellites and, therefore, the reported solution might not be an accurate solution. However, it is expected that scenario C should be viable with the same number of road vehicles as scenario B, since the road vehicle schedule is more flexible due to the available storage. Experiments conducted for the Wallen neighbourhood even showed a reduction of 25% for the number of road vehicles when introducing $15\mathrm{m}^3$ storage at selected satellites, indicating scenario C could potentially operate with fewer road vehicles than scenario B.

Despite the higher number of road vehicles, the total road distance travelled in scenario C is lower than in scenario B. This improvement is due to the availability of storage at selected satellites, which allows road vehicles to make shorter, more efficient trips by collecting loads from nearby satellites, even when no vessel is currently docked at those satellites.

In scenario D, the vessel schedule is nearly fully utilised, as shown in Figure 5.28h. However, some vessels experience long waiting times at satellites. This is undesirable for real-life applications as it means a vessel occupies a dock for extended periods. To address this, reducing waiting times could be included in the objective function, encouraging the model to optimise accordingly. The road vehicles are less efficiently utilised, with significant idle time for each vehicle, as shown Figure 5.28g. The schedule could be further improved; for instance, the trips of LEFV 8 and LEFV 9 could be merged without issues. Again, this suboptimal utilisation of road vehicles can, to a certain extend, be attributed to the model's large solution space. Further iterations could enhance the schedule.

The road distances are identical for scenarios C and D, and only differ from scenario B by three kilometres. This is expected given that road distance is primarily influenced by the number of satellites. However, the distances on water decrease more significantly for scenario C and D, attributed to more efficient routing made possible by having storage capacity at all satellites.

In the current situation all deliveries are conducted via road transport. This situation represents the existing scenario and is modelled as a straightforward vehicle routing problem with capacity constraints. A single depot is placed at the city's border, ensuring that only the distances travelled within the city centre are considered. The vehicle characteristics are consistent with those used in the IWLT system, with a capacity $q^V = 15m^3$ and speed $v^V = 5m/s$.

Table 5.8 presents the distances travelled for both the current situation and the selected IWLT system scenarios. The IWLT system scenarios result in vehicle kilometres reductions of 22%, 24%, 27% and 28% compared to the current situation, for scenario A, B, C and D, respectively. These reductions are a positive step, but the primary goal of the IWLT system is to minimise distance on the roads. All three scenarios accomplish this goal with substantial reductions, 70% for scenario A, 71% for scenario B and 72% for scenario B and C, signifying major improvements over the current situation.

Table 5.8: Distance travelled on the roads and canals for the current situation and the IWLT system scenarios

Scenario	Road kilometres	Water kilometres	Vehicle kilometres
Current situation	579	X	579
A	172	278	450
B	166	273	439
C	163	260	423
D	163	252	415

5.6. Summary and Conclusions

This chapter aims to answer the question: *What is the performance of the proposed IWLT system under different scenarios of interest?* The IWLT system demonstrates significant potential for enhancing urban logistics, particularly in densely populated city centres like Amsterdam. Various experiments were conducted to evaluate the performance of different IWLT system scenarios. The experiments reveal several crucial insights into the system's performance under these scenarios, which can assist in implementation and further development.

In such IWLT systems, it is important to balance the workload at satellites. To do so, two methods are evaluated to limit the number of customers assigned to satellites: one based on maximum customers (factor b) and the other on maximum throughput (factor a). Experiments indicated that allowing more customers to be assigned to satellites results in fewer kilometres travelled on the roads, and a factor of $b = 1.5$ times the evenly divided number of customers per satellite provided a balance between optimal assignments and even distribution of satellite utilisation.

An important decision variable for implementing IWLT systems is the number of satellites. Experiments investigating the system's performance for varying numbers of satellites were conducted. The best performing scenarios were found to have between 9 and 13 satellites for the entire Horeca sector in Amsterdam. Beyond 13 satellites, the system performance declined due to sub-optimal customer assignments and increased vehicle travel. Experiments with smaller customer sets indicated that the best performing number of satellites decreased linearly with the total demand. For a smaller city area like the Wallen neighbourhood, fewer satellites (2-4) performed most efficient, considering different demand sets.

Another factor in the system's performance is the time span allowed for transshipment operations. Experiments show that extending the maximum time span significantly reduces the number of vehicles required. Longer time spans enable vessels to perform multiple trips, consequently lessening the peak load on road vehicles. Yet, in practice, it might be difficult to have long time spans for such operations within the urban areas.

The analysis of different storage scenarios at satellites for both the Wallen neighbourhood and the entire city centre reveals several key insights. Introducing storage capacity at satellites significantly reduces the required number of vessels, with a reduction of 25% found for $15\mathrm{m}^3$ storage at four selected satellites for the entire city centre. The results indicate that having some storage available provides enough flexibility for the system to operate more efficiently. Increasing the storage capacity can improve the performance, but shows less significant improvements.

For the considered IWLT system, the vessel depots are located quite far from the city centre, ensuing long distances to and from the depots, and thus long travel times for the vessels. The effect of closer depot placement was inspected. Reductions of 27% in distance on the waterways and 33% in required vessels were found.

The performance of the IWLT system largely depends on the capacity of the road vehicles, since smaller capacities necessitate more trips, consequently increasing the distance travelled on the roads and the number of road vehicles required. The experiments show a substantial reduction in road vehicles increasing the capacity from $5\mathrm{m}^3$ to $10\mathrm{m}^3$ and further improvements for up to $15\mathrm{m}^3$. Additional capacities reduce the number of road vehicles but offer less significant reductions.

The system's behaviour under varying demand distributions was investigated, to analyse the scalability and sensitivity. The total demand has an approximately linear relation with the required number of vehicles and the distances travelled. Higher demand naturally necessitates more resources but follows a predictable pattern. The number of customers with demand shows a less clear relationship with system performance, highlighting that total demand volume is a more critical factor than the number of customers.

Furthermore, the sensitivity to transshipment times was analysed. Increased transshipment times at customer locations result in a higher number of required road vehicles. However, the system shows resilience up to a point, accommodating increased transshipment times without a proportional increase in vehicle requirements. There is a minor increase in the number of required vessels with higher transshipment times, attributed to longer waiting times at satellites.

The insights obtained from the experiments were combined to create four distinct IWLT system scenarios. The performance of the four scenarios was compared with the current situation, where all deliveries are conducted via road transport. Substantial reductions in vehicle kilometres of 22% to 28% were found, depending on the scenario. The road distance was reduced by 70% to 72% compared to the current situation. To accomplish these reductions, the system requires 11 to 22 road vehicles and 7-9 vessels, depending on the time span and storage capacities of the scenario.

All in all, the IWLT system results significant reductions in total vehicle kilometres. While this reduction is a promising result, the shift of a significant portion of the transportation burden to waterways is a strategic advantage, leveraging the underutilised canal network in Amsterdam. A 70% to 72% reduction in road kilometres is found compared to the current situation, which aligns with the system's primary objective of reducing the burden on the roads.

6 Conclusions & Recommendations

The development of integrated water- and land-based transportation systems necessitates numerous design decisions. However, existing decision models often lack consideration for practical applications and synchronisation, particularly when dealing with large-scale problem instances, as described in Section 2.3. Therefore, this thesis aims to answer the research question: *What is the potential of integrated water- and land-based inland transportation systems to improve city logistics towards liveable cities?* This question is answered by first investigating relevant IWLT systems and the associated design choices. Next, current state-of-the-art decision models are considered. With this knowledge, the modelling approach for the decision model is developed. Lastly, the performance of different IWLT system scenarios is investigated and the potential of the IWLT system is evaluated.

In Section 3.2 the problem is defined as a two-echelon multi-trip location routing problem with satellite synchronisation (2E-MTLRP-SS), incorporating capacitated vehicles, multiple depots and a global time window, with a possibility of satellite storage. The approach used to develop the decision model for this problem is given in Section 3.3 and consists of decomposing the problem in a facility location problem, second-echelon vehicle routing problem, first-echelon vehicle routing problem and scheduling problem. The vehicle routing and scheduling problems are further divided into multiple sub-problems. In Chapter 4, metaheuristic are developed for each problem, interconnected through synchronisation in time, space, and load, which facilitates the resolution of large-scale problem instances.

The results obtained for the case of Amsterdam provide realistic estimates for the required number of vehicles and demonstrate that the IWLT system is feasible for implementation in Amsterdam. Furthermore, the results indicate that the proposed IWLT system could significantly reduce the burden on the road by utilising waterways, thus decreasing urban traffic and associated environmental, societal, and economic aftereffects. This thesis investigates several practical considerations for implementing IWLT systems in urban logistics.

The experiments conducted on the Wallen neighbourhood offer valuable insights. Given the significant investment required to implement an IWLT system, it may be prudent to start with a smaller, more focused system targeting a critical area of the city centre. For instance, supplying the ”Wallen” area, which includes 345 Horeca locations, demands substantially fewer resources than servicing the entire city centre. A system with just two vessels, two road vehicles, and two satellites is sufficient to meet the demands of this area.

Combining the results of the performed experiments, four distinct IWLT system scenarios to supply Horeca in the entire city centre were created and evaluated. Comparing the performance of these scenarios with the current situation, where all deliveries are conducted via road transport, the IWLT system scenarios achieve substantial reductions in distances on the roads. Specifically, vehicle kilometres are reduced by 22% to 28%, depending on the scenario. The primary objective of minimising road distance is successfully accomplished, with potential reductions of 70% to 72% compared to the current situation.

These findings suggest that the IWLT system shows great potential for a more efficient urban logistics operation, reducing traffic congestion and environmental impact. The system’s performance improves with longer operational time spans and shows resilience to variations in demand and transshipment times. This makes it a viable option for cities looking to optimise their logistics networks.

In summary, the developed decomposition based decision model is capable of handling complex, large-scale problem instances and provides feasible solutions for real-life applications. It offers valuable insights for logistics service providers and urban planners, facilitating the development of efficient, sustainable transportation systems that contribute to the goal of making cities more livable. This research demonstrates the potential of IWLT systems to significantly improve urban logistics, with specific recommendations for implementation in Amsterdam.

The remainder of this chapter provides recommendations based on the results of this study. It aims to bridge the gap between theoretical models and practical implementation by offering recommendations tailored to the unique logistical needs of Amsterdam. By addressing these aspects, the chapter aims to inform and guide the municipality of Amsterdam in implementing the IWLT system for city logistics, contributing to a more efficient and sustainable urban environment. Additionally, it outlines future research directions to enhance the robustness and applicability of the decision model.

6.1. Recommendations for Practice

The design and implementation of an efficient IWLT system involve several critical considerations. This discussion delves into the practical aspects and offers recommendations based on the results and insights obtained from the experiments, in addition to the insights provided in Section 5.6.

One of the key design choices is determining the number of satellites to use. The findings suggest that increasing the number of satellites up to a certain number generally enhances system performance in terms of distances travelled on the roads. Increasing the number of satellites past this number results in less efficient customer assignment and, therefore, more distance on the roads. The number of satellites for this turning point seems linearly related to the total demand of the customers. Utilising between 9 and 13 satellites is recommended to effectively supply the entire Horeca sector in Amsterdam. Two to four satellites are sufficient to supply Horeca locations in the Wallen neighbourhood.

However, there are practical considerations for using the satellites for Horeca supply, such as available space and potential conflicts with tourism activities. It is recommended that the municipality investigates the feasibility of dedicated logistics satellites, separate from tourism activities, to avoid congestion and ensure the uninterrupted flow of goods.

For areas on the outskirts of the city centre or specific neighbourhoods with low demand, implementing direct deliveries might be a viable strategy. This approach eliminates the need to for satellites in these areas, allowing satellites to be allocated closer to neighbourhoods with many Horeca. This could streamline operations and concentrate logistical efforts where they are most needed.

Storage capacity at satellites significantly impacts the efficiency of the logistics network. The results highlight that even modest storage capacities of $15\mathrm{m}^3$ at a few selected satellite locations can lead to substantial reductions in the number of required vessels and road vehicles. Although limited space and the visual impact of storage facilities in the city centre might be a concern, it is recommended to investigate the implementation of storage at a few strategic locations. Careful planning and aesthetic design can mitigate the visual impact while providing the logistical benefits of storage capacity at satellites.

The time span available for logistics operations plays a crucial role in system efficiency. Extended time spans allow for more flexible scheduling and can lead to fewer required vehicles and vessels. However, the practicalities of urban life must be considered, such as daytime tourism and nighttime noise regulations. An alternative approach could involve splitting the time span into two windows: one in the morning and another in the evening. This approach could accommodate logistical needs without overwhelming the city during peak hours, though it may result in less conventional working hours for employees.

Moreover, the simultaneous arrival of vessels, as highlighted in Subsection 5.3.2, presents a scheduling challenge for road vehicles, and will further do so for a split time span. To enhance the utilisation of road vehicles and overall system efficiency, it is suggested to stagger the loading times at the depot for vessels. This would prevent concurrent arrivals and allow for better synchronised schedules. However, this could lead to an increased number of required vessels, for which additional experiments should be conducted. With some adjustments, the developed decision model can investigate this scenario.

Another approach to provide a longer time span without causing nuisance in the city centre during peak hours could be to assign varying time windows to neighbourhoods. For instance, the busy inner city can be supplied in the morning from 6 A.M. to 10 A.M., after which the vehicles move to neighbourhoods with less busyness during daytime. The time windows should comply with the time it takes to perform a vessel trip to efficiently utilise the vessels.

The placement of depots is another critical factor. The experiments indicate that the chosen depot locations lead to long travel distances, thus increasing the number of required vessels. Relocating depots closer to the city centre could improve efficiency, but this comes with trade-offs. Closer depots might mean longer supply routes from the highway, which could offset some of the benefits. A thorough investigation is recommended to find a balanced solution.

For real-life applications, constraints to limit the number of customers assigned to a satellite were implemented, to evenly distribute the satellite utilisation and minimise inconveniences for city residents. However, this constraint has a negative effect on the distances travelled on the roads. It is important to investigate the nuisance transshipment activities cause for city residents, so the trade-off in distance and nuisance can be made.

For the investigated IWLT system, it is assumed that vessels are equipped with onboard cranes, which is practical given the limited space and potential visual impact of cranes in the city centre. However, if the number of satellites is less than the number of vessels, it may be more cost-effective to install cranes at the satellites instead. This approach could reduce the overall cost of the system while maintaining operational efficiency.

All the results discussed are based on simulated data. For a more accurate and reliable design, it is essential to collect and analyse real-life data on demand patterns and transshipment times. Pilot studies and real-world trials would provide valuable insights and help refine the model to better reflect actual conditions.

6.2. Recommendations for Further Research

The decomposition approach used for the decision model for IWLT systems shows promising results. However, there are certain limitations to the model, which are investigated in this section.

Due to the significant problem size for the city of Amsterdam, comparing outcomes across different scenarios poses challenges due to the variability in solution quality. Especially since the problem size changes for the scenarios. For instance, when investigating the number of satellites to utilise, the problem size increases when more satellites are used. As noted, conducting comprehensive tests becomes essential to accurately gauge the impact of the number of opened satellites on vehicle requirements. This necessitates extensive computational experiments to ensure dependable data for decision-making. The same effect is seen for the varying storage scenarios at satellites. Introducing storage increases the solution space, negatively impacting the quality of the solutions.

Preliminary tests were conducted for dedicating road vehicles to neighbourhoods, but without storage at the satellites this did not improve the results. For further research, dedicating road vehicles to a set of satellites with storage capacity can be investigated, since the reduction of the solution space might have positive effects for this scenario.

Another method to improve the solution quality is to iterate the scheduling process. By reducing the problem set in each iteration, the solutions can be further improved. Some tests to implement iterations were performed, but rounding errors of the solver resulted in infeasible initial solutions. This problem can be solved by various post-processing methods. For future work, it is recommended to implement feedback loops and iterations between the models to improve the solution quality.

Futhermore, given that the quality of results is influenced by the initial solution provided, improving these heuristics could lead to more consistent outcomes. Enhancing the quality of initial solutions has the potential to minimise variation in solution quality, thereby enhancing overall system performance.

Despite its strengths, the developed methodology is not without limitations. One limitation is its reliance on simplified assumptions and demand data, which may not fully capture the complexity of real-world logistics operations. Several assumptions underlie the developed methodology, shaping its scope and applicability. These include assumptions regarding demand patterns, vehicle capacities, and operational constraints. While these assumptions enable the formulation of tractable optimisation problems, they also introduce simplifications that may not hold in practice.

The developed decision model is specifically designed for the city of Amsterdam, but it can be adapted for other cities by implementing alternative infrastructure networks. Additionally, modifying vehicle characteristics is straightforward, allowing the model to be applied to various two-echelon location routing problems.

Overall, the decision model facilitates the comparison of various IWLT system scenarios. However, the complexity of large problem instances and the interdependencies between different models can impact solution quality. To enhance the results, an iterative approach could be adopted. This method would gradually reduce the problem size and minimise reliance on the solution quality of preceding models.

Bibliography

Amsterdam emission zones. (n.d.). Retrieved February 6, 2023, from https://urbanaccessregulations.eu/countries-mainmenu-147/netherlands-mainmenu-88/amsterdam

Anderluh, A., Nolz, P. C., Hemmelmayr, V. C., & Crainic, T. G. (2021). Multi-objective optimization of a two-echelon vehicle routing problem with vehicle synchronization and 'grey zone' customers arising in urban logistics. *European Journal of Operational Research*, *289*(3), 940–958. https://doi.org/10.1016/j.ejor.2019.07.049

Benjelloun, A., Crainic, T. G., & Bigras, Y. (2010). Towards a taxonomy of City Logistics projects. *Procedia - Social and Behavioral Sciences*, *2*(3), 6217–6228. https://doi.org/10.1016/j.sbspro.2010.04.032

Bijvoet, B. (2023, December 20). *Multimodal city logistics using waterways in amsterdam: Proof-of-concept of a two- echelon distribution network*. Delft University of Technology. http://resolver.tudelft.nl/uuid:a1123f0a-3995-4675-8ef4-02c0fe5a4686

Boccia, M., Crainic, T. G., Sforza, A., & Sterle, C. (2011). Location-Routing Models for Designing a Two-Echelon Freight Distribution System. *CIRRELT*.

Bull, A., for Latin America, U. N. E. C., & the Caribbean. (2004, January). *Traffic Congestion*. United Nations, Economic Commission for Latin America; the Caribbean.

Büyüközkan, G., & Ilıcak, Ö. (2022). Smart urban logistics: Literature review and future directions. *Socio-Economic Planning Sciences*, *81*, 101197. https://doi.org/10.1016/j.seps.2021.101197

Caris, A., Limbourg, S., Macharis, C., Van Lier, T., & Cools, M. (2014). Integration of inland waterway transport in the intermodal supply chain: a taxonomy of research challenges. *Journal of Transport Geography*, *41*, 126–136. https://doi.org/10.1016/j.jtrangeo.2014.08.022

CCNR. (2021, November). *Inland waterway transport embedded in urban logistics- ccnr - observation du marché*. Retrieved March 31, 2023, from https://inland-navigation-market.org/chapitre/2-inland-waterway-transport-embedded-in-urban-logistics/?lang=nl

City of Amsterdam. (n.d.). *Stricter rules for heavy vehicles*. Retrieved February 6, 2023, from https://www.amsterdam.nl/en/traffic-transport/stricter-rules-heavy-vehicles/

Contardo, C., Hemmelmayr, V., & Crainic, T. G. (2012). Lower and upper bounds for the two-echelon capacitated location-routing problem. *Computers and Operations Research*, *39*(12), 3185–3199. https://doi.org/10.1016/j.cor.2012.04.003

Cordaan, Gemeente Utrecht, Gemeente Amsterdam, Universiteit van Amsterdam, & Iv-Infra b.v. (n.d.). *Handreiking particuliere kademuren*. https://assets.amsterdam.nl/publish/pages/983539/handreiking_particuliere_kademuren_spread.pdf

Côté, J., Guastaroba, G., & Speranza, M. (2017). The value of integrating loading and routing. *European Journal of Operational Research*, *257*(1), 89–105. https://doi.org/10.1016/j.ejor.2016.06.072

Cuda, R., Guastaroba, G., & Speranza, M. (2015). A survey on two-echelon routing problems. *Computers & Operations Research*, *55*, 185–199. https://doi.org/10.1016/j.cor.2014.06.008

Dablanc, L., & Montenon, A. (2015). Impacts of environmental access restrictions on freight delivery activities. *Transportation Research Record*, *2478*(1), 12–18. https://doi.org/10.3141/2478-02

Daduna, J. R. (2019, September 30). *Developments in city logistics - the path between expectations and reality*. Springer Science+Business Media. https://doi.org/10.1007/978-3-030-31140-7_1

Daggers, T., & Heidenreich, J. (2013, November). *City Logistics with Electric Vehicles* (tech. rep.). International Bicycle Consultancy.

Dalfard, V. M., Kaveh, M., & Nosratian, N. E. (2012). Two meta-heuristic algorithms for two-echelon location-routing problem with vehicle fleet capacity and maximum route length constraints. *Neural Computing and Applications*, *23*(7-8), 2341–2349. https://doi.org/10.1007/s00521-012-1190-0

Dellaert, N., Dashty Saridarq, F., Van Woensel, T., & Crainic, T. G. (2019). Branch-and-Price–Based Algorithms for the Two-Echelon Vehicle Routing Problem with Time Windows. *Transportation Science*, *53*(2), 463–479. https://doi.org/10.1287/trsc.2018.0844

Dellaert, N., Van Woensel, T., Crainic, T. G., & Dashty Saridarq, F. (2021). A multi-commodity two-Echelon capacitated vehicle routing problem with time windows: Model formulations and solution approach. *Computers and Operations Research*, *127*, 105154. https://doi.org/10.1016/j.cor.2020.105154

Demir, E., Huang, Y., Scholts, S. S., & Van Woensel, T. (2015). A selected review on the negative externalities of the freight transportation: Modeling and pricing. *Transportation Research Part E-logistics and Transportation Review*, *77*, 95–114. https://doi.org/10.1016/j.tre.2015.02.020

(DHPC), D. H. P. C. C. (2024). DelftBlue Supercomputer (Phase 2) [https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2].

Divieso, E., Junior, O. F. L., & De Oliveira, H. C. (2021). The use of waterways for urban logistics: The case of brazil. *Theoretical and Empirical Researches in Urban Management*, *16*(1), 62–85.

Diziain, D., Taniguchi, E., & Dablanc, L. (2014). Urban logistics by rail and waterways in france and japan. *Procedia - Social and Behavioral Sciences*, *125*, 159–170. https://doi.org/10.1016/j.sbspro.2014.01.1464

Drexl, M. (2012). Synchronization in Vehicle Routing—A Survey of VRPs with Multiple Synchronization Constraints. *Transportation Science*, *46*(3), 297–316. https://doi.org/10.1287/trsc.1110.0400

Drexl, M., & Schneider, M. (2015). A survey of variants and extensions of the location-routing problem. *European Journal of Operational Research*, *241*(2), 283–308. https://doi.org/10.1016/j.ejor.2014.08.030

Economic, E., & Committee, S. (2014, January). *NAIADES II: Proposal for a Regulation of the European Parliament and of the Council amending Council Regulation on a Community-fleet capacity policy to promote inland waterway transport* (tech. rep. No. TEN/532). https://transport.ec.europa.eu/transport-modes/inland-waterways/promotion-inland-waterway-transport/naiades-ii_en

Escobar-Vargas, D., Crainic, T. G., & Contardo, C. (2021). Synchronization in Two-Echelon Distribution Systems: Models, Algorithms, and Sensitivity Analyses. *CIRRELT*, *50*.

European Union. (2021, September). *Climate-neutral and smart cities*. Retrieved March 3, 2023, from https://research-and-innovation.ec.europa.eu/funding/funding-opportunities/funding-programmes-and-open-calls/horizon-europe/eu-missions-horizon-europe/climate-neutral-and-smart-cities_en

Farahani, R. Z., SteadieSeifi, M., & Asgari, N. (2010). Multiple criteria facility location problems: A survey. *Applied Mathematical Modelling*, *34*(7), 1689–1709. https://doi.org/10.1016/j.apm.2009.10.005

Gemeente Amsterdam. (n.d.). *Ontheffing zwaar verkeer – checken en aanvragen*. Retrieved November 20, 2022, from https://www.amsterdam.nl/verkeer-vervoer/zwaar-verkeer/ontheffing-checken-aanvragen/

Gemeente Amsterdam. (2020, April). *Herstellen en verbinden: Bouwen aan het fundament van de stad* (Programmaplan 2020). https://experience.arcgis.com/experience/5f24774720454550ae8e2b93e909f564/page/Documenten/

Gemeente Amsterdam. (2021). *Uitkomsten proef ophalen afval per boot*. Retrieved November 20, 2022, from https://www.amsterdam.nl/afval-hergebruik/proef-afval-ophalen-per-boot/uitkomsten-proef-ophalen-afval-per-boot/

Goededinge.be. (2020). *Bioboot: Verse bio groenten recht van het veld*. Retrieved March 31, 2023, from https://goedinge.be/bioboot.html

Grangier, P., Gendreau, M., Lehuédé, F., & Rousseau, L.-M. (2016). An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal of Operational Research*, *254*(1), 80–91. https://doi.org/10.1016/j.ejor.2016.03.040

Groothedde, B. F. E. M., Ruijgrok, C., & Tavasszy, L. (2005). Towards collaborative, intermodal hub networks. *Transportation Research Part E-logistics and Transportation Review*, *41*(6), 567–583. https://doi.org/10.1016/j.tre.2005.06.005

HAROPA - Ports de Paris. (2012). *Logistique urbaine: Franprix, un exemple réussi*. Retrieved March 31, 2023, from https://scot.metropolegrandparis.fr/wp-content/uploads/2018/12/MGP_Labo_SCOT_fiche-HAROPA_logistiqueurbaine_181206.pdf

Hemmelmayr, V. C., Cordeau, J.-F., & Crainic, T. G. (2012). An adaptive large neighborhood search heuristic for Two-Echelon Vehicle Routing Problems arising in city logistics. *Computers and Operations Research*, *39*(12), 3215–3228. https://doi.org/10.1016/j.cor.2012.04.007

Huang, Y., Savelsbergh, M., & Zhao, L. (2018). Designing logistics systems for home delivery in densely populated urban areas. *Transportation Research Part B: Methodological*, *115*, 95–125. https://doi.org/10.1016/j.trb.2018.07.006

Huijgen, M., Lange, K., Warmerdam, J., Oskam, I., & Ploos van Amstel, W. (2022). *Eindrapport kiem-ce project wow - waste-on-water: Collectieve afvalinzameling over water*. HvA Urban Technology.

Jandl, O. M. (2016). *Implementing inland waterway transportation in urban logistics*. Chalmers University of Technology.

Janjevic, M., & Ndiaye, A. B. (2014). Inland waterways transport for city logistics: A review of experiences and the role of local public authorities. *WIT Transactions on the Built Environment*, *138*, 279–292. https://doi.org/10.2495/UT140241

Jia, S., Deng, L., Zhao, Q., & Chen, Y. (2022). An adaptive large neighborhood search heuristic for multi-commodity two-echelon vehicle routing problem with satellite synchronization. *Journal of Industrial and Management Optimization*, *19*(2), 1187. https://doi.org/10.3934/jimo.2021225

Karademir, C., Alves Beirigo, B., Negenborn, R., & Atasoy, B. Two-echelon multi-trip vehicle routing problem with synchronization for an integrated water- and land-based transportation system [hEART 2022: 10th Symposium of the European Association for Research in Transportation; Conference date: June 2022]. In: hEART 2022: 10th Symposium of the European Association for Research in Transportation; Conference date: June 2022. 2022.

Lehr, T. (2017). Smart Cities: Vision on-the-Ground. *Smart Cities*, 3–15. https://doi.org/10.1007/978-3-319-59381-4_1

Li, H.-Q., Liu, Y., Kaihang, C., & Lin, Q. (2020). The two-echelon city logistics system with on-street satellites. *Computers and Industrial Engineering*, *139*, 105577. https://doi.org/10.1016/j.cie.2018.12.024

Maes, J., Sys, C., & Vanelslander, T. (2012, June). *Vervoer te water: Linken met stedelijke distributie?* (No. D/2012/11.528/8).

Marques, G., Sadykov, R., Deschamps, J.-C., & Dupas, R. (2020b). An improved branch-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem. *Computers and Operations Research*, *114*, 104833. https://doi.org/10.1016/j.cor.2019.104833

Marquès, G., Sadykov, R., Dupas, R., & Deschamps, J.-C. (2020a). A Branch-Cut-and-Price Approach for the Single-Trip and Multi-Trip Two-Echelon Vehicle Routing Problem with Time Windows. *Transportation Science*, *56*(6), 1598–1617. https://doi.org/10.1287/trsc.2022.1136

Mhamedi, T., Andersson, H., Cherkesly, M., & Desaulniers, G. (2022). A Branch-Price-and-Cut Algorithm for the Two-Echelon Vehicle Routing Problem with Time Windows. *Transportation Science*, *56*(1), 245–264. https://doi.org/10.1287/trsc.2021.1092

Mirhedayatian, S. M., Crainic, T. G., Guajardo, M., & Wallace, S. W. (2019). A two-echelon location-routing problem with synchronisation. *Journal of the Operational Research Society*, *72*(1), 145–160. https://doi.org/10.1080/01605682.2019.1650625

Mommens, K., & Macharis, C. (2012, July). *Pallets on the inland waterways* (No. D/2012/11.528/13). Retrieved March 31, 2023, from https://medialibrary.uantwerpen.be/oldcontent/container33836/files/Beleidsondersteunende_papers/BP2012_5_pallets.pdf

Moshref-Javadi, M., Lee, S., & Winkenbach, M. (2020). Design and evaluation of a multi-trip delivery model with truck and drones. *Transportation Research Part E: Logistics and Transportation Review*, *136*, 101887. https://doi.org/10.1016/j.tre.2020.101887

Nikbakhsh, E., & Zegordi, S. (2010). A HEURISTIC ALGORITHM AND A LOWER BOUND FOR THE TWO-ECHELON LOCATION-ROUTING PROBLEM WITH SOFT TIME WINDOW CONSTRAINTS. *Scientia Iranica*, *17*(1), 36–47. http://scientiairanica.sharif.edu/article_3323_914c872ce83884ecb171cbce20a77 pdf

Observatoire Régional Transports & Logistique - Grand Est. (2020, July 21). *Nouveau service de logistique urbaine fluviale à strasbourg*. Retrieved April 1, 2023, from https://www.ortl-grandest.fr/nouveau-service-logistique-urbaine-fluviale-strasbourg/

Organization, W. H. (2022, December). Ambient (outdoor) air pollution. https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health

Parr, T., & Register of Initiatives in Pedal Powered Logistics. (2017, October 11). *Boat-bike: Dhl's multimodal amsterdam logistics chain*. Retrieved December 3, 2022, from https://www.rippl.bike/en/rippl-36-boat-bike-dhls-multimodal-amsterdam-logistics-chain/

Prodhon, C., & Prins, C. (2014). A survey of recent research on location-routing problems. *European Journal of Operational Research*, *238*(1), 1–17. https://doi.org/10.1016/j.ejor.2014.01.005

Ritchie, H. (2018, June). Urbanization. https://ourworldindata.org/urbanization

Schneider, M., & Drexl, M. (2017). A survey of the standard location-routing problem. *Annals of Operations Research*, *259*(1-2), 389–414. https://doi.org/10.1007/s10479-017-2509-0

Sluijk, N., Florio, A. M., Kinable, J., Dellaert, N., & Van Woensel, T. (2023). Two-echelon vehicle routing problems: A literature review. *European Journal of Operational Research*, *304*(3), 865–886. https://doi.org/10.1016/j.ejor.2022.02.022

Technische Universität Berlin. (2022, February 22). *Swarm on the water - innovative freight transport for the capital*. Retrieved April 1, 2023, from https://www.tu.berlin/en/about/profile/press-releases-news/2022/februar/a-swarm-autonomous-transport-boats/

Toth, P. (2002). *The Vehicle Routing Problem*. SIAM.

Towards sustainable urban distribution using city canals: The case of amsterdam. (2017). In R. van Duin, M. van de Kamp, & R. Kortmann (Eds.).

TRiLOGy. (n.d.). *About*. Retrieved April 14, 2023, from https://trilogy-tud.github.io/about/

Van Duin, J., & Quak, H. (2007). City logistics: a chaos between research and policy making? A review. https://doi.org/10.2495/ut070141

Vidal, T., Laporte, G., & Matl, P. (2020). A concise guide to existing and emerging vehicle routing problem variants. *European Journal of Operational Research*, *286*(2), 401–416. https://doi.org/10.1016/j.ejor.2019.10.010

Wang, K., Pesch, E., Kress, D., Fridman, I., & Boysen, N. (2022). The Piggyback Transportation Problem: Transporting drones launched from a flying warehouse. *European Journal of Operational Research*, *296*(2), 504–519. https://doi.org/10.1016/j.ejor.2021.03.064

Winkenbach, M., Kleindorfer, P. R., & Spinler, S. (2016). Enabling Urban Logistics Services at La Poste through Multi-Echelon Location-Routing. *Transportation Science*, *50*(2), 520–540. https://doi.org/10.1287/trsc.2015.0624

Wojewódzka-Król, K., & Rolbiecki, R. (2019). The Role of Inland Waterway Transport in City Logistics. *Transport Economics and Logistics*, *84*, 103–114. https://doi.org/10.26881/etil.2019.84.09

A

Appendix: Scientific Paper

Scientific paper starts on next page.

Multi-modal last-mile delivery: Designing integrated water- and land-based transportation systems for Horeca supply in Amsterdam

L.C. Brockhoff, C. Karademir, B. Atasoy and M.W. Ludema

This research presents a comprehensive study on the development of Integrated Water and Land-based Transportation (IWLT) Systems for city logistics. The research addresses the growing challenges of urban traffic by proposing a decision model for multi-modal transportation systems that leverages waterways alongside traditional road networks. The problem is defined as a two-echelon multi-trip location routing problem with satellite synchronisation (2E-MTLRP-SS), incorporating capacitated vehicles, multiple depots and time constraints. A decomposition-based decision model is introduced, breaking down the problem into manageable sub-problems interconnected through synchronisation in time, space, and load. The decision model uses metaheuristics to be able to handle large-scale, realistic problems and provide feasible solutions for real-life applications. The model's effectiveness is demonstrated through a case study in Amsterdam, showing the potential of IWLT systems to reduce congestion-related issues and improve the livability of cities. Different scenarios for the IWLT system are investigated, to assist Amsterdam's system developers in making design choices for implementation. The proposed decision model is widely applicable to multi-modal transportation systems all over the world.

I. Introduction

More and more people are living in urban areas, the percentage of the population living in cities keeps growing (Ritchie, 2018). All these people need food and beverages, their waste has to be collected, and many have to commute. While at the same time, e-commerce is rapidly expanding (Huang et al., 2018). This together results in a growing number of vehicles in urban areas, which has, among other things, a negative impact on the quality of life in cities (Daggers & Heidenreich, 2013).

This study is motivated by the various negative consequences of increased urban traffic on the quality of life inside cities [1] [2] [3] and investigates methods to reduce the effect of urban freight logistics. Change is needed to improve city logistics, since the quality of life in cities keeps worsening.

One way to reduce the increase in urban traffic is to shift modalities or integrate different modalities for multi-modal transportation systems. A multi-modal transportation system coordinates the use of two or more modes of transport. From depots, first-mode vehicles are used for transport to satellites. Satellites are transshipment locations, where the cargo is transshipped between the transportation modes. From the satellites, second-mode vehicles perform deliveries to customers.
Interest in the use of waterways in city logistics is growing, since the capacity of inland waterways is currently underused, and transport using inland waterways has the lowest external costs in terms of emissions, noise, accidents and bottlenecks [4] compared to other modes of transport.

However, despite the advantages, waterways are not often implemented in city logistics yet due to limited research on this issue [4]. To implement an IWLT system in city logistics, many design choices at the strategic and tactical levels need to be made. Therefore, the need for a decision model that covers large-scale real-life applications arises. Existing decision models often do not account for practical applications, especially for large-scale instances, which require high-level synchronisation.

An IWLT system consists of three main problems, the routes of the first-mode vehicles, the locations of the satellites and the routes of the second-mode vehicles. These can be modelled as a two-echelon location routing problem (2E-LRP), or a combination of a facility location problem (FLP) and a two-echelon vehicle routing problem (2E-VRP). Many variants exist for these problems, adding extra attributes for a more realistic representation of real-life applications considering practical limitations. The variants important for the IWLT system are multiple trips, time-windows, satellite capacity and satellite synchronisation. However, the basic versions of these problems do not include the required synchronisation for real-life implementation. Variants including multiple trips and time-windows are extensively studied in literature. However, variants including satellite synchronisation and limited storage capacities are relatively new.

The problem is defined as a two-echelon multi-trip location routing problem with satellite synchronisation (2E-MTLRP-SS), incorporating capacitated vehicles, multiple depots and a global time window, with a possibility of satellite storage. Existing decision models often do not account for practical applications, which require high-level synchronisation, especially for large-scale instances. A decomposition-based decision model is introduced, breaking down the problem into manageable sub-problems interconnected through synchronisation in time, space, and load. The decision model is capable of handling large-scale, realistic problems and providing feasible solutions for real-life applications. The model's effectiveness is demonstrated through a case study in Amsterdam, showing the potential of IWLT systems to reduce urban traffic and its negative aftereffects. Different scenarios for the IWLT system are investigated, to assist the municipality of Amsterdam in making design choices for implementation and creating policies for regulations. The proposed decision model is widely applicable to multi-modal transportation systems all over the world.

II. Literature

Two-echelon vehicle routing problems are extensively researched and over the years many variants have been studied [5]. A large body of work exists on many variants and therefore, this paper will focus only on the two-echelon vehicle routing problems with satellite synchronisation and/or satellite capacity (2E-VRP-SS or 2E-VRP-SC), possibly with different side constraints.

Marquès et al. [6] suggest a mixed integer programming formulation for the problem with a branch-cut-and-price algorithm to solve it. They are the first to propose an exact algorithm for the two-echelon vehicle routing problem with multi-trip, time-windows and satellite synchronisation (2E-MTVRPTW-SS) and include the possibility of multiple depots.

Some relatively new research is being conducted by Karademir et al. [7]. The focus is on an IWLT system in the city centre, this is why they consider an important constraint, namely that only one transfer can take place at a time. Multiple transfer operations that happen simultaneously are not feasible in busy areas with limited space. They are the first to take this into account. The problem solved is a two-echelon vehicle routing problem with time-windows, multiple trips and satellite synchronisation and is formulated as a mixed-integer linear programming problem. They solve instances with one depot, four satellites and 10 customers.

Li et al. [8] use a variable neighbourhood search heuristic to solve the two-echelon logistics system with on-street satellites that uses time windows and satellite transshipment constraints, which in the termination of this paper is equal to the 2E-MTVRPTW-SS. They can solve instances with one depot, up to 30 satellites and 900 customers in under two hours. Next to this, they evaluate the economic difference between the use of electric or diesel vehicles and different vehicle capacities.

Jia et al. [9] provide both a heuristic and exact solution method for the two-echelon vehicle routing problem with multiple depots, time-windows, satellite capacity and satellite synchronisation. A mixed-integer programming model and an adaptive large neighbourhood search are developed. They are able to solve problems with 2 depots, 10 satellites and 260 customers.

Anderluh et al. [10] use a large neighbourhood search embedded in a heuristic rectangle/cuboid splitting to solve the two-echelon vehicle routing problem with multi-trip and satellite synchronisation (2E-MTVRP-SS). They neglect time-windows and the instances they solve are smaller than those of Li et al. [8], but what makes their research interesting is its option to use multiple objectives, the standard economic objective, but also negative external effects, like emissions and disturbances, caused by congestion and noise. This possibility makes their solution method especially interesting when design choices still have to be made.

All research discussed above assumes known satellite locations, however, when investigating the implementation of an IWLT system, suitable satellite locations are not always predetermined. Two-echelon location routing problems do consider satellite selection. The available research on the two-echelon location routing problem is significantly less than that on the two-echelon vehicle routing problem, especially concerning variations including satellite synchronisation.

Relatively new research is conducted by Bijvoet [11], who solve a two-echelon multi-trip vehicle routing problem with synchronisation (2E-MTVRP-SS) with decomposition-based heuristics. Special in the work is their consideration of multiple trips for both echelons and usage of a heterogeneous fleet for the second echelon. They solve large-scale instances with one depot, 45 satellites and 758 customers. Before solving the 2E-MTVRP-SS suitable satellites are determined from a set of potential locations.

Contardo et al. [12] observe the 2E-LRP can be decomposed in two LRPs, connected by capacitated satellites. They use a branch-and-cut algorithm to solve the problem with multiple depots. An initial solution for the second echelon is constructed. After this, a solution for the first echelon is constructed by randomly selecting one depot and serving all satellites from it. A destroy-repair iteration is performed on the second-echelon and then on the first-echelon problem. Local Search is only performed on the second-echelon problem.

Mirhedayatian et al. [13] claim to be the first to study a two-echelon location routing problem with time windows and synchronisation (2E-LRPTW-SS). They propose a decomposition-based heuristic solution approach, which is done in three stages. First, a configuration of satellite locations is chosen, then, customers are assigned for this configuration and lastly, the routes of the echelons are established. Feedback loops between the stages ensure working towards the best solution. Different sets of instances are tested and solved for at most 40 nodes. The average computation time for the instances was $2993s$.

Escobar-Vargas et al. [14] presents two mixed-integer programming formulations and an exact solution framework by a dynamic time discretisation scheme for a two-echelon location routing problem with time windows and satellite synchronisation. They formulate the problem as a Two-Echelon Multi-Attribute Location-Routing Problem with fleet synchronisation at intermediate facilities (2E-MALRPS), which results in a 2E-LRPTW-SS by the definitions used in this paper. The two mixed-integer programming formulations used are a compact formulation and a time-space formulation. Because of the temporal dimension of the time-space formulation, the model is more realistic but also less scalable. They propose a dynamic discretisation discovery (DDD) framework to improve the scalability. The DDD solution framework is able to solve instances of 6 depots, 4 satellites and 10 customers optimally in $4936s$ and find feasible solutions for all instances up to 6 depots, 4 satellites and 30 customers in $36000s$.

III. Methodology

A. Problem definition

The problem is to supply customers using multi-modal transportation. Cargo originates from a depot of set DC_w, with unlimited storage and loading capacity, allowing simultaneous loading of multiple vehicles. Transshipment at the depot takes t^{DC} minutes per vessel.

The cargo is then transported by vessels of set F from a depot to satellites. Vessels have a capacity of $q^{\mathrm{W}}\,[m^3]$ and a speed $v^{\mathrm{W}}\,[m/s]$. They can perform multiple trips of set W and visit multiple satellites in one trip, if those trips and satellites are assigned to the same depot.

The satellite locations have to be selected from a set S of potential location, of which N^{S} can be opened. Satellites in the standard configuration have no storage capacity, $q^{\mathrm{S}} = 0$, necessitating direct transshipment from vessels to road vehicles, a process taking t^{S} minutes. Vessels might have to wait at a satellite until the cargo is picked up and transshipment activities can only be performed on one vessel and one road vehicle at a satellite simultaneously. However, the satellite capacity can be adjusted for specific cases by changing parameter q^{S}.

Road vehicles of set R transport the cargo from satellites to customers in set C, with a demand of $q_c\,[m^3]$ per customer and the demand of all customers has to be satisfied. Each road vehicle can perform multiple trips of set V and can visit multiple customers in a trip, as long as their load does not exceed their capacity of $q^{\mathrm{V}}\,[m^3]$. Road vehicles have a speed of $v^{\mathrm{V}}\,[m/s]$, and transshipment at a customer takes a fixed t^{C} minutes. Road vehicles start their first trip and end their last trip at a road vehicle depot, DC_v.

Routes are established for both modalities: waterways for first echelon vehicles and roads for second echelon vehicles. Distances between depot, satellites, and customers are given by Δ_{ij}.

All transshipment activities must occur within a maximum time span, $t^{\max}$ minutes. Vessels can start their trip before the beginning of the time span and exceed this time window when travelling back to the depot. Road vehicles can still perform deliveries of the last trip.

This problem is defined as a two-echelon multi-trip location routing problem with satellite synchronisation (2E-MTLRP-SS), incorporating capacitated vehicles, multiple depots and a global time window, with a possibility of satellite storage. Both echelons have a homogeneous fleet. The primary objective is to minimise road burden while ensuring real-life feasibility in terms of costs and time. This involves minimising the number of vehicles required and adhering to all time constraints. Additionally, minimising the distance travelled on the waterways is a sub-objective to ensure that reducing road traffic does not result in excessive congestion on the waterways.

Key decision variables include satellite locations, the number of satellites to open, and vehicle numbers for both modalities. Vehicle characteristics are governed by regulations and system requirements and are represented as parameters. The routes of the vehicles are an important factor for the objectives, which are evaluated by kilometres on the roads and waterways.

B. Modelling approach

The strategy used in this research, is to decompose the problem in an FLP and two separate VRPs for water and street level, while incorporating integration and synchronisation, and a scheduling problem. For the two VRPs, using only exact methods reduces the problem variations and instances that can be tackled. Using only heuristic methods can result in sub-optimal results. Therefore, to achieve high-quality results, both heuristic and exact methods are developed and combined. The scheduling problem is added to enable multiple trips and reduce the required number of vehicles. Multiple trips could also be implemented in the vehicle routing problems, but this makes the problem size significantly larger.

The problem is decomposed into four problems: the facility location problem, the second-echelon vehicle routing problem, the first-echelon vehicle routing problem and the scheduling problem. The VRPs and scheduling problem each consist of multiple sub-problems. Below, an overview of the (sub-)problems is given:

- **Facility location problem :**
  MILP model to determine the satellite locations to open and assign customers to those satellites
- **Second-echelon vehicle routing problem :**
  - VRP road initial:
    Heuristic method to establish initial routes for the road vehicles
  - VRP road improvement:
    MILP model to improve the initial road vehicle routes
  - Split trips road vehicles:
    Simple heuristics to split the road vehicle routes into separate trips
- **First-echelon vehicle routing problem :**
  - VRP water initial:
    Heuristic method to establish initial routes for the water vehicles
  - VRP water improvement + synchronisation:
    MILP model to improve the initial water vehicle routes and implement synchronisation between the two echelons
- **Scheduling problem :**
  - Scheduling road vehicles:
    MILP model to schedule the road vehicle

trips and determine the required number of road vehicles while respecting synchronisation constraints to water vehicles

- Scheduling water vehicles:
  MILP model to schedule the water vehicle trips and determine the required number of water vehicles while respecting synchronisation constraints to road vehicles
- Scheduling total system:
  MILP model to improve the schedules for both echelons while respecting synchronisation constraints

1. Facility Location Problem

The basic version of the facility location problem is given below. The customer assignment is decision variable U_{ij}, which is $U_{ij} = 1$ if customer j is assigned to satellite i. $O_i = 1$ if satellite i is open. The objective is to minimise the sum of the distances between satellites and their assigned customers, as shown in Equation 1.

$$\min \sum_{i \in S} \sum_{j \in C} U_{ij} * \Delta_{ij} \tag{1}$$

$$\sum_{i \in S} U_{ij} = 1 \qquad \forall j \in C \tag{2a}$$

$$U_{ij} \leq O_i \qquad \forall i \in S, j \in C \tag{2b}$$

$$\sum_{i \in S} O_i \leq N^{\mathrm{S}} \tag{2c}$$

Constraint Equation 2a ensures each customer is assigned to one satellite, constraint Eq. (2b) makes sure the satellite can only be used when it is opened. While Eq. (2c) limits the number of opened satellites to N^S.

Other variants of the FLP are investigated, adding constraints to limit the number of customers assigned to a satellite, since assigning too many customers to one satellite is not desirable. Because of the transshipment time at satellites, it might not be possible to serve all these locations within a reasonable time.

Two options are considered to limit the number of customers assigned to a satellite. The first method is to allow a maximum number of customers to be assigned to a satellite, implemented by Equation 3a. The second option is to limit the throughput allowed at a satellite, given by constraint Equation 3b. The throughput is the units of load transferred through one satellite.

$$\sum_{j \in C} U_{ij} \leq \frac{|C|}{N^S} \cdot b \qquad \forall i \in S \tag{3a}$$

$$\sum_{j \in C} q_j U_{ij} \leq \frac{\sum_{j \in C} q_j}{N^S} \cdot a \qquad \forall i \in S \tag{3b}$$

2. Second-Echelon VRP

First, an initial solution is created for the second-echelon vehicle routing problem using heuristics inspired by Greedy and Nearest Neighbour heuristics. For each satellite one vehicle supplies all customers assigned to that satellite, customers are greedily added to a vehicle trip, until the vehicle capacity is reached, upon which the vehicle returns to the satellite and starts a new trip.

Next, the solution found by the heuristics is used as an initial solution for the MILP model for the second-echelon vehicle routing problem. This model is a basic version of the VRP with capacity constraints, the specifics can be found in Brockhoff [15]. From this model, the routes of the second-echelon vehicle trips are obtained, with their duration and demand at their satellite. Post-processing calculations provide the time it takes to perform trip k and start trip l: p_{kl}, which is important for the road vehicle scheduling problem.

3. First-echelon VRP and Synchronisation

The previous models were straightforward, but in the first-echelon vehicle routing problem, integration of the two echelons is applied, leading to a more complex model.

Again, first, an initial solution for the first-echelon vehicle routes is created, using the same method as for the second-echelon vehicle routing problem. Then, a more elaborate MILP model is developed, with capacity, time and synchronisation constraints. Basic constraints are implemented in the same manner as the second-echelon vehicle routing problems. More complex constraints are added, relevant for synchronising the two echelons involve obtaining the sequence in which vehicles arrive (B_{ikl}, Eq. (4)) and depart (G_{ikl}, Eq. (5)) satellites. For vehicle trips k and l in the combined set of trips for both echelons WV, $B_{ikl} = 1$ if vehicle k arrives at satellite i after vehicle l, $G_{ikl} = 1$ if vehicle trip k leaves satellite i after vehicle trip l. The constraints for these sequences are only implemented if both vehicle trips k and l visit satellite i, indicated by $Y_{ikl} = 1$. Eq. (7) ensures the synchronisation in terms of load at satellites. The entire mathematical model is specified in Brockhoff [15], below, the most important synchronisation constraints are given.

$$Y_{ikl} = 1 \Rightarrow \quad A_{ik} - K * B_{ikl} - A_{il} \leq 0 \quad \forall k, l \in WV, i \in \bar{S} \tag{4a}$$

$$Y_{ikl} = 1 \Rightarrow \quad B_{ikl} + B_{ilk} = 1 \quad \forall k, l \in WV, i \in \bar{S} \tag{4b}$$

$$B_{ikl} + B_{ilk} \leq 1 \quad \forall k, l \in WV, i \in \bar{S} \tag{4c}$$

$$Z^{\mathrm{WV}}_{ik} = 0 \Rightarrow \quad B_{ikl} = B_{ilk} = 0 \quad \forall k, l \in WV, i \in \bar{S} \tag{4d}$$

$$Y_{ikl} = 1 \Rightarrow \quad D_{ik} - K * G_{ikl} - D_{il} \leq 0 \quad \forall k, l \in WV, i \in \bar{S} \tag{5a}$$

$$Y_{ikl} = 1 \Rightarrow \quad G_{ikl} + G_{ilk} = 1 \quad \forall k, l \in WV, i \in \bar{S} \tag{5b}$$

$$B_{ikl} = 1 \Rightarrow \quad G_{ikl} = 1 \quad \forall k, l \in V0, i \in \bar{S} \tag{5c}$$

$$B_{ikl} = 1 \Rightarrow \quad G_{ikl} = 1 \quad \forall k, l \in W0, i \in \bar{S} \tag{5d}$$

$$Z^{\mathrm{WV}}_{ik} = 0 \Rightarrow \quad \sum_{l \in WV} G_{ikl} + \sum_{l \in WV} G_{ilk} = 0 \quad \forall i \in \bar{S}, k \in WV \tag{5e}$$

$$B_{ikl} = 1 \Rightarrow \quad LS_{ik} - LS_{il} - Q^{\mathrm{W}}_{ik} \geq 0 \quad \forall k, l \in WV0, i \in \bar{S} \tag{6a}$$

$$LS_{ik} \leq \sum_{w \in W} Q^{\mathrm{W}}_{iw} \quad \forall k \in WV, i \in \bar{S} \tag{6b}$$

$$Z^{\mathrm{WV}}_{ik} = 0 \Rightarrow \quad LS_{ik} = 0 \quad \forall k \in WV, i \in \bar{S} \tag{6c}$$

$$Z^{\mathrm{WV}}_{ik} = 1 \Rightarrow \quad 0 \leq \sum_{l \in V} L^{V}_{il} * G_{ikl} + L^{V}_{ik} - LS_{ik} \leq q^{\mathrm{S}}_{i} \quad \forall i \in \bar{S}, k \in W \tag{7a}$$

4. Scheduling Problem

The last problem is the **scheduling problem of vehicle trips**, which schedules the found trips for the road and water vehicles. This problem is split into three sub-problems: MIP optimisations for first the road vehicle schedule; second, the water vehicle schedule; and lastly, the integrated schedule for all vehicles. The scheduling problem is split up to reduce the problem instances for MIP optimisation. The outputs of the separate scheduling problems are used as initial solutions for the next scheduling problem. Throughout the sub-problems the complexity reduces and the solution improves.

Each scheduling model is an addition to the water vehicle routing problem, the constraints given for the first-echelon vehicle routing problem are still valid, with extra constraints added for each scheduling problem.

Scheduling the trips is necessary to determine the number of vehicles required for performing all deliveries within a specified time span. With unlimited vehicles, each vehicle could perform one trip and the time span would be minimal. However, vehicles are expensive, so this is not desirable. Also, if unlimited time is available, all deliveries could be made by just one vehicle per echelon. Again, this is not desirable. Each day, new orders are made, and with such a system, the orders will pile up. Therefore, a balance has to be found between the time span and the number of vehicles.

Road Vehicle Scheduling
First, a basic initial schedule for the road vehicle trips is determined, by adding trips to a road vehicle until the time span is reached. This initial schedule is used to reduce the size of the problem for the MIP solver. The initial schedule reduces the number of road vehicles for the MIP solver by approximately 25%.

The constraints added to the first-echelon vehicle routing problem include basic vehicle trip routing constraints for the road vehicles, such as round trips, leaving the depot only once and performing each trip once. An important new decision variable is T^{V}_{klr}, which indicates whether road vehicle r performs trip k and next performs trip l. Equation 8 ensures a road vehicle can only perform the trips sequentially if the start time of trip l is later than or equal to the end time of trip k.

$$T^{\mathrm{V}}_{klr} = 1 \Rightarrow \qquad A^{\mathrm{R}}_{lr} \geq A^{\mathrm{R}}_{kr} + p_{kl} \qquad \forall r \in R, k \in V0, l \in V \quad (8)$$

Water Vehicle Scheduling
The water vehicle scheduling model exists of similar constraints as the road vehicle scheduling model, but with the decision variables only for the water vehicles. The road vehicle schedule is integrated as a fixed solution, only the arrival times can be adjusted, if that improves the water vehicle schedule and the schedule remains feasible for the road vehicle. The water vehicle trips found in the first-echelon vehicle routing problem are scheduled to water vehicles. The goal is to minimise the number of water vehicles required to perform all trips, while respecting the synchronisation constraints. Water vehicles can only perform trips that start from the same depot.

Integrated Scheduling
Now the road and water vehicle sets are reduced by the previous scheduling models, the models are integrated to improve the schedules for both echelons. The results of the previous models are implemented as an initial solution, to guide the model in the right direction. The decision variables are active for both echelons, meaning the model has the freedom to adjust both schedules. The objective of this model is to minimise the required number of vehicles for the two echelons and to minimise the distance travelled on the roads.

C. Case Amsterdam

This research is conducted in collaboration with the municipality of Amsterdam. The specific IWLT system for the city centre of Amsterdam is solved with the model to provide the municipality with insights for implementation, while simultaneously verifying the modelling approach developed in this research. Data about the demand is collected, parameter values determined and possible satellite locations, customer (Horeca) locations and the network are specified.

Experiments are performed on problem instances for Horeca supply in the city of Amsterdam. The canal and road network are obtained from previous research on IWLT systems done between Delft University of Technology and the municipality of Amsterdam. These networks are connected by satellites, of which the nodes are included in both networks. More information about the constructions of the networks can be found in the research by Bijvoet [11].

The customer (Horeca) locations can be obtained through public data from the municipality of Amsterdam. The city centre counts 1635 Horeca locations. Furthermore, the potential satellite locations are determined by selecting existing transfer sites in the city centre, 56 in total. The locations used in this research are equal to those in Bijvoet [11].

The research of Bijvoet [11] provides 10 demand sets for the Horeca locations, each set representing one simulated day. The demand is based on the probability of 45% that a location has a demand per day. The demand can be one, two or three units. In the work of Bijvoet [11], a unit is specified as one rolling container, which is 0.8m in length, 0.64m in width and 1.6m in height, resulting in 0.8192m^3. The 10 demand sets are all quite similar, with the number of Horeca locations with demand between 696 and 758 per day, and the total demand between 1416 and 1520 units. It is important to investigate the efficiency of the system when demand changes and, with that, the scalability of the system. Extra demand sets with more extreme values are created to test the adaptability of the system. Table 1 shows the demand probability distribution, the total demand and number of customers with demand for each set.

The demand units were determined as 0.8192m^3, but in the rest of this research one demand unit is equal to one cubic meter. This makes calculations more clear and accounts for sub-optimal use of vehicle capacity.

Table 1. Demand probability distribution per demand set with the total demand and number of customers with demand for one day, demands in m^3

Set						Demand	Customers
1	Demand	0	1	2	3	988	506
	Probability	70%	10%	10%	10%		
2	Demand	0	1	2	3	1498	750
	Probability	55%	15%	15%	15%		
3	Demand	0	1	2	3	1952	971
	Probability	40%	20%	20%	20%		
4	Demand	0	1	2	3	2502	1240
	Probability	25%	25%	25%	25%		

Some input parameters are determined for the case, some of these parameters are bounded by city regulations on vehicle characteristics. The transshipment times are obtained from Bijvoet [11].

Below, an overview of the parameter values used for the experiments is given. These values are the baseline for all experiments unless otherwise stated in the experiment description.

$q^{\mathrm{R}} = 15m^3$ capacity of road vehicles
$v^{\mathrm{V}} = 5m/s$ speed of road vehicles
$q^{\mathrm{W}} = 50m^3$ capacity of water vehicles
$v^{\mathrm{W}} = 1.6m/s$ speed of water vehicles
$t^{\mathrm{DC}} = 25min$ transshipment time at the depot
$t^{\mathrm{S}} = 3min$ transshipment time at satellites
$t^{\mathrm{C}} = 1.5min$ transshipment time at customers
$t^{\max} = 480min$ maximum time span
$q^{\mathrm{S}} = 0$ storage capacity of satellites

To quickly investigate some scenarios and the model's sensitivity, a smaller test set is created. This set exists of the Horeca locations in a busy city area, the "Wallen". This area contains 345 Horeca locations, which is approximately 21% of the total case.

D. Experiments

Experiments are conducted on the total case and the test set, which allow for the investigation of decision variables under different scenarios of interest, as well as validating the modelling approach used.

First, model settings are investigated, starting with the computation time for the sub-problems. Also, the different FLP strategies for limiting the number of customers assigned to satellites are tested. Second, system scenarios are explored, varying the number of opened satellites and the maximum time span. Third, sensitivity analyses are performed, to understand how the system responds to different parameter values and demand sets. Lastly, the overall system performance is evaluated.

1. Model settings

Computation time

The time limit parameter specifies the maximum amount of computation time allowed for the solver to find a solution. It is essential to strike a balance between computation time and solution quality, particularly in the context of large IWLT systems. While the optimisation model should produce results within a reasonable time frame, the definition of "reasonable time" in this application is nuanced. For these tests, the number of opened satellites is set to $N^{\mathrm{S}} = 15$. The rest of the parameters are specified in subsection III.C. To be able to investigate the impact of changing the time limit of one model, the computation time of that model is varied, while the time limits of the other problems stay at 7200s.

The facility location problem finds optimal solutions within 200s for the total case, so varying the computation time for the FLP is unnecessary.

Increasing the time limit for the second-echelon vehicle routing problem from 100s to 1000s reduces the distance on the roads significantly, and up to 3600s there is still some reduction visible. Increasing the time limit further does not improve the solution much further. A computation time of 3600s results in an optimality gap of 10%. The first-echelon vehicle routing problem combined with the synchronisation is a complex model.

Increasing the computation time does not have any visible effect up to 7200s. At 7200s, the distances on the roads and waterways decrease. The results do not change when increasing the computation time further up to 10800s. Varying the time limit for the road scheduling problem has a large impact on both the number of road vehicles required and the distance travelled on the roads. The results keep improving for increased computation times, but the effect is less significant for higher time limits. The optimality gap converges to approximately 6%.

The water vehicle scheduling model only affects the number of water vehicles required to perform the trips found by the first-echelon vehicle routing model. For a computation time of 3600s, the number of required vehicles decreases from 34 to 16, which is a reduction of more than 50%. Increasing the computation time to 10800s results in 13 required water vehicles, representing a reduction of 62%.

FLP strategies

Two variants to limit the customers assigned to satellites in the FLP are given before. For both of these constraints, many possible equations can be used that change the tightness of the constraint. It is possible to precisely even out the number of customers so each satellite has the same number of customers assigned, but this might not have the best results since some customers will be assigned to satellites further away. Some freedom can be implemented, allowing the assignment of more customers to satellites when that is more favourable for the distance travelled on the roads. It is investigated how much freedom is necessary to get good quality solutions while distributing the satellite utilisation.

Experiments with the constraints are performed on the Wallen neighbourhood defined for the demand distribution provided by Bijvoet [11], resulting in 151 Horeca locations with a total demand of $290m^3$ in the Wallen neighbourhood. The factors a and b are varied from 1 to 2.5 and the number of opened satellites N^S is set to 2 or 3.

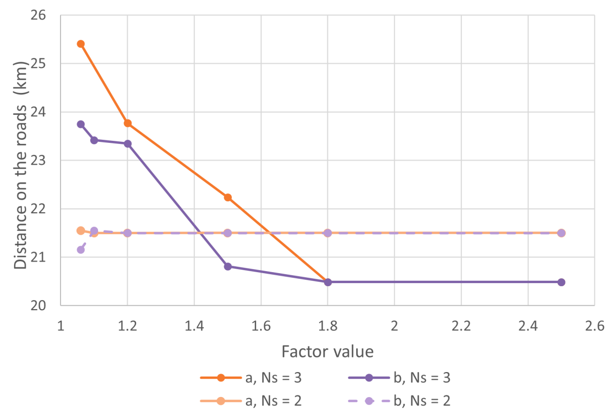


Figure 1. Total distance travelled on the roads under different FLP constraints for 2 and 3 opened satellites (Ns=2,3), factor a limits the throughput, factor b limits the number of customers

In Figure 1 the distances travelled on the roads under different FLP constraints are shown. As can be expected, loosening the constraint results in fewer kilometres travelled on the roads. However, allowing 1.5 times the evenly divided number of customers to be assigned to a satellite provides enough flexibility for near-optimal customer assignment to satellites while distributing the utilisation more evenly.

The impact of adjusting the maximum throughput constraint appears to have a larger negative impact on the road distance, compared to tightening the maximum customer constraint. This can be attributed to the need to assign customers with higher demand to more distant satellites under throughput constraints. While the constraint on the maximum number of customers allows for more favourable assignments by selecting the customer with the least additional distance, the throughput constraint might necessitate less optimal assignments.

2. *Scenarios*

Number of Satellites

One of the most important design choices for developing an IWLT system is the number of satellites to open. Having a small number of satellites in the city centre means these satellites are used intensively, which can create nuisance under city residents. However, a large number of satellites might also not be desirable since satellites require blockage of parking spaces and can congest the waterways when transshipment is taking place. Therefore, it is important to have insights into the effect of the number of satellites on the road and water kilometres, so these factors can be weighted and decisions can be made.

The model is run for 3 to 25 opened satellites to investigate the effect of the number of satellites, with the FLP constraint on the number of customers with $b = 1.5$.

It is interesting to analyse the systems performance for the results of the road scheduling problem first, since all scenarios use the same number of road vehicles after this scheduling problem because of the lower bound on this. Therefore, the results are not yet dependent on the extra distance travelled from and to the road vehicle depots by added vehicles and can be easily compared. The optimality gaps determined by Gurobi for these scenarios are approximately equal to the Optimality gaps for fewer opened satellites and the same number of road vehicles is used. Figure 2 shows the distances travelled on the roads found by the road vehicle scheduling problem and found after the combined scheduling problem. Looking at the distances after the road scheduling problem, it can be seen that the distance reduces substantially for each extra opened satellite for up to 9 satellites, is at a minimum for 12 opened satellites and starts to increase for extra opened satellites. This indicates the systems performance is better for 9 to 13 opened satellites, which can have three causes, first: the FLP constraint forces customers to be assigned to the extra opened satellites, even if these locations are less favourable, second: vehicles might have to travel more between satellites, third: the road vehicle depots might be located further away from some satellites. Investigating the results of the distance travelled after the combined scheduling model, the same trend is visible. Noteworthy is that no improvements on the distance is found in the combined scheduling problem for 16 or more opened satellites. The optimality gaps of the combined scheduling model determined by Gurobi for these scenarios are approximately equal to the optimality gaps for fewer opened satellites.

Concluding, opening 9 to 13 satellites seems to be a reasonable choice. For the remaining experiments, 12 satellites are opened, to ensure the system's adaptability to different scenarios.

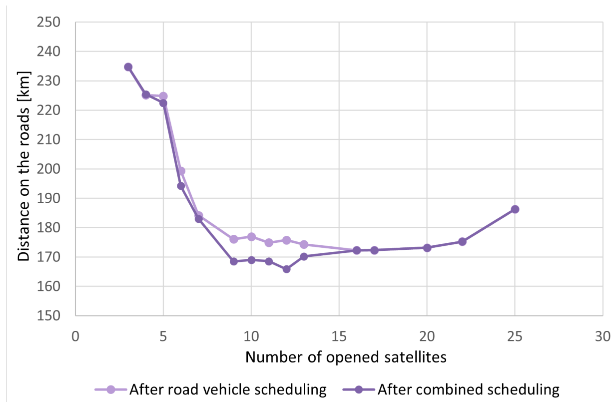


Figure 2. Distances travelled on the roads after road vehicle scheduling and combined scheduling, supplying the entire city centre with demand set 2, for 3 to 25 opened satellites

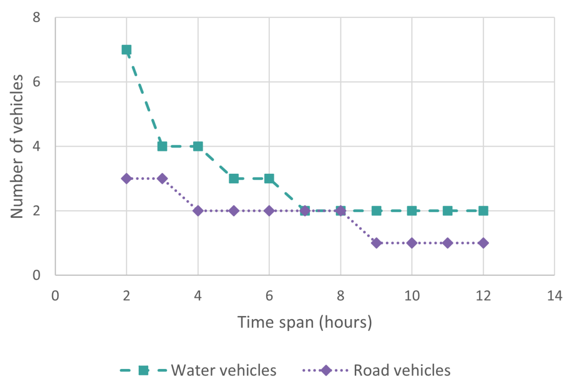
Time Span

The time span in which the deliveries are performed is crucial for the IWLT system to be feasible in real-life applications. The available time impacts the system requirements to serve all customers. To see the effect on these requirements, different maximum time spans are tested and the results investigated.

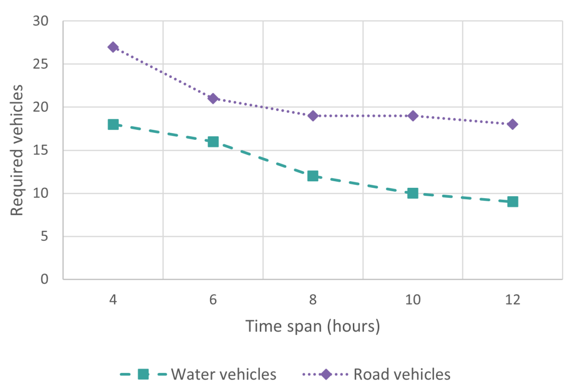
The maximum time spans ($t^{\max}$) evaluated are 4, 6, 8, 10, and 12 hours, with 12 satellites opened for the entire city center of Amsterdam, using the demand data provided by Bas-2023, specified in Table 1 set 2. Additional experiments are conducted for the Wallen neighborhood with time spans ranging from 2 to 12 hours, in increments of 1 hour, utilising two satellites. The demand distribution for these experiments is also based on set 2 but is limited to the Horeca locations in the Wallen neighbourhood.

The impact of increasing the time span can best be shown through the number of vehicles required, as shown in Figure 3, especially for vessels. Half of the vessels are required when extending the time span from 4 to 12 hours, which is expected since vessel trips have long completion times, so with a shorter time span, vehicles are not always able to perform multiple trips.

The decrease is also visible for road vehicles. However, the decrease is less significant. This phenomenon can be linked to the vessel schedule. Most of the vessels arrive at approximately the same time at satellites, so at that moment, many road vehicles are required as well.



(a) Wallen neighbourhood

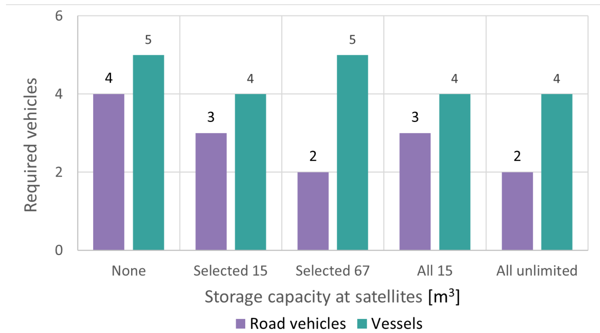


(b) Entire city centre

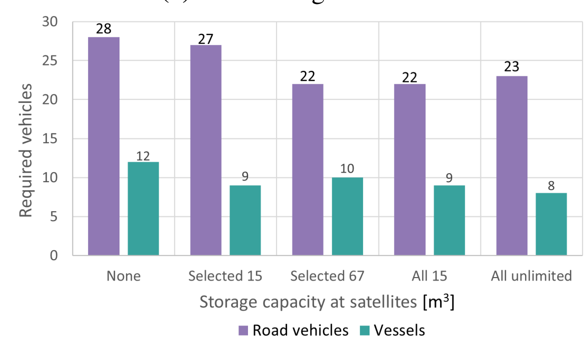
Figure 3. Required number of vehicles for varying time spans $t^{\max}$

Storage Capacity Satellites

Since space is scarce in most city centres, the basic scenario investigated assumes no storage capacity at satellites. However, at certain locations, some storage might be feasible, potentially enhancing system performance. Through field research, satellite locations with potential for storage are identified. To supply the entire city centre with 12 satellites, four of the locations show significant potential to incorporate storage facilities. In the Wallen neighbourhoods with four satellites, two of the satellites are feasible for storage.



(a) Wallen neighbourhood



(b) Entire city centre

Figure 4. Required number of vehicles for different storage scenarios at satellites

Figure 4 shows the required vehicles for different storage scenarios at satellites for the Wallen neighbourhood and the entire city centre. As can be seen, having 15m^3 storage capacity at the selected satellites lowers the number of vessels, from 5 to 4 for the Wallen neighbourhood and from 12 to 9 for the entire city centre, which are significant improvements. Further improvements are observed for the increased storage scenarios, with a small discrepancy at a storage capacity of 67m^3 for the selected satellites, where the number of vessels increase while the number of road vehicles decrease. This effect is likely due to the storage capacity of 67m^3 at the selected satellites exceeding the vessel capacity of 50m^3. Consequently, when the larger storage is utilised by the road vehicle schedule, it might necessitate more complex movements of the vessels to accommodate this utilisation.

The hypothetical scenario of unlimited storage capacity at all satellites further reduces the number of vessels, requiring only 23 road vehicles and 8 vessels. This scenario highlights the substantial impact of satellite storage capacity on the logistics network, demonstrating significant performance gains with storage. However, the most significant improvement in required vessels for the entire city centre is made when increasing the storage at the selected satellites from zero to 15m^3, indicating that having some storage available provides enough flexibility for the system to operate more efficiently.

3. Sensitivity Analyses

Demand Sets

Different demand sets are specified in subsection III.C. These demand sets are implemented in the full case scenario with 12 satellites and a time span of 8 hours. The required number of vehicles for each demand set are shown in Figure 5. The number of water and road vehicles increases approximately linearly with the size of the demand sets.

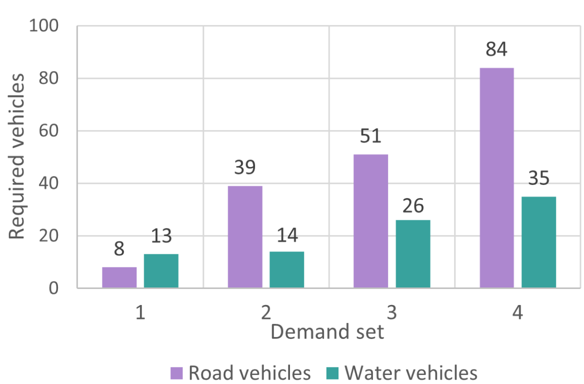


Figure 5. Number of required road and water vehicles for different demand sets

To get better insight in the influence of the demand and number of customers with demand, additional experiments are performed on the Wallen case. An overview of the demand sets is given in Table 2.

Table 2. Overview of the demand sets

Demand set	Total demand [m^3]	Customers with demand
1	220	114
2	290	151
3	428	212
4	541	263
5	679	345
6	687	279
7	345	345
8	870	174

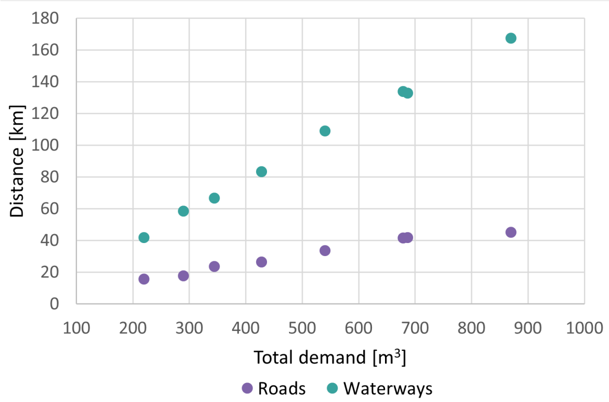
Figure 6 shows the results for the demand sets of Table 2 in the Wallen neighbourhood, with the total demand on the x-axis. As can be seen, the relation between the total demand and the results for the distances and number of vehicles is linear. The linear relationship between the total demand and the distances travelled indicates a predictable pattern in how demand affects distances. It suggests that the model is robust and reacts predictably to changes in demand, which is a desirable property for any decision-making tool. This robustness builds confidence in the model's use for real-life applications.

Transshipment Times

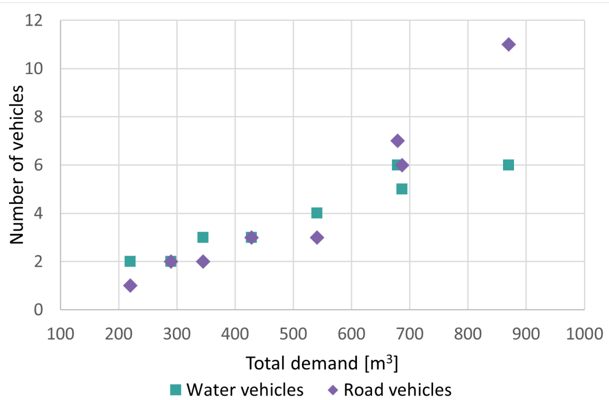
The transshipment time, particularly at customers, constitutes a significant portion of the total time span. The transshipment time often exceeds travel time in terms of duration, especially for road vehicles. The transshipment times used in the other experiments are based on the work of Bijvoet [11]. However, it is worth noting that these times seem to be optimistic and may not accurately reflect real-world scenarios.

This sensitivity analysis involves testing the IWLT system requirements under different transshipment times at customers. The analysis evaluates the system's sensitivity to changes in transshipment times and helps identify thresholds where performance may be significantly impacted.

The number of required road vehicles increases



(a) Distance travelled on the roads and waterways for different total demand in the Wallen neighbourhood



(b) Required number of vehicles for different total demand in the Wallen neighbourhood

Figure 6. Results for the demand sets in the Wallen neighbourhood

for longer transshipment times, however, more than tripling the transshipment time of $t^C = 1.5min$ in the entire city centre only requires 26% more road vehicles. Furthermore, doubling the transshipment time of $t^C = 5min$ only increases the road vehicle requirement by 16%. For the Wallen case, no increase in the number of vehicles is required for transshipment times up to 5 minutes.

Increasing the transshipment time at customers does also affect the number of water vehicles required, however less significant. Since road vehicles have longer trip times, water vehicles might have to wait longer at satellites, which can ultimately results in more required water vehicles.

With these results, the system does not appear to be overly sensitive to variability in customer transshipment times. When the road vehicles are not fully utilised, the increased transshipment times can be accommodated. When the transshipment time is increased further, a linear relation between the required number of road vehicles and increased time seems to exist.

4. *Overall system performance*

Based on the experimental analysis, it is essential to evaluate how the IWLT system performs compared to the current situation. Leveraging insights from the experiments, four system scenarios are selected to assess performance, identify bottlenecks, and compare the results with the current state. The scenarios represent various combinations of the key design choices, specified in Table 3

Table 3. Selected scenarios for performance evaluation

Scenario	Number of satellites	Time span [hours]	Satellite storage
A	9	4	None
B	12	8	None
C	12	8	$15m^3$ selected four
D	12	12	$15m^3$ all

For these scenarios, the model is solved with more allocated computational resources, specifically by allocating more CPUs and tasks, to ensure a comprehensive and accurate comparison of the IWLT system with the current situation. The FLP strategy used is to limit the number of customers with parameter $b = 1.5$, and the demand follows the distribution of set 2 Table 1. The parameters defined in subsection III.C do not change unless specified in Table 3.

In the current situation all deliveries are conducted via road transport. This situation represents the existing scenario and is modelled as a straightforward vehicle routing problem with capacity constraints. A single depot is placed at the city's border, ensuring that only the distances travelled within the city centre are considered. The vehicle characteristics are consistent with those used in the IWLT system, with a capacity $q^V = 15m^3$ and speed $v^V = 5m/s$.

Table 4 presents the distances travelled for both the current situation and the selected IWLT system scenarios. The IWLT system scenarios result in vehicle kilometres reductions of 22%, 24%, 27% and 28% compared to the current situation, for scenario A, B, C and D, respectively. These reductions are a positive step, but the primary goal of the IWLT system is to minimise distance on the roads. All three scenarios accomplish this goal with substantial reductions, 70% for scenario A, 71% for scenario B and 72% for scenario B and C, signifying major improvements over the current situation.

Table 4. Selected scenarios for performance evaluation

Scenario	Road kilometres	Water kilometres	Vehicle kilometres
Current	579	X	579
A	172	278	450
B	166	273	439
C	163	260	423
D	163	252	415

IV. Results

The Integrated Water-Land Transport (IWLT) system demonstrates significant potential for enhancing urban logistics, particularly in densely populated city centres like Amsterdam. The experiments reveal several crucial insights into the system's performance under varying scenarios and parameters, which can assist in implementation and further development.

Two methods were evaluated to limit the number of customers assigned to satellites: one based on maximum customers (factor b) and the other on maximum throughput (factor a). Experiments indicated that allowing more customers to be assigned to satellites results in fewer kilometres travelled on the roads, and a factor of $b = 1.5$ times the evenly divided number of customers per satellite provided a balance between optimal assignments and even distribution of satellite utilisation.

The optimal number of satellites was found to be between 9 and 13 for the entire Horeca sector in Amsterdam. Beyond 13 satellites, the system performance declined due to sub-optimal customer assignments and increased vehicle travel. Experiments with smaller customer sets indicated that the optimal number of satellites decreased linearly with the total demand. For a smaller city area like the Wallen neighbourhood, fewer satellites (2-4) were optimal based on demand sets.

Extending the maximum time span for transshipment operations significantly reduces the number of vehicles required. Longer time spans enable water vehicles to perform multiple trips, lessening the peak load on road vehicles.

The total demand has a linear relationship with the required number of vehicles and the distances travelled. Higher demand naturally necessitates more resources but follows a predictable pattern. The number of customers with demand shows a less clear relationship with system performance, highlighting that total demand volume is a more critical factor than the number of customers.

Increased transshipment times at customer locations result in a higher number of required road vehicles. However, the system shows resilience up to a point, accommodating increased transshipment times without a proportional increase in vehicle requirements. There is a minor increase in the number of required water vehicles with longer transshipment times, attributed to longer waiting times at satellites.

Implementing the IWLT system results in a 24% reduction in total vehicle kilometres. While this reduction is a promising result, the shift of a significant portion of the transportation burden to waterways is a strategic advantage, leveraging the underutilised canal network in Amsterdam. A 71% reduction in road kilometres is found compared to the current situation, which aligns with the system's primary objective of reducing the burden on the roads.

V. Conclusions

The results obtained for the case of Amsterdam provide realistic estimates for the required number of vehicles and demonstrate that the IWLT system is feasible for implementation in Amsterdam. Furthermore, the results indicate that the proposed IWLT system could significantly reduce the burden on the road by utilising waterways, thus decreasing urban traffic and associated environmental, societal, and economic aftereffects.

This research highlights several practical considerations for implementing Integrated Water- and Land-Based Transportation (IWLT) systems in urban logistics. One key finding is the significant impact of the time span on the number of water vehicles required. Since water vehicles are costly, minimising their number is crucial to making the system attractive for logistics service providers. The number of water vehicles needed is directly related to the number of trips they can complete within the given time span. However, the simultaneous arrival of water vehicles presents a scheduling challenge. To enhance the utilisation of road vehicles and overall system efficiency, it is suggested to stagger the loading times at the depot for water vehicles. This would prevent concurrent arrivals and allow for better synchronised schedules. Furthermore, it can be interesting to investigate making storage available at some suitable satellite locations. This could improve the scheduling, since water vehicles would have no waiting time. Storage capacity can be implemented in the model by adjusting the satellite stock constraints.

Another critical factor for practical applications is the number of satellites opened. Opening between 9 and 13 satellites is recommended to effectively supply the entire Horeca sector in Amsterdam. If the municipality aims to further distribute the logistical burden, opening up to 20 satellites can still yield favourable results. At least 9 satellites should be opened to achieve significant reductions in road distance travelled and to ensure optimal system performance.

The experiments conducted on the test case offer valuable insights. Given the significant investment required to implement an IWLT system, it may be prudent to start with a smaller, more focused system targeting a critical area of the city centre. For instance, supplying the "Wallen" area, which includes 345 Horeca locations, demands substantially fewer resources than servicing the entire city centre. A system with just two water vehicles, two road vehicles, and two satellites is sufficient to meet the demands of this area.

Combining the results of the performed experiments, four distinct IWLT system scenarios to supply Horeca in the entire city centre were created and evaluated. Comparing the performance of these scenarios with the current situation, where all deliveries are conducted via road transport, the IWLT system scenarios achieve substantial reductions in distances on the roads. Specifically, vehicle kilometres are reduced by 22% to 28%, depending on the scenario. The primary objective of minimising road distance is successfully accomplished, with reductions of 70% to 72% compared to the current situation.

These findings suggest that the IWLT system can lead to a more efficient urban logistics operation, reducing traffic congestion and environmental impact. The system's performance improves with longer operational time spans and shows resilience to variations in demand and transshipment times. This makes it a viable option for cities looking to optimise their logistics networks.

The developed model is capable of handling large problem instances and provides feasible solutions for real-life applications. The model offers valuable insights for logistic service providers and system designers, facilitating the development of efficient transportation systems.

References

[1] Daggers, T., and Heidenreich, J., "City Logistics with Electric Vehicles," , 11 2013.

[2] Benjelloun, A., Crainic, T. G., and Bigras, Y., "Towards a taxonomy of City Logistics projects," *Procedia - Social and Behavioral Sciences*, Vol. 2, No. 3, 2010, pp. 6217–6228. doi: 10.1016/j.sbspro.2010.04.032, URL http://dx.doi.org/10.1016/j.sbspro.2010.04.032.

[3] "Towards sustainable urban distribution using city canals: the case of Amsterdam," ????

[4] Economic, E., and Committee, S., "NAIADES

II: Proposal for a Regulation of the European Parliament and of the Council amending Council Regulation on a Community-fleet capacity policy to promote inland waterway transport," , 1 2014. URL https://transport.ec.europa.eu/transport-modes/inland-waterways/promotion-inland-waterway-transport/naiades-ii_en.

[5] Sluijk, N., Florio, A. M., Kinable, J., Dellaert, N., and Van Woensel, T., "Two-echelon vehicle routing problems: A literature review," *European Journal of Operational Research*, Vol. 304, No. 3, 2023, pp. 865–886. doi: 10.1016/j.ejor.2022.02.022, URL http://dx.doi.org/10.1016/j.ejor.2022.02.022.

[6] Marquès, G., Sadykov, R., Dupas, R., and Deschamps, J.-C., "A Branch-Cut-and-Price Approach for the Single-Trip and Multi-Trip Two-Echelon Vehicle Routing Problem with Time Windows," *Transportation Science*, Vol. 56, No. 6, 2020a, pp. 1598–1617. doi: 10.1287/trsc.2022.1136, URL http://dx.doi.org/10.1287/trsc.2022.1136.

[7] Karademir, C., Alves Beirigo, B., Negenborn, R., and Atasoy, B., "Two-echelon Multi-trip Vehicle Routing Problem with Synchronization for An Integrated Water- and Land-based Transportation System," 2022. HEART 2022: 10th Symposium of the European Association for Research in Transportation; Conference date: June 2022.

[8] Li, H.-Q., Liu, Y., Kaihang, C., and Lin, Q., "The two-echelon city logistics system with on-street satellites," *Computers and Industrial Engineering*, Vol. 139, 2020, p. 105577. doi: 10.1016/j.cie.2018.12.024.

[9] Jia, f., Lei, given i=Q, f., given=Quanwu, and given i=Y, f., given=Yun-Kai, "An adaptive large neighborhood search heuristic for multi-commodity two-echelon vehicle routing problem with satellite synchronization," Vol. 19, No. 2, ????, p. 1187. doi: 10.3934/jimo.2021225, URL https://doi.org/10.3934/jimo.2021225.

[10] Anderluh, A., Nolz, P. C., Hemmelmayr, V. C., and Crainic, T. G., "Multi-objective optimization of a two-echelon vehicle routing problem with vehicle synchronization and 'grey zone' customers arising in urban logistics," *European Journal of Operational Research*, Vol. 289, No. 3, 2021, pp. 940–958. doi: 10.1016/j.ejor.2019.07.049, URL http://dx.doi.org/10.1016/j.ejor.2019.07.049.

[11] Bijvoet, "Multimodal city logistics using waterways in Amsterdam: proof-of-concept of a two- echelon distribution network," ???? URL http://resolver.tudelft.nl/uuid:a1123f0a-3995-4675-8ef4-02c0fe5a4686.

[12] Contardo, C., Hemmelmayr, V., and Crainic, T. G., "Lower and upper bounds for the two-echelon capacitated location-routing problem," *Computers and Operations Research*, Vol. 39, No. 12, 2012, pp. 3185–3199. doi: 10.1016/j.cor.2012.04.003, URL http://dx.doi.org/10.1016/j.cor.2012.04.003.

[13] Mirhedayatian, S. M., Crainic, T. G., Guajardo, M., and Wallace, S. W., "A two-echelon location-routing problem with synchronisation," *Journal of the Operational Research Society*, Vol. 72, No. 1, 2019, pp. 145–160. doi: 10.1080/01605682.2019.1650625, URL http://dx.doi.org/10.1080/01605682.2019.1650625.

[14] Escobar-Vargas, D., Crainic, T. G., and Contardo, C., "Synchronization in Two-Echelon Distribution Systems: Models, Algorithms, and Sensitivity Analyses," *CIRRELT*, Vol. 50, 2021.

[15] Brockhoff, "Multi-Modal Last-Mile Delivery: Developing Integrated Water- and Land-based Transportation Systems for City Logistics," , ????

B

Appendix: Python code

B.1. Facility Location Problem

```python
import gurobipy as gb
from gurobipy import quicksum, GRB
import time
import os
import numpy as np
import pandas as pd
import math


#%% Import data
path = os.getcwd() + "/Inputs/"
t_lim_FLP = int(os.getenv('t_lim_FLP'))
mip_FLP_str = os.getenv('mip_FLP')
mip_FLP = float(mip_FLP_str)

def FLP_num_cust(Ns,
         number,
         customers,
         df_horeca_data_info,
         satellite_locations,
         horeca_sets,
         directed,
         df_SE_shortest_dist_directed_True,
         df_SE_shortest_dist_directed_False,
         df_horeca_demand_scenarios):

    path_out = os.getcwd() + "/Outputs/"
    model = gb.Model('FLP')
    np.random.seed(123)

    if directed == 'true':
        df_SE_shortest_dist = df_SE_shortest_dist_directed_True.fillna(10001)
    elif directed == 'false':
        df_SE_shortest_dist = df_SE_shortest_dist_directed_False


    #%% Variables

    # Opening satellites
    y = {}
    for satellite_id in satellite_locations.index.tolist():
        y[satellite_id] = model.addVar(vtype = GRB.BINARY, name = f'y[{satellite_id}]')

    # Customer assignment
    Y = {}
    for satellite_id in satellite_locations.index.tolist():
        for customer_id in customers.index.tolist(): #df_horeca_data_info.index.tolist():
            Y[satellite_id, customer_id] = model.addVar(vtype = GRB.BINARY, name = f'Y[{satellite_id}, {customer_id}]')


    #%% Objective function
```

```python
model.setObjective(quicksum(df_SE_shortest_dist.at[satellite_locations.at[f'{satellite_id}','road_node'],df_horeca_data_info.at[f'{customer_id}','road_node']] * Y[satellite_id, customer_id] for satellite_id in satellite_locations.index.tolist() for customer_id in customers.index.tolist()))
model.modelSense = GRB.MINIMIZE
model.update()

#%% Constraints
for satellite_id in satellite_locations.index.tolist():
        max_customers = int(len(customers)/Ns + number/Ns)
        constr_capacity = model.addConstr(quicksum(Y[satellite_id, customer_id] for customer_id in customers.index.tolist()) <= max_customers)

# Customer assignment: each customer is assigned to one satellite.
for customer_id in customers.index.tolist(): #df_horeca_data_info.index.tolist():
    constr_assignment = model.addConstr(quicksum(Y[satellite_id, customer_id] for satellite_id in satellite_locations.index.tolist()) == 1 )

# Opening constraint: a satellite needs to be open to assign customers.
for satellite_id in satellite_locations.index.tolist():
    for customer_id in customers.index.tolist(): #df_horeca_data_info.index.tolist():
        constr_opening = model.addConstr(Y[satellite_id, customer_id] <= y[satellite_id])

# Number of satellites constraint: the number of satellites opened is equal to the set number of satellites.
constr_Ns = model.addConstr(quicksum(y[satellite_id] for satellite_id in satellite_locations.index.tolist()) <= Ns)

#%% Solve the MIP problem
print("start optimizing")
model.setParam( 'OutputFlag', True)
model.setParam ('MIPGap', mip_FLP);
model.setParam('Seed', 123)
model.setParam('Timelimit', t_lim_FLP)
model._obj = None
model._bd = None
model._obj_value = []
model._time = []
model._start = time.time()
model.optimize()
mip_gap_FLP = model.MIPGap
obj_FLP = model.getObjective().getValue()
satellites_chosen = pd.DataFrame({'index':satellite_id, 'id': satellite_locations.at[satellite_id, 'id'], 'road_node': satellite_locations.at[satellite_id, 'road_node'], 'canal_node': satellite_locations.at[satellite_id, 'canal_node'], 'available': y[satellite_id].X } for satellite_id in satellite_locations.index.tolist())
satellites_chosen = satellites_chosen.set_index('index')


#%%
customer_assignment_set_1 = customer_assignment_set_2 = customer_assignment_set_3 = customer_assignment_set_4 = customer_assignment_set_5 = customer_assignment_set_6 = customer_assignment_set_7 = customer_assignment_set_8 = customer_assignment_set_9 = customer_assignment_set_10 = pd.DataFrame(columns=['SE_node', 'demand', 'assigned', 'via_satellite', 'by_sev'])

for horeca_set in horeca_sets:
    assigned = []
    satellite = []
    indices = []
    for customer_id in customers.index.tolist():
        for satellite_id in satellite_locations.index.tolist():
            if Y[satellite_id, customer_id].x == 1:
                distance = df_SE_shortest_dist.at[satellite_locations.at[f'{satellite_id}','road_node'],df_horeca_data_info.at[f'{customer_id}','road_node']]
                if df_horeca_demand_scenarios.at[f'{customer_id}', f'set_{horeca_set}'] > 0:
                    assigne = True
                    dist_sat_hor = df_SE_shortest_dist.at[satellite_locations.at[f'{satellite_id}','road_node'],df_horeca_data_info.at[f'{customer_id}','road_node']]
                    dist_hor_sat = df_SE_shortest_dist.at[df_horeca_data_info.at[f'{
```

```
customer_id}','road_node'],satellite_locations.at[f'{satellite_id}','road_node']]
                    tot_dist = dist_hor_sat + dist_sat_hor
                    if dist_hor_sat > 10000 or dist_sat_hor > 10000 or math.isnan(dist_hor_sat) or math.isnan(dist_sat_hor):
                        assigne = False
                    indices.append(customer_id)
                    assigned.append({'SE_node':int(df_horeca_data_info.at[customer_id, 'road_node']), 'demand':int(df_horeca_demand_scenarios.at[f'{customer_id}', f'set_{horeca_set}']),'assigned': assigne,'via_satellite':satellite_id, 'dist_sat_hor': int(dist_sat_hor), 'dist_hor_sat': int(dist_hor_sat), 'tot_dist': int(tot_dist)} )
        if horeca_set == 1:
            customer_assignment_set_1 = pd.DataFrame(assigned, index= indices)
        if horeca_set == 2:
            customer_assignment_set_2 = pd.DataFrame(assigned, index= indices)
        if horeca_set == 3:
            customer_assignment_set_3 = pd.DataFrame(assigned, index= indices)
        if horeca_set == 4:
            customer_assignment_set_4 = pd.DataFrame(assigned, index= indices)
        if horeca_set == 5:
            customer_assignment_set_5 = pd.DataFrame(assigned, index= indices)
        if horeca_set == 6:
            customer_assignment_set_6 = pd.DataFrame(assigned, index= indices)
        if horeca_set == 7:
            customer_assignment_set_7 = pd.DataFrame(assigned, index= indices)
        if horeca_set == 8:
            customer_assignment_set_8 = pd.DataFrame(assigned, index= indices)
        if horeca_set == 9:
            customer_assignment_set_9 = pd.DataFrame(assigned, index= indices)
        if horeca_set == 10:
            customer_assignment_set_10 = pd.DataFrame(assigned, index= indices)

    return (satellites_chosen, customer_assignment_set_1, customer_assignment_set_2, customer_assignment_set_3,
            customer_assignment_set_4, customer_assignment_set_5, customer_assignment_set_6,
            customer_assignment_set_7, customer_assignment_set_8, customer_assignment_set_9, customer_assignment_set_10, mip_gap_FLP, obj_FLP)
```

B.2. Second-Echelon Trip Generation

```
# Old file: Total_model_FLP_VRPs_MIP_times_parameters.py

#%% Import libraries
import gurobipy as gb
import time
import os
import numpy as np
import pandas as pd
import pickle
import copy
from gurobipy import quicksum, GRB
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

#%% Set path
server = 'True'
print('Split models')

if server == 'False':
    path = os.getcwd() + "\Inputs\\"
    path_out = os.getcwd() + "\Outputs\\"
    from FLP_solver_definition_number_customers_horeca_sets_Laudy import FLP_num_cust
    from FLP_solver_definition_horeca_sets_capacity_assignment import FLP_capacity

if server == 'True':
    path = os.getcwd() + "/Inputs/"
    path_out = os.getcwd() + "/Outputs/"
    from FLP_solver_definition_number_customers_horeca_sets_Laudy_server_numcust_demand_storage import FLP_num_cust
    from FLP_solver_definition_horeca_sets_capacity_assignment_server import FLP_capacity

```

```python

#%% Scenario inputs
directed = 'true'              # Indicate wether to use directed or undirected distance matrix
FLP_constraint = 'num_cust'    # Which FLP constraint to use, either capacity or num_cust
Nc = 750                       # Insert the number of customers to consider
horeca_sets = np.arange(1,11)  # Which horeca sets to evaluate
horeca_set = 1                 # If not testing all horeca sets, insert one to evaluate


#%% Import network and scenario data
df_horeca_demand_scenarios = pd.read_excel(path + f'df_horeca_demand_scenarios.xlsx', index_col=0)
df_horeca_demand_scenarios.index = df_horeca_demand_scenarios.index.astype(str)
df_horeca_data_info = pd.read_excel(path + f'df_horeca_data_info.xlsx', index_col=0)
df_horeca_data_info.index = df_horeca_data_info.index.astype(str)
customer_locations = df_horeca_data_info.iloc[:,0]

if server == 'False':
    df_SE_shortest_dist_directed_False = pickle.load(open(path + 'df_SE_shortest_dist_directed-False_nodes_all.pickle', 'rb'))
    df_SE_shortest_dist_directed_True_1 = pickle.load(open(path + 'df_SE_shortest_dist_directed-True_nodes_all.pickle', 'rb'))
    df_SE_shortest_dist_directed_True = df_SE_shortest_dist_directed_True_1.fillna(1001)
    dict_FE_shortest_dist_directed_True_1 = pickle.load(open(path + 'dict_FE_shortest_dist_directed-True_nodes_all.pickle', 'rb'))

if server == 'True':
    pickle_off = open(path + 'df_SE_shortest_dist_directed-True_nodes_all.pickle', 'rb')
    df_SE_shortest_dist_directed_True_1 = pd.read_pickle(pickle_off)
    df_SE_shortest_dist_directed_True = df_SE_shortest_dist_directed_True_1.fillna(1001)

    pickle_off = open(path + 'dict_FE_shortest_dist_directed-True_nodes_all.pickle', 'rb')
    dict_FE_shortest_dist_directed_True_1 = pd.read_pickle(pickle_off)
    df_SE_shortest_dist_directed_False = dict_FE_shortest_dist_directed_True_1
assigned = []
indices = []
customers = [[0]*3]*len(customer_locations)
for customer_id in df_horeca_data_info.index.tolist():
    if df_horeca_demand_scenarios.at[f'{customer_id}', f'set_{horeca_set}'] > 0:
        indices.append(customer_id)
        assigned.append({'road_node':int(df_horeca_data_info.at[customer_id, 'road_node']), 'demand':int(df_horeca_demand_scenarios.at[f'{customer_id}', f'set_{horeca_set}'])} )
customers = pd.DataFrame(assigned, index= indices)# df_horeca_data_info.index.tolist() )


satellite_locations = pd.read_excel(path + "satellite_nodes_storage_full.xlsx", index_col=0)
vehicles = pd.read_excel(path +"Road_vehicles.xlsx", index_col=0)
road_nodes = pd.read_excel(path + "satellites_customers_road_nodes.xlsx", index_col = 0)

if directed == 'true':
    dist = df_SE_shortest_dist_directed_True
elif directed == 'false':
    dist = df_SE_shortest_dist_directed_False

#%% Parameters
speed_v = int(os.getenv('speed_v'))
tranship_s = int(os.getenv('tranship_s'))
tranship_c = int(os.getenv('tranship_c'))
fev_profile = 5
capacity_fe = int(os.getenv('capacity_fe'))
speed_fe_str =  os.getenv('speed_fe')
speed_fe = float(speed_fe_str)
service_time_fe = int(os.getenv('service_time_fe'))
capacity_s = int(os.getenv('capacity_s'))
capacity_se = int(os.getenv('capacity_se'))
Ns = int(os.getenv('NrSatellites'))
number = int(os.getenv('number'))

t_limits_VRP_E2_str = os.getenv('t_limits_VRP_E2')
t_limits_VRP_E2 = eval(t_limits_VRP_E2_str)
```

```
t_lim_VRP_E1 = int(os.getenv('t_lim_VRP_E1'))
t_lim_sched_road = int(os.getenv('t_lim_sched_road'))
t_lim_sched_water = int(os.getenv('t_lim_sched_water'))
t_lim_sched_total = int(os.getenv('t_lim_sched_total'))
time_span = int(os.getenv('time_span'))
mip_VRP_E2_str = os.getenv('mip_VRP_E2')
mip_VRP_E2 = float(mip_VRP_E2_str)
mip_VRP_E1_str = os.getenv('mip_VRP_E1')
mip_VRP_E1 = float(mip_VRP_E1_str)
mip_sched_r_str = os.getenv('mip_sched_r')
mip_sched_r = float(mip_sched_r_str)
mip_sched_w_str = os.getenv('mip_sched_w')
mip_sched_w = float(mip_sched_w_str)
mip_sched_t_str = os.getenv('mip_sched_t')
mip_sched_t = float(mip_sched_t_str)
storage_set = os.getenv('storage_set')
save_title = os.getenv('save_title')

capacity_s = {}
for i in satellite_locations.index.tolist():
    capacity_s[i] = satellite_locations.at[i,f'capacity_{storage_set}']
    if capacity_s[i] > 0:
        print(i, capacity_s[i])

#%%
S_DC = {}
df_fe_distance_matrix = dict_FE_shortest_dist_directed_True_1[f'vessel_profile_{fev_profile}'
    ].copy()
dict_FE_new = pd.read_csv(path + 'distance_matrix_DCs.csv',sep=';',header=None)
dist_fe_new = pd.DataFrame(dict_FE_new)
dist_fe_new.index = dist_fe_new.index + 1
new_index = {old_index:old_index + 1 for old_index in dist_fe_new.columns}
dist_fe_new = dist_fe_new.rename(columns=new_index)
dist_fe = dist_fe_new.fillna(99999)


#%% Sets
vessels_total = pd.read_excel(path +"Water_vehicles.xlsx", index_col=0)
vessels = vessels_total
W_id = vessels.index.tolist()
zero = ['zero']
W0_id = zero + W_id

#%% Start loop over number of satellites
N_s = []
results = []
for t_lim_VRP_E2 in t_limits_VRP_E2:
    print('Number of customer value FLP: ', number)
    print('FLP for Ns:', Ns)
    start_time_FLP = time.time()
    if FLP_constraint == 'num_cust':
        if server == 'True':
            satellites_new, customer_assignment_set_1, customer_assignment_set_2,
    customer_assignment_set_3, customer_assignment_set_4, customer_assignment_set_5,
    customer_assignment_set_6, customer_assignment_set_7, customer_assignment_set_8,
    customer_assignment_set_9, customer_assignment_set_10, mip_gap_FLP, obj_FLP =
    FLP_num_cust(Ns,
                        number,
                        customers,
                        df_horeca_data_info,
                        satellite_locations,
                        horeca_sets,
                        directed,
                        df_SE_shortest_dist_directed_True,
                        df_SE_shortest_dist_directed_False,
        df_horeca_demand_scenarios)
        elif server == 'False':
            satellites_new, customer_assignment_set_1, customer_assignment_set_2,
    customer_assignment_set_3, customer_assignment_set_4, customer_assignment_set_5,
    customer_assignment_set_6, customer_assignment_set_7, customer_assignment_set_8,
    customer_assignment_set_9, customer_assignment_set_10 = FLP_num_cust(Ns,
```

```
                    df_horeca_data_info,
                    satellite_locations,
                    horeca_sets,
                    directed,
                    df_SE_shortest_dist_directed_True,
                    df_SE_shortest_dist_directed_False)
    elif FLP_constraint == 'capacity':
        satellites_new, customer_assignment_set_1, customer_assignment_set_2,
    customer_assignment_set_3, customer_assignment_set_4, customer_assignment_set_5,
    customer_assignment_set_6, customer_assignment_set_7, customer_assignment_set_8,
    customer_assignment_set_9, customer_assignment_set_10 = FLP_capacity(Ns,
                df_horeca_data_info,
                satellite_locations,
                directed,
                df_SE_shortest_dist_directed_True,
                df_SE_shortest_dist_directed_False,
                horeca_sets)


    if horeca_set == 1:
        customer_assignment_set_1 = customer_assignment_set_1# df_horeca_data_info.index.
    tolist() )
    if horeca_set == 2:
        customer_assignment_set_1 = customer_assignment_set_2# df_horeca_data_info.index.
    tolist() )
    if horeca_set == 3:
        customer_assignment_set_1 = customer_assignment_set_3# df_horeca_data_info.index.
    tolist() )
    if horeca_set == 4:
        customer_assignment_set_1 = customer_assignment_set_4# df_horeca_data_info.index.
    tolist() )
    if horeca_set == 5:
        customer_assignment_set_1 = customer_assignment_set_5# df_horeca_data_info.index.
    tolist() )
    if horeca_set == 6:
        customer_assignment_set_1 = customer_assignment_set_6# df_horeca_data_info.index.
    tolist() )
    if horeca_set == 7:
        customer_assignment_set_1 = customer_assignment_set_7# df_horeca_data_info.index.
    tolist() )
    if horeca_set == 8:
        customer_assignment_set_1 = customer_assignment_set_8# df_horeca_data_info.index.
    tolist() )
    if horeca_set == 9:
        customer_assignment_set_1 = customer_assignment_set_9# df_horeca_data_info.index.
    tolist() )
    if horeca_set == 10:
        customer_assignment_set_1 = customer_assignment_set_10# df_horeca_data_info.index.
    tolist() )

    available_satellites = satellites_new[satellites_new['available'] == 1].index.tolist()
    end_time_FLP = time.time()
    solving_time_FLP = end_time_FLP - start_time_FLP

    # Create sets for VRP E2
    indi = []
    assignment = []
    c_a = []
    c_aa = []
    sc_a = []
    sc_aa = []
    for s in available_satellites:
        indi.append(s)
        c_assignment = []
        sc_assignment = [s]
        for customer_id in customer_assignment_set_1[0:Nc].index.tolist():#indices:
            if customer_assignment_set_1.at[f'{customer_id}','via_satellite'] == s:
                c_assignment.append(customer_id)
                sc_assignment.append(customer_id)

        c_a.append((s,c_assignment))
```

```
    sc_a.append((s,sc_assignment))
    assignment.append((s,c_assignment))
c_aa = pd.DataFrame(c_a, index = indi)
sc_aa = pd.DataFrame(sc_a, index = indi)

#%%
road_node = []
canal_node = []

DC = ['DC_1','DC_2','DC_3']
DC_canal_nodes = [387,127,389]
canal_node_d = [{'canal_node': DC_canal_nodes}]

canal_nodes_d = pd.DataFrame({'canal_node':DC_canal_nodes}, index = DC)
zer = 0
road_nodes_s = [[0]*3]*len(available_satellites)
canal_nodes_s = [[0]*3]*len(available_satellites)
for satellite in available_satellites:
    road_node.append({'road_node':satellites_new.at[satellite,'road_node'], 'demand': zer})
    canal_node.append({'canal_node':satellites_new.at[satellite,'canal_node']})
road_nodes_s = pd.DataFrame(road_node, index = available_satellites)
canal_nodes_s = pd.DataFrame(canal_node, index = available_satellites)
road_nodes_s_c = pd.concat([road_nodes_s, customers], ignore_index=False)
canal_nodes_d_s = pd.concat([canal_nodes_d, canal_nodes_s], ignore_index=False)

#%%
R_ids = vehicles.index.tolist()
R_id = R_ids[0:Ns]
S_id = available_satellites
C_new = customers.index.tolist()
C_id = C_new[0:Nc]
SC_id = S_id + C_id
DS_id = DC + S_id
print('Number of customers: ', len(C_id))
r_s = {}            # set of satellites assigned to vehicle r
r_c = {}            # set of customers assigned to vehicle r
r_sc = {}           # set of customers and satellites assigned to vehicle r
s_c = {}            # set of customers assigned to satellite s
s_sc = {}           # set of satellite and customers of satellite s
s_r = {}            # set of vehicles assigned to satellite s
vehicle_numb = 0
indi_r = []
for r in R_id:
    indi_r.append(r)
    r_s[r] = c_aa[0][vehicle_numb]
    r_c[r] = c_aa[1].get([r_s[r]])[0]
    r_sc[r] = sc_aa[1].get([r_s[r]])[0]
    if vehicle_numb >= len(c_aa[0])-1:
        vehicle_numb = 0
    elif vehicle_numb < len(c_aa[0])-1:
        vehicle_numb +=1

r_s_df = []
r_s_df = pd.DataFrame(r_s,index = [0]).transpose()

s_rr = {}
for s in S_id:
    s_r[s] = r_s_df[r_s_df[0] == s].index.tolist()
    s_c[s] = c_aa[1].get([s])[0]
    s_sc[s] = sc_aa[1].get([s])[0]



#%%
# Prefetch data
canal_nodes_dict = canal_nodes_d_s.loc[:,'canal_node']
road_nodes_dict = road_nodes_s_c.loc[:,'road_node']
r_transship_t_c_dict = vehicles.loc[:, 'transship_t_c']
r_speed_dict = vehicles.loc[:, 'speed']
```

```
#%% Create initial solution for VRP E2
print('Creating initial solution VRP E2 for Ns:', Ns)
X_R_init = {}
Q_R_init = {}
iterations = np.arange(0,500)
for r in R_id:
    for i in r_sc[r]:
        Q_R_init[i,r] = 0
        for j in r_sc[r]:
            X_R_init[i,j,r] = 0

r_c_left = copy.deepcopy(r_c)
for r in R_id:

    i = r_sc[r][0]
    load_r = 0
    for n in iterations:
        dis_old = 99999
        if len(r_c_left[r]) == 0:
            X_R_init[i,r_sc[r][0],r] = 1
            break
        for j in r_c_left[r]:
            dis = dist.at[road_nodes_dict[i],road_nodes_dict[j]]
            if dis < dis_old:
                dis_old = dis
                c = j
        load_r += df_horeca_demand_scenarios.at[f'{c}', f'set_{horeca_set}']
        if capacity_se >= load_r:
            Q_R_init[c,r] = df_horeca_demand_scenarios.at[f'{c}', f'set_{horeca_set}']
            X_R_init[i,c,r] = 1
            i = c
            r_c_left[r].remove(c)
        if capacity_se < load_r:
            X_R_init[i,r_sc[r][0],r] = 1
            load_r = 0
            i = r_sc[r][0]


#%% Get initial road km
D_r_init = 0

for r in R_id:
    for i in r_sc[r]:
        for j in r_sc[r]:
            if X_R_init[i,j,r] == 1:
                D_r_init += dist.at[road_nodes_dict[i],road_nodes_dict[j]]
print('Distance on the roads for initial solution: ', D_r_init)
#%% Get initial road trips
nr_trips = 0
for r in R_id:
        i = r_sc[r][0]
        for j in r_sc[r]:
            if X_R_init[i,j,r] == 1:
                nr_trips += 1
Nr_v_init = nr_trips

#%% VRP E2
print('Working on VRP E2 for Ns:', Ns)

start_time_VRP_road = time.time()
model = gb.Model('VRP_E2')
np.random.seed(123)
time_limit = t_lim_VRP_E2

# Path from i to j, if used by vehicle r: = 1, else: = 0
X_R = {}
for r in R_id:
    for i in r_sc[r]:
        for j in r_sc[r]:
            X_R[i,j,r] = model.addVar(vtype = GRB.BINARY, name = 'X_Ra')
```

```
    # Arrival time of vehicle r at i
    A_R = {}
    for r in R_id:
        for i in r_sc[r]:
            A_R[i,r] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'A_R')

    # Quantity delivered to customer i or picked up at satellite i by vehicle r
    Q_R = {}
    for r in R_id:
        for i in r_sc[r]:
            Q_R[i,r] = model.addVar(vtype = GRB.INTEGER, name = 'Q_R')

    # Customer or satellite is visited by vehicle r: = 1, if not: = 0
    Z_R = {}
    for r in R_id:
        for i in r_sc[r]:
            Z_R[i,r] =  model.addVar(vtype = GRB.BINARY, name = 'Z_R')

    # Accumulated load of road vehicle r at customer i
    L_R = {}
    for r in R_id:
        for i in r_sc[r]:
            L_R[i,r] =  model.addVar(vtype = GRB.INTEGER, name = 'L_R')

    # Distance travelled per vehicle r
    D_R = {}
    for r in R_id:
        D_R[r] = model.addVar(lb =0.0, vtype = GRB.INTEGER, name = 'D_R')

    # Load picked up at satellites by vehicle r
    W_R = {}
    for r in R_id:
        r_s[r] = r_s[r] if isinstance(r_s[r], list) else [r_s[r]]
        for i in r_s[r]:
            W_R[i,r] = model.addVar(lb=0.0, vtype = GRB.INTEGER, name = 'W_R')


    # Objective function
    model.setObjective(quicksum(D_R[r] for r in R_id))

    model.modelSense = GRB.MINIMIZE
    model.update()


    # Constraints
    model.update()

    # 1. A vehicle never goes from i to i
    for r in R_id:
        for i in r_sc[r]:
            for j in r_sc[r]:
                if i == j:
                    constr_self = model.addConstr(X_R[i,j,r] == 0, name = 'Constr_1')

    # 2b. Each satellite has to be visited at least the number of times needed for the demand
      of the customers divided by the vehicle capacity
    for s in S_id:
        s_r[s] = s_r[s] if isinstance(s_r[s], list) else [s_r[s]]
        constr_visit_2b = model.addConstr(quicksum(X_R[s,j,r]  for j in s_sc[s] for r in s_r[
    s]) >= quicksum(df_horeca_demand_scenarios.at[f'{i}', f'set_{horeca_set}'] for i in s_c[s
    ])/ capacity_se, name = 'Constr_2b')

    # 3. Vehicle r can only leave node if it also arrived there
    for r in R_id:
        for i in r_sc[r]:
                constr_arrival = model.addConstr(quicksum(X_R[i,j,r] for j in r_sc[r]) ==
    quicksum(X_R[j,i,r] for j in r_sc[r]), name = 'Constr_3')

    # 4. Nodes that are visited by vehicle r
    for r in R_id:
        for i in r_sc[r]:
```

```
        model.addConstr(Z_R[i,r] <= quicksum(X_R[i,j,r] for j in r_sc[r]))
        for j in r_sc[r]:
            constr_visits_r = model.addGenConstrIndicator(X_R[i,j,r], True, Z_R[i,r],GRB.EQUAL, 1, name = 'Constr_4')

# 2. Each customer i has to be visited by at least one vehicle r
for s in S_id:
    s_r[s] = s_r[s] if isinstance(s_r[s], list) else [s_r[s]]
    for i in s_c[s]:
        constr_visit_new = model.addConstr(quicksum(Z_R[i,r] for r in s_r[s]) >= 1, name = 'Constr_2')

# 5. The demand delivered to i is zero if vehicle r does not visit i
for r in R_id:
    for i in r_sc[r]:
        constr_demand_5 = model.addGenConstrIndicator (Z_R[i,r], False, Q_R[i,r], GRB.EQUAL, 0, name = 'Constr_5')

# 6. Demand satisfaction constraint
for s in S_id:
    s_r[s] = s_r[s] if isinstance(s_r[s], list) else [s_r[s]]
    for i in s_c[s]:
        constr_demand_6 = model.addConstr(quicksum(Q_R[i,r] for r in s_r[s]) == df_horeca_demand_scenarios.at[f'{i}', f'set_{horeca_set}'], name = 'Constr_6')

# 7. No load is delivered to satellites
for r in R_id:
    r_s[r] = r_s[r] if isinstance(r_s[r], list) else [r_s[r]]
    for i in r_s[r]:
        constr_demand_7 = model.addConstr (Q_R[i,r] == 0, name = 'Constr_7')

# 7b. The accumulated load is zero at satellites
for r in R_id:
    r_s[r] = r_s[r] if isinstance(r_s[r], list) else [r_s[r]]
    for i in r_s[r]:
        constr_demand_7b = model.addConstr (L_R[i,r] == 0, name = 'Constr_7b')

#8a. Maximum capacity of vehicle r indicator version:
for r in R_id:
    for i in r_sc[r]:
        for j in r_c[r]:
            constr_capacity_8a = model.addGenConstrIndicator(X_R[i,j,r], True, L_R[j,r] - L_R[i,r] - Q_R[j,r], GRB.EQUAL, 0, name = 'Constr8')

# 8b. No L_R if not visited
for r in R_id:
    for i in r_sc[r]:
        constr_capacity_8b = model.addGenConstrIndicator (Z_R[i,r], False, L_R[i,r], GRB.EQUAL, 0, name = 'Constr_8b')

# 8c. The load delivered to customer i by vehicle r is always less than or equal to the accumulated load of r at customer i:
for r in R_id:
    for i in r_c[r]:
        constr_capacity_8c = model.addConstr(Q_R[i,r] <= L_R[i,r], name = 'Constr_8c')

# 8d. The accumulated load of vehicle r at customer i is always less than or equal to the maximum capacity of vehicle r:
for r in R_id:
    for i in r_c[r]:
        constr_capacity_8d = model.addConstr( L_R[i,r] <= capacity_se, name = 'Constr_8d')

# 12. Distance travelled per vehicle r
for r in R_id:
    constr_distance_12 =  model.addConstr(D_R[r] == quicksum(dist.at[road_nodes_dict[i], road_nodes_dict[j]] * X_R[i,j,r] for i in r_sc[r] for j in r_sc[r]), name = 'Constr_12' )


for (i, j, r), value in X_R_init.items():
    X_R[i, j, r].start = value
```

```
    for (i, r), value in Q_R_init.items():
        Q_R[i,r].start = value

    # Start optimisation
    print("start optimizing")
    model.setParam( 'OutputFlag', True)     # silencing gurobi output or not
    model.setParam ('MIPGap', mip_VRP_E2);        # find the optimal solution with optimalitygap of 20%
    model.setParam('SoftMemLimit', 50)
    model.setParam('MIPFocus',0)
    model.setParam('Seed', 123)
    if time_limit:
        model.setParam('Timelimit', time_limit)
    model._obj = None
    model._bd = None
    model._obj_value = []
    model._time = []
    model._start = time.time()
    model.optimize()

    mip_gap_E2 = model.MIPGap
    end_time_VRP_road = time.time()
    time_VRP_E2 = end_time_VRP_road - start_time_VRP_road


    #%% Split long trips road vehicles
    print('Working on split trips VRP E2 for Ns:', Ns)
    # Create list of trips per vehicle
    r_v = {}
    s_v = {}
    s_x = 0
    Nv = np.arange(1,int(500/Ns))
    for r in R_id:
        v_Nv = []
        s_Nv = []
        s_x += 1
        for n in Nv:
            vehicle_Nv = [f'S{s_x}_V{n}']
            v_Nv = v_Nv + vehicle_Nv
        for s in r_s[r]:
            for m in Nv:
                vehicle_Nv_s = [f'S{s_x}_V{m}']
                s_Nv = s_Nv + vehicle_Nv_s
            s_v[s] = s_Nv
        r_v[r] = v_Nv

    # Create V_id, set of all trips
    V_id = []
    for r in R_id:
            for v in r_v[r]:
                V_id.append(v)

    # Split the trips found by VRP E2
    X = {}
    for r in R_id:
        for i in r_sc[r]:
            for j in r_sc[r]:
                    X[i,j,r] = X_R[i,j,r].X

    Y_V = {}
    for r in R_id:
            for v in r_v[r]:
                for i in r_sc[r]:
                    for j in r_sc[r]:
                        Y_V[i,j,v] = 0

    L_R_tot = {}
    for r in R_id:
            L_R_R = 0
            for i in r_sc[r]:
```

```
                    L_R_R += Q_R[i,r].X
            L_R_tot[r] = L_R_R

    for r in R_id:
        sum_X = 1
        L_R = 0
        for v in r_v[r]:
            L_V = 0
            for k in r_sc[r]:
                    sum_X += X[r_s[r][0],k,r]
            if sum_X > 0:
                sum_X = 0
                i = r_s[r][0]
                vehicle = 1
                while vehicle == 1:
                    for j in r_sc[r]:
                        if X[i,j,r] == 1:
                            L_V += Q_R[j,r].X
                            X[i,j,r] = 0
                            Y_V[i,j,v] = 1
                            i = j
                            if j == r_s[r][0]:
                                vehicle = 0
                                break

    # Create Z_V and D_V
    Z_V = {}
    D_V = {}

    for v in V_id:
        for i in DS_id:
            Z_V[i,v] = 0

    for r in R_id:
            for v in r_v[r]:
                distance_v = 0
                for i in r_sc[r]:
                    Z_V[i,v] = 0
                    for j in r_sc[r]:
                        if Y_V[i,j,v] == 1:
                            distance_v += dist.at[road_nodes_dict[i],road_nodes_dict[j]]
                            Z_V[i,v] = 1
                D_V[v] = distance_v



    # Create Q_V and L_V for VRP water
    Q_V = {}
    L_V = {}
    T_V = {}
    vessels = pd.read_excel(path +"Water_vehicles.xlsx", index_col=0)

    for v in V_id:
        for i in S_id:
            L_V[i,v] = 0

    W_id = vessels.index.tolist()
    for r in R_id:
            for w in W_id:
                L_V[r_s[r][0],w] = 0
            for v in r_v[r]:
                num_cust = 0
                Load = 0
                for i in r_c[r]:
                    if Z_V[i,v] == 1:
                        num_cust += 1
                        Q_V[i,v] = df_horeca_demand_scenarios.at[f'{i}', f'set_{horeca_set}']
                        Load += Q_V[i,v]
                L_V[r_s[r][0],v] = Load
                if num_cust > 0:
                    T_V[v] = transship_s + transship_c * num_cust + D_V[v] / speed_v
```

```
                if num_cust == 0:
                    T_V[v] = 0



    # Create LS_V[i]: total load picked up at satllite
    LS_V = {}
    for i in S_id:
        load = 0
        for v in V_id:
            load += L_V[i,v]
        LS_V[i] = load

    # Only select v with routes
    V_id_new = []
    for r in R_id:
        for v in r_v[r]:
            visits = 0
            for i in r_c[r]:
                if Z_V[i,v] == 1:
                    visits += 1
            if visits >= 1:
                V_id_new.append(v)
    V_id = V_id_new.copy()
    Nr_v_VRP_E2 = len(V_id_new)
    #%%
    # Total distance road:
    D_r = 0
    for r in R_id:
        if D_R[r].X >0:
            D_r += D_R[r].X

    # Create length trips needed for scheduling
    E_V = {}            # End customer of trip v
    for r in R_id:
        for v in r_v[r]:
            distance_v = 0
            for i in r_sc[r]:
                Z_V[i,v] = 0
                for j in r_sc[r]:
                    if Y_V[i,j,v] == 1:
                        distance_v += dist.at[road_nodes_dict[i],road_nodes_dict[j]]
                        Z_V[i,v] = 1
                    if Y_V[i,r_s[r][0],v] == 1:
                        E_V[v] = i
            D_V[v] = distance_v

    #%%
    v_s = {}
    for r in R_id:
        for v in r_v[r]:
            v_s[v] = r_s[r][0]
    #%%
    # Determine closest vehicle depot location for each trip
    depots = [103144, 101875, 101642, 102344,100243]
    v_d = {}
    for v in V_id:
        dist_depot = 99999
        for d in depots:
            dist_v_d = dist.at[d,road_nodes_dict[v_s[v]]]
            if dist_v_d < dist_depot:
                dist_depot = dist_v_d
                depot_v = d
        v_d[v] = depot_v



    # Distance to go from tril k to trip l D_E[k,l]
    D_E = {}            # Distance from end customer of trip l to satellite of k, minus the distance from end customer of l to satellite of l
    for l in V_id:
```

```
        D_E['zero',l] = dist.at[v_d[l],road_nodes_dict[v_s[l]]]
        D_E[l,'zero'] = 99999
        for k in V_id:
            D_E[l,k] = dist.at[road_nodes_dict[E_V[l]],road_nodes_dict[v_s[k]]] - dist.at[
    road_nodes_dict[E_V[l]],road_nodes_dict[v_s[l]]]
    D_E['zero', 'zero'] = 0

    # Total distance of trip l + distance to start of k - distance from last customer of l to
      satellite of l

    D_T = {}            # Distance from end customer of trip l to satellite of k, minus the
    distance from end customer of l to satellite of l
    for l in V_id:
        D_T['zero',l] = dist.at[v_d[l],road_nodes_dict[v_s[l]]]
        D_T[l,'zero'] = D_V[l] - dist.at[road_nodes_dict[E_V[l]],road_nodes_dict[v_s[l]]] +
    dist.at[road_nodes_dict[E_V[l]],v_d[l]]
        for k in V_id:
            D_T[l,k] = D_V[l] + dist.at[road_nodes_dict[E_V[l]],road_nodes_dict[v_s[k]]] -
    dist.at[road_nodes_dict[E_V[l]],road_nodes_dict[v_s[l]]]
    D_T['zero', 'zero'] = 0
```

B.3. First-Echelon Trip Generation

```
# Old file: Total_model_FLP_VRPs_MIP_times_parameters.py

#%% Import libraries
import gurobipy as gb
import time
import os
import numpy as np
import pandas as pd
import pickle
import copy
from gurobipy import quicksum, GRB
import warnings


    #%% Create initial solution VRP E1
    print('Creating initial solution VRP E1 for Ns:', Ns)
    # Heuristics routes vessels X_W, L_W, Q_W with stock new + B
    WV_id = W_id + V_id
    WV0_id = zero + WV_id

    L_deliver = {}
    X_W_init = {}
    Q_W_init = {}
    L_W_init = {}
    S_init = {}
    B_init = {}
    D_init = {}

    # Determine closest DC for each satellite
    DC_S = {}
    for s in S_id:
        dist_DC = 999999
        for dc in DC:
            dist_s_DC = dist_fe.at[canal_nodes_dict[dc], canal_nodes_dict[s]]
            if dist_s_DC < dist_DC:
                dist_DC = dist_s_DC
                dc_s = dc
        DC_S[s] = dc_s


    S_DC = {'DC_1': [], 'DC_2': [], 'DC_3': []}

    for s in S_id:
        dc = DC_S.get(s)
        if dc in S_DC:
            S_DC[dc].append(s)
    DC_used = []
    for d in DC:
```

```
        used = 0
        for s in S_id:
            if s in S_DC[d]:
                used = 1
        if used ==1:
            DC_used.append(d)
    DC = DC_used.copy()
    DS_id = DC + S_id



    #%% Initial solution VRP-E1 with multiple depots with neighbourhoods

    W_id = vessels.index.tolist()
    WV_id = W_id + V_id
    WV0_id = zero + WV_id
    for i in DS_id:
        for k in WV_id:
            L_W_init[i,k] = 0
        for l in WV0_id:
            Q_W_init[i,l] = 0
        for j in DS_id:
            for w in W_id:
                X_W_init[i,j,w] = 0
                L_deliver[j,w] = 0

    for i in S_id:
        for k in WV0_id:
            for l in WV0_id:
                B_init[i,k,l] = 0
                D_init[i,k,l] = 0

    S_left = S_id.copy()
    L_left = LS_V.copy()
    Nr_visits = np.arange(0,Ns+1)
    S_satisfied = 0
    V_left = V_id.copy()
    S_save = S_id.copy()

    dc_count = 0
    S_DC_satisfied = 0

    visited_wv = {}
    visited_wv_list = []
    for i in S_id:
        visited_wv[i] = []

    for w in W_id:
        if S_satisfied == len(S_id):
            break
        d = DC[dc_count]
        capacity_w = capacity_fe
        i = DS_id[0]
        L_W_ = 0
        S_left = S_save.copy()

        for n in Nr_visits:
            dist_old = 99999
            v_to_remove = []
            for s in S_left:

                if s not in S_DC[d]:

                    continue
                elif s in S_DC[d]:

                    if n == 0:
                        i = DC_S[s]

                    distance_ = dist_fe.at[canal_nodes_dict[i],canal_nodes_dict[s]]
                    if distance_ < dist_old:
```

```
                    dist_old = distance_
                    j = s
                    L_left_ = L_left[j]
                    k = i
                    if n == 0:
                        i = DC_S[s]
                        depot = DC_S[s]
        i = k
        L_request = 0
        visited_v_list = []
        visited_v_list = visited_wv[j].copy()
        for wv in visited_v_list:
            B_init[j,w,wv] = 1
            D_init[j,w,wv] = 1          # New

        for v in V_left:
            if L_V[j,v] > 0 :
                if L_request < capacity_w:
                    L_request += L_V[j,v]
                    if L_request <= capacity_w:
                        v_to_remove.append(v)
                       # V_left.remove(v)
                        L_deliver[j,w] = L_request
                        visited_wv_list = []
                        visited_wv_list = visited_wv[j].copy()
                        for wv in visited_v_list:
                            D_init[j,v,wv] = 1                 # New

                        visited_wv_list.append(w)
                        B_init[j,v,'zero'] = 1
                        for wv in visited_wv_list:
                            B_init[j,v,wv] = 1
                        visited_wv_list.append(v)
                        visited_wv[j] = visited_wv_list
                        visited_v_list.append(v)
                    elif L_request > capacity_w:
                        L_request -= L_V[j,v]
                        continue
        for wv in visited_v_list:
            D_init[j,w,wv] = 1
        for v in v_to_remove:
            V_left.remove(v)
        Q_W_init[j,w] = L_deliver[j,w]
        capacity_w -= Q_W_init[j,w]
        L_W_ += Q_W_init[j,w]

        L_left_ -= Q_W_init[j,w]
        L_left[j] = L_left_
        if L_left[j] == 0:
            S_save.remove(j)
            S_satisfied += 1
            S_DC_satisfied += 1

            if S_satisfied == len(S_id):
                if Q_W_init[j,w] > 0:
                    L_W_init[j,w] = L_W_
                    X_W_init[i,j,w] = 1
                    X_W_init[j,depot,w] = 1
                    B_init[j,w,'zero'] = 1
                    break
        if Q_W_init[j,w] > 0:
            X_W_init[i,j,w] = 1
            B_init[j,w,'zero'] = 1
            L_W_init[j,w] = L_W_
            i = j
            S_left.remove(j)
            print('removed: ', j)
            print('S_DC_satisfied: ', S_DC_satisfied)
            if S_DC_satisfied == len(S_DC[d]):
                    dc_count += 1
                    print('DC satisfied count')
```

```
                        S_DC_satisfied = 0
                        X_W_init[j,depot,w] = 1
                        break
                if len(S_left) == 0:
                    X_W_init[j,depot,w] = 1
                    break
                found = False
                for k in S_left:
                    if k in S_DC[d]:
                        found = True
                if not found:
                    X_W_init[j,depot,w] = 1
                    break
            if Q_W_init[j,w] == 0:
                X_W_init[i,depot,w] = 1
                if S_DC_satisfied == len(S_DC[d]):
                        dc_count += 1
                        S_DC_satisfied = 0
                break

    #
    # Initial solution Z_WV
    Z_WV_init = {}
    for i in DS_id:
        for w in WV_id:

            Z_WV_init[i,w] = 0

    for w in W_id:
        for i in DS_id:
            for j in DS_id:
                if X_W_init[i,j,w] == 1:
                    Z_WV_init[i,w] = 1
                    Z_WV_init[j,w] = 1
                    print(w,i,j)
                    D_init[i,w,'zero'] = 1

    for v in V_id:
        for i in S_id:
            if Z_V[i,v] == 1:
                Z_WV_init[i,v] = 1
                D_init[i,v,'zero'] = 1
    #%%
    if S_DC_satisfied == len(S_DC[d]):
        print('yes')
        DC_W = {}

    for w in W_id:
        for d in DC:
            for i in S_id:
                if X_W_init[i,d,w] == 1:
                    DC_W[w] = d
#%%

    # Initial solution Y
    Y_init = {}
    for i in DS_id:
        for k in WV_id:
            for l in WV_id:
                Y_init[i,k,l] = 0
                if k != l:
                    if Z_WV_init[i,k] == 1:
                        if Z_WV_init[i,l] == 1:
                            Y_init[i,k,l] = 1

    # Make sure B_init[i,k,l] is zero if not both vehicles visit i
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                if Y_init[i,k,l] == 0:
                    if  B_init[i,k,l] > 0:
```

```
                    print('fixed error ', i,k,l)
                if  B_init[i,l,k] > 0:
                    print('fixed error ', i,l,k)
                B_init[i,k,l] = 0
                B_init[i,l,k] = 0

# Initial solution D_w
D_w_init = 0
D_w_s = {}
for w in W_id:
    for i in DS_id:
        for j in DS_id:
            if X_W_init[i,j,w] == 1:
                D_w_init += dist_fe.at[canal_nodes_dict[i],canal_nodes_dict[j]]
                D_w_s[j,w] = dist_fe.at[canal_nodes_dict[i],canal_nodes_dict[j]]

print('Distance on the waterways for initial solution: ', D_w_init)

# Check B_init[i,k,l]
for i in S_id:
    for k in WV_id:
        for l in WV_id:
            if Y_init[i,k,l] == 1:
                bb = B_init[i,k,l] + B_init[i,l,k]
                if bb < 1:
                    print(B_init[i,k,l], B_init[i,l,k], ' error for ', i,k,l)
            b = B_init[i,k,l] + B_init[i,l,k]
            if b > 1:
                print('error for ', i,k,l)


# Check if all demand is delivered
Delivered = 0
for k in W_id:
    for i in S_id:
        if Z_WV_init[i,k] >= 1:
            Delivered += Q_W_init[i,k]
print(Delivered)


# Only select w that are used
W_used = []
for w in W_id:
    w_visits = 0
    for i in S_id:
        if Z_WV_init[i,w] == 1:
                w_visits += 1
    if w_visits >= 1:
        W_used.append(w)

Nr_w_init = len(W_used)
W_id = W_used.copy()
WV_id = W_id + V_id
WV0_id = zero + WV_id
W0_id = zero + W_id
V0_id = zero + V_id

for i in S_id:
    for k in V0_id:
        for l in V0_id:
            if B_init[i,k,l] == 1:
                D_init[i,k,l] = 1
    for k in W0_id:
        for l in W0_id:
            if B_init[i,k,l] == 1:
                D_init[i,k,l] = 1


# New initial solutions only for w in W_used
Z_WV_init_used = {}
for w in WV_id:
```

```
        for i in DS_id:
            Z_WV_init_used[i,w] = 0

    for w in W_id:
        for i in DS_id:
            for j in DS_id:
                if X_W_init[i,j,w] == 1:
                    Z_WV_init_used[i,w] = 1
                    Z_WV_init_used[j,w] = 1

    for v in V_id:
        for i in S_id:
            if Z_V[i,v] == 1:
                Z_WV_init_used[i,v] = 1

    Y_init_used = {}
    for i in DS_id:
        for k in WV_id:
            for l in WV_id:
                Y_init_used[i,k,l] = 0
                if k != l:
                    if Z_WV_init[i,k] == 1:
                        if Z_WV_init[i,l] == 1:
                            Y_init_used[i,k,l] = 1


    X_W_init_used = {}
    Q_W_init_used = {}
    L_W_init_used = {}
    B_init_used = {}
    D_init_used = {}

    for i in DS_id:
        for k in WV_id:
            L_W_init_used[i,k] = L_W_init[i,k]
        for l in WV0_id:
            Q_W_init_used[i,l] = Q_W_init[i,l]
        for j in DS_id:
            for w in W_id:
                X_W_init_used[i,j,w] = X_W_init[i,j,w]
                L_deliver[j,w] = 0

    for i in S_id:
        for k in WV0_id:
            for l in WV0_id:
                B_init_used[i,k,l] = B_init[i,k,l]
                D_init_used[i,k,l] = D_init[i,k,l]
    print('Number of road vehicle trips: ', len(V_id))
    # Check values for Z_WV_init
    for w in W_id:
        for i in DS_id:
            for j in DS_id:
                if X_W_init[i,j,w] == 1:
                    if Z_WV_init[i,w] != 1:
                        print('error ', w, i)
                    if Z_WV_init[j,w] != 1:
                        print('error ', w, j)

    #%%
    model.dispose()

    #%% VRP E1
    print('Working on VRP E1 for Ns:', Ns)
    start_VRP_E1 = time.time()
    model = gb.Model('VRP_E1')
    np.random.seed(123)
    MIPGap = 0.25
    time_limit = t_lim_VRP_E1
    K = 9999
    print("Waiting time vessels added in objective")
    # Variables
```

```
    # New
    # Path from i to j, if used by vessel w: = 1, else: = 0
    X_W = {}
    for w in W_id:
        for i in DS_id:
            for j in DS_id:
                X_W[i,j,w] = model.addVar(vtype = GRB.BINARY, name = 'X_W')

    # Binary variable, Y[i,k,l] = 1 if both k and l visit i
    Y = {}
    for k in WV_id:
        for l in WV_id:
            for i in DS_id:
                Y[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'Y')


    # Arrival time of vehicle r at i
    A_WV = {}
    for i in DS_id:
        for k in WV_id:
            A_WV[i,k] = model.addVar(lb = -500, ub = 999999, vtype = GRB.CONTINUOUS, name = 'A_WV')

    # Difference in arrival times of vehicle
    A_D = {}
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                A_D[i,k,l] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'A_D')

    # Difference in arrival times of vehicle
    A_DD = {}
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                A_DD[i,k,l] = model.addVar(lb = -999999, ub = 999999, vtype = GRB.CONTINUOUS, name = 'A_DD')


    # New
    # Quantity delivered to customer i or picked up at satellite i by vehicle r
    Q_W = {}
    for w in WV0_id:
        for i in DS_id:
            Q_W[i,w] = model.addVar(lb = 0.0,  vtype = GRB.INTEGER, name = 'Q_W')


    # New
    # Customer or satellite is visited by vehicle r: = 1, if not: = 0
    Z_W = {}
    for w in W_id:
        for i in DS_id:
            Z_W[i,w] =  model.addVar(vtype = GRB.INTEGER, name = 'Z_W')

    # Customer or satellite is visited by vehicle r: = 1, if not: = 0
    Z_WV = {}
    for k in WV_id:
        for i in DS_id:
            Z_WV[i,k] =  model.addVar(vtype = GRB.BINARY, name = 'Z_WV')


    # New
    # Accumulated load of road vehicle r at customer i
    L_W = {}
    for w in WV_id:
        for i in DS_id:
            L_W[i,w] =  model.addVar(lb=0.0, vtype = GRB.INTEGER, name = 'L_W')


    # Accumulated load delivered to satellite i by vehicles before and including k
    LS = {}
```

```
    for i in S_id:
        for k in WV0_id:
            LS[i,k] = model.addVar(lb = 0.0, vtype = GRB.INTEGER, name = 'LS')


    # total distance travelled over water
    D_w = model.addVar(vtype = GRB.INTEGER, name = 'D_w')

    # Number of water vehicles used
    Nw = {}
    for w in W_id:
        Nw[w] = model.addVar(vtype = GRB.BINARY, name = 'Nw')


    # Stock at satellite i after arrival of vehicle k
    S = {}
    for i in S_id:
        for k in WV_id:
            S[i,k] = model.addVar(lb = -100, vtype = GRB.INTEGER, name = 'S')

    B = {}
    for i in S_id:
        for k in WV0_id:
            for l in WV0_id:
                B[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'B')


    # New for departure times
    D = {}
    for i in S_id:
        for k in WV0_id:
            for l in WV0_id:
                D[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'D')


    D_WV = {}
    for i in DS_id:
        for k in WV_id:
            D_WV[i,k] = model.addVar(lb = 0.0, ub = 999999, vtype = GRB.CONTINUOUS, name = '
    D_WV')

    W = {}
    for i in DS_id:
        for w in W0_id:
            W[i,w] = model.addVar(vtype = GRB.CONTINUOUS, name = 'W')


    #  Objective function
    model.setObjective(quicksum(dist_fe.at[canal_nodes_dict[i],canal_nodes_dict[j]] * X_W[i,j
    ,w] for w in W_id for i in DS_id for j in DS_id) + 100 * quicksum(Nw[w] for w in W_id) +
    quicksum(W[i,w] for i in S_id for w in W_id))

    model.modelSense = GRB.MINIMIZE
    model.update()

    # Constraints
    # 1. A vehicle never goes from i to i
    for w in W_id:
        for i in DS_id:
            for j in DS_id:
                if i == j:
                    constr_w_1 = model.addConstr(X_W[i,j,w] == 0, name='Constr_1')

    # 2. Vehicle r can only leave node if it also arrived there
    for w in W_id:
        for i in DS_id:
            # if i != j:
                constr_w_2 = model.addConstr(quicksum(X_W[i,j,w] for j in DS_id) == quicksum(
    X_W[j,i,w] for j in DS_id), name='Constr_2')

    # 2b. New for neighbourhoods, X_W[i,j,w]=0 if i,j not assigned to same depot as w
```

```python
    for w in W_id:
        for d in DC:
            if d != DC_W[w]:
                for i in S_id:
                    for j in S_DC[d]:
                        constr_w_2b = model.addConstr(X_W[i,j,w] == 0, name='Constr_2b')

    # 3a. New for neighbourhoods, Z_WV[i,w] = 0 if  not in DC_W
    for w in W_id:
        for d in DC:
            if d != DC_W[w]:
                for s in S_DC[d]:
                    constr_w_3a = model.addConstr(Z_WV[s,w] == 0, name='Constr_3a')

    # 3. Nodes that are visited by vehicle w
    for w in W_id:
        for i in DS_id:
            constr_w_3b = model.addConstr(Z_WV[i,w] == quicksum(X_W[i,j,w] for j in DS_id), name='Constr_3')

    # 4b. Nodes that are visited by vehicle r
    for v in V_id:
        for i in DS_id:
            constr_w_4c = model.addConstr(Z_WV[i,v] == Z_V[i,v], name='Constr_4')

    # New
    # 5. The demand delivered to i is zero if vehicle r does not visit i
    for w in W_id:
        for i in DS_id:
            constr_w_5 = model.addGenConstrIndicator (Z_WV[i,w], False, Q_W[i,w], GRB.EQUAL, 0, name='Constr_5')

    # 6. Demand satisfaction constraint
    for i in S_id:
            constr_w_6 = model.addConstr(quicksum(Q_W[i,w] for w in W_id) == LS_V[i], name='Constr_6') #s_v[i])) #
            constr_w_6b = model.addConstr(Q_W[i,'zero'] == 0, name='Constr_6b')

    # New
    # 7. No load is delivered to DC
    # 8. The accumulated load at the DC is zero
    for w in W_id:
        DC = DC if isinstance(DC, list) else [DC]
        for i in DC:
            constr_w_7 = model.addConstr (Q_W[i,w] == 0, name='Constr_7')
            constr_w_8 = model.addConstr(L_W[i,w] == 0, name='Constr_8')

    # 8b. No load delivered by road vehicles
    for v in V_id:
        for i in DS_id:
            constr_w_8b = model.addConstr(Q_W[i,v] == 0, name='Constr_8b')
            constr_w_8c = model.addConstr(L_W[i,v] == 0, name='Constr_8c')

    # 9a. Maximum capacity of vehicle r indicator version:
    for w in W_id:
        for i in DS_id:
            for j in S_id:
                constr_w_9a = model.addGenConstrIndicator(X_W[i,j,w], True, L_W[j,w] - L_W[i,w] - Q_W[j,w], GRB.EQUAL, 0, name='Constr_9a')

    # New
    # 9b. No L_R if not visited
    for w in W_id:
        for i in DS_id:
            constr_w_9b = model.addGenConstrIndicator (Z_WV[i,w], False, L_W[i,w], GRB.EQUAL, 0, name='Constr_9b')

    # New
    # 9c. The load delivered to customer i by vehicle r is always less than or equal to the accumulated load of r at customer i:
    for w in W_id:
```

```
        for i in S_id:
            constr_w_9c = model.addConstr(Q_W[i,w] <= L_W[i,w], name='Constr_9c')

    # New
    # 9d. The accumulated load of vehicle r at customer i is always less than or equal to the maximum capacity of vehicle r:
    for w in W_id:
        for i in S_id:
            constr_w_9d = model.addConstr( L_W[i,w] <= capacity_fe, name='Constr_9d')


    # # Arrival time constraints:
    for k in WV_id:
        for i in S_id:
            constr_time_span = model.addConstr(A_WV[i,k] >= 0, name = 'constr_time_span')


    # 10. Sequential visits to satellites by vessels
    for w in W_id:
        for i in DS_id:
            for j in S_id:
                constr_time_10a = model.addGenConstrIndicator(X_W[i,j,w], True, A_WV[j,w] - A_WV[i,w] - dist_fe.at[canal_nodes_dict[i],canal_nodes_dict[j]] / (speed_fe * 60) - W[i,w] - Q_W[i,w] * 0.2, GRB.GREATER_EQUAL, 0, name='Constr_10')

    # 10b. The arrival time at the first satellite of trip w is the arrival time at the depot - travel time - service time
    for w in W_id:
        for j in S_id:
            DC = DC if isinstance(DC, list) else [DC]
            for i in DC:
                constr_time_10b = model.addGenConstrIndicator(X_W[i,j,w], True, A_WV[j,w] - A_WV[i,w] - dist_fe.at[canal_nodes_dict[i],canal_nodes_dict[j]] / (speed_fe * 60) - service_time_fe / 60, GRB.EQUAL, 0, name = 'Constr_10b')


    # 11. Binary variable Y[i,k,l] is one if both k and l visit i
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                if k != l:
                    constr_Y_11 = model.addConstr(Y[i,k,l] == gb.and_(Z_WV[i,k], Z_WV[i,l]), name='Constr_11')


    # 12. Arrival times of vehicles at satellites cannot be the same
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                constr_time_12a = model.addConstr(A_DD[i,k,l] == A_WV[i,k] - A_WV[i,l], name='Constr_12a')
                constr_time_12b = model.addConstr(A_D[i,k,l] == gb.abs_(A_DD[i,k,l]), name='Constr_12b')


    # 13a. Arrival times of road vehicles at satellites cannot be the same
    for i in S_id:
        for k in V_id:
            for l in V_id:
                if k != l:
                    constr_time_13a = model.addGenConstrIndicator(Y[i,k,l], True, A_D[i,k,l], GRB.GREATER_EQUAL, 3, name='Constr_13a') #180)
                    constr_time_13a_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13a_1')


    # 13b. Arrival times of a water vehicles is later than the departure time of another water vehicle
    for i in S_id:
        for k in W_id:
            for l in W_id:
```

```
            if k != l:
                constr_time_13b = model.addGenConstrIndicator(B[i,k,l], True, A_WV[i,k] - D_WV[i,l] + 0.0001, GRB.GREATER_EQUAL, 0, name='Constr_13b') #600)
                constr_time_13b_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13b_1')
# 13b. Arrival times of water and road vehicles at satellites cannot be the same
for i in S_id:
    for k in W_id:
        for l in V_id:
            if k != l:
                constr_time_13c = model.addGenConstrIndicator(Y[i,k,l], True, A_D[i,k,l], GRB.GREATER_EQUAL, 0.0101, name='Constr_13c') #600)
                constr_time_13c_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13c_1')

#13c. Arrival times at satellites cannot be later than the maximum time span
for i in S_id:
    for k in WV_id:
        constr_time_13d = model.addConstr(D_WV[i,k] <= time_span - 1, name='Constr_13d')


# 14. Arrival time is infinite if a vehicle does not visit satellite i
for i in S_id:
    for k in WV_id:
        constr_time_14 = model.addGenConstrIndicator(Z_WV[i,k], False, A_WV[i,k], GRB.EQUAL, 0)

# # Satellite synchronisation constraints:

# 15. Binary variable = 1 if vehicle k arrives at the same time or after vehicle l
for i in S_id:
    for k in WV_id:
        constr_binary_150 = model.addGenConstrIndicator(Z_WV[i,k], True, B[i,k,'zero'], GRB.EQUAL, 1 )
        for l in WV_id:
            constr_binary_15a = model.addGenConstrIndicator(Y[i,k,l], True, A_WV[i,k] - K * B[i,k,l] - A_WV[i,l], GRB.LESS_EQUAL, 0 )
            constr_binary_15b = model.addGenConstrIndicator(Y[i,k,l], True, B[i,k,l] + B[i,l,k], GRB.EQUAL, 1)
            constr_binary_15c = model.addConstr(B[i,k,l] + B[i,l,k] <= 1)
            constr_binary_15d = model.addGenConstrIndicator(Z_WV[i,k], False, B[i,k,l], GRB.EQUAL, 0 )
            constr_binary_15e = model.addGenConstrIndicator(Z_WV[i,k], False, B[i,l,k], GRB.EQUAL, 0 )

# 16. Load delivered to satellite i by all vehicles before k and k
for i in S_id:
    for k in WV0_id:
        for l in WV0_id:
                constr_load_16a = model.addGenConstrIndicator(B[i,k,l], True, LS[i,k] - LS[i,l] - Q_W[i,k], GRB.GREATER_EQUAL,0)

for i in S_id:
    for k in WV_id:
        constr_load_16b = model.addConstr(LS[i,k] <= quicksum(Q_W[i,w] for w in W_id))
        constr_load_16c = model.addGenConstrIndicator(Z_WV[i,k], False, LS[i,k], GRB.EQUAL, 0 )

# 17. New Stock at satellites constraints
for i in S_id:
    for k in WV_id:
            constr_stock_17a = model.addGenConstrIndicator (Z_WV[i,k], True, S[i,k] + quicksum(L_V[i,l] * B[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.EQUAL, 0)

for i in S_id:
    for k in WV_id:
        constr_stock_17b = model.addConstr(S[i,k] >= 0)
        constr_stock_17c = model.addConstr(S[i,k] <= capacity_s[i] + capacity_fe)

constr_water_km = model.addConstr(D_w == quicksum(dist_fe.at[canal_nodes_dict[i], canal_nodes_dict[j]] * X_W[i,j,w] for w in W_id for i in DS_id for j in DS_id))
```

```
    for w in W_id:
        for i in S_id:
            constr_Nw = model.addGenConstrIndicator(Z_WV[i,w], True, Nw[w], GRB.EQUAL, 1)

    # New for departure times
    for i in S_id:
        for v in V_id:
            constr_departure_1 = model.addGenConstrIndicator(Z_WV[i,v], True, D_WV[i,v] - A_WV[i,v] - transship_s / 60, GRB.EQUAL, 0, name = 'constr_dep_1')
        for w in W_id:
            constr_departure_2 = model.addGenConstrIndicator(Z_WV[i,w], True, D_WV[i,w] - A_WV[i,w] - W[i,w] - Q_W[i,w] * 0.2, GRB.EQUAL, 0, name = 'constr_dep_2' )


    for i in S_id:
        for k in V0_id:
            for l in V0_id:
                constr_departure_3 = model.addGenConstrIndicator(B[i,k,l], True, D[i,k,l], GRB.EQUAL, 1, name = 'constr_dep_3')
        for k in W0_id:
            for l in W0_id:
                constr_departure_4 = model.addGenConstrIndicator(B[i,k,l], True, D[i,k,l], GRB.EQUAL, 1, name = 'constr_dep_4')

        for k in WV_id:
            constr_departure_8 =  model.addGenConstrIndicator (Z_WV[i,k], False, quicksum(D[i,k,l] for l in WV_id) + quicksum(D[i,l,k] for l in WV_id), GRB.EQUAL, 0, name = 'constr_dep_8')
            for l in WV_id:
                constr_departure_5 = model.addGenConstrIndicator(Y[i,k,l], True, D_WV[i,k] - K* D[i,k,l] - D_WV[i,l] + 0.0001 , GRB.LESS_EQUAL, 0, name = 'constr_dep_5')
                constr_departure_6 = model.addGenConstrIndicator(Y[i,k,l], True, D[i,k,l] + D[i,l,k], GRB.EQUAL, 1, name = 'constr_dep_6')


    for i in S_id:
        for k in W_id:
            constr_departure_7 = model.addGenConstrIndicator (Z_WV[i,k], True, quicksum(L_V[i,l] * D[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.LESS_EQUAL, 0, name = 'constr_dep_7')
            constr_departure_7_b = model.addGenConstrIndicator (Z_WV[i,k], True, quicksum(L_V[i,l] * D[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.GREATER_EQUAL, - capacity_s[i], name = 'constr_dep_7_b')



    for (i, j, w), value in X_W_init_used.items():
        X_W[i, j, w].start = value

    for (i, w), value in Q_W_init_used.items():
        Q_W[i,w].start = value

    for (i,w), value in L_W_init_used.items():
        L_W[i,w].start = value

    for (i,w), value in Z_WV_init_used.items():
        Z_WV[i,w].start = value

    for (i,k,l), value in Y_init_used.items():
        Y[i,k,l].start = value

    for (i,k,l), value in B_init_used.items():
        B[i,k,l].start = value


    # Start optimisation

    print("start optimizing")
    model.setParam( 'OutputFlag', True)
```

```
    model.setParam ('MIPGap', mip_VRP_E1);
    model.setParam('FeasibilityTol', 1e-6)
    model.setParam('MIPFocus', 0)
    model.setParam('SubMIPNodes', 20000)
    model.setParam('Seed', 123)
    model.setParam('SoftMemLimit', 70)
    if time_limit:
        model.setParam('Timelimit', time_limit)
    model._obj = None
    model._bd = None
    model._obj_value = []
    model._time = []
    model._start = time.time()
    model.optimize()
    mip_gap_vrp_E1 = model.MIPGap

    end_VRP_E1 = time.time()
    time_VRP_E1 = end_VRP_E1 - start_VRP_E1

    #%% Save results VRP E1
    X_W_init_s = model.getAttr('X', X_W)
    Y_init_s = model.getAttr('X', Y)
    A_WV_init_s = model.getAttr('X', A_WV)
    A_D_init_s = model.getAttr('X', A_D)
    A_DD_init_s = model.getAttr('X', A_DD)
    Q_W_init_s = model.getAttr('X', Q_W)
    Z_WV_init_s = model.getAttr('X', Z_WV)
    L_W_init_s = model.getAttr('X', L_W)
    LS_init_s = model.getAttr('X', LS)
    S_init_s = model.getAttr('X', S)
    B_init_s = model.getAttr('X', B)
    D_init_s = model.getAttr('X', D)
    D_WV_init_s = model.getAttr('X', D_WV)
    W_init_s = model.getAttr('X', W)

    D_w_VRP_E1 = D_w.X
    print('Distance on waterways after VRP_E1: ', D_w_VRP_E1)
    W_used_VRP_E1 = []
    for w in W_id:
        w_visits = 0
        for i in S_id:
            if Z_WV_init_s[i,w] == 1:
                    w_visits += 1
        if w_visits >= 1:
            W_used_VRP_E1.append(w)
    Nr_w_VRP_E1 = len(W_used_VRP_E1)
    W_id = W_used_VRP_E1.copy()

    # Calculate time it takes to perform trips for road vehicles
    Nc_V = {}           #Number of customers visited in trip v Nc_V
    for r in R_id:
        for v in r_v[r]:
            Nc_V[v] = quicksum(Z_V[i,v] for i in r_c[r])
    Nc_V['zero'] = 0


    P_V = {}
    for l in V_id:
        P_V['zero',l] = D_T['zero',l]/(speed_v * 60) + (transship_c / 60) * Nc_V['zero'] +
    transship_s / 60
        for k in V0_id:
            P_V[l,k] = D_T[l,k]/(speed_v * 60) + (transship_c / 60) * Nc_V[l] + transship_s /
     60
            P_V[l,k] = P_V[l,k].getValue()
    P_V['zero', 'zero'] = 0

    #%%
    for w in W_id:
        for i in DS_id:
            if Z_WV_init_s[i,w] == 1:
                print(w,i,A_WV[i,w].X)
```

```
    #%%
    DC = DC if isinstance(DC, list) else [DC]
    for i in DC:
        print(i)
    print(D_r, mip_gap_E2, D_w_VRP_E1, mip_gap_vrp_E1)
    #%%
    DC_W = {}
    for w in W_id:
        for d in DC:
            for i in S_id:
                if X_W_init_s[i,d,w] == 1:
                    DC_W[w] = d
    #%%
    with open(f'output_VRPs_{save_title}_{Ns}_{t_lim_VRP_E2}.txt', 'w') as f:
        f.write(f'D_r_VRP_E2:\n{D_r}\n')
        f.write(f'MIP_VRP_E2:\n{mip_gap_E2}\n')
        f.write(f'D_w_VRP_E1:\n{D_w_VRP_E1}\n')
        f.write(f'MIP_VRP_E1:\n{mip_gap_vrp_E1}\n')
        for var_name, var_values in [
            ('X_W', X_W_init_s),
            ('Y', Y_init_s),
            ('A_WV', A_WV_init_s),
            ('A_D', A_D_init_s),
            ('A_DD',A_DD_init_s),
            ('Q_W', Q_W_init_s),
            ('Z_WV', Z_WV_init_s),
            ('L_W', L_W_init_s),
            ('LS', LS_init_s),
            ('S', S_init_s),
            ('B', B_init_s),
            ('D', D_init_s),
            ('D_WV', D_WV_init_s),
            ('W', W_init_s),
            ('P_V', P_V),
            ('L_V', L_V),
            ('LS_V', LS_V),
            ('Z_V', Z_V),
            ('v_s', v_s),
            ('L_V', L_V),
            ('D_T', D_T),
            ('v_d', v_d),
            ('canal_nodes_dict', canal_nodes_dict)
        ]:
            f.write(f'{var_name}:\n')
            for key, value in var_values.items():
                if isinstance(value, gb.LinExpr):
                    value = value.getValue()
                f.write(f'  {key}: {value}\n')
        f.write('V_id:\n')
        for v in V_id:
            f.write(f'{v}\n')
        f.write('W_id:\n')
        for w in W_id:
            f.write(f'{w}\n')
        f.write('S_id:\n')
        for s in S_id:
            f.write(f'{s}\n')
        f.write('DC:\n')
        for d in DC:
            f.write(f'{d}\n')
    with open(f'output_VRPs_plot_{save_title}_{Ns}_{t_lim_VRP_E2}.txt', 'w') as f:
        for var_name, var_values in [
            ('Y_V', Y_V),
            ('s_c', s_c),
            ('v_d', v_d),
            ('road_nodes_dict', road_nodes_dict)
            ]:
            f.write(f'{var_name}:\n')
            for key, value in var_values.items():
                if isinstance(value, gb.LinExpr):
```

```
                value = value.getValue()
            f.write(f'  {key}: {value}\n')


    model.dispose()
```

B.4. Scheduling Problem

B.4.1. Road Vehicle Scheduling

```
#%% Import libraries
import gurobipy as gb
import time
import os
import numpy as np
import pandas as pd
import pickle
import math
import copy
import sys
import matplotlib.pyplot as plt
from openpyxl import load_workbook
from gurobipy import quicksum, GRB

#%% Set path
server = 'True'

if server == 'False':
    path = os.getcwd() + "\Inputs\\"
    path_out = os.getcwd() + "\Outputs\\"
    from FLP_solver_definition_number_customers_horeca_sets_Laudy import FLP_num_cust
    from FLP_solver_definition_horeca_sets_capacity_assignment import FLP_capacity

if server == 'True':
    path = os.getcwd() + "/Inputs/"
    path_out = os.getcwd() + "/Outputs/"


#%% Scenario inputs
directed = 'true'                # Indicate wether to use directed or undirected distance
     matrix
FLP_constraint = 'num_cust'      # Which FLP constraint to use, either capacity or num_cust
Nc = 750                         # Insert the number of customers to consider
horeca_sets = np.arange(1,11)    # Which horeca sets to evaluate
horeca_set = 1                   # If not testing all horeca sets, insert one to evaluate



#%% Import network and scenario data
df_horeca_demand_scenarios = pd.read_excel(path + f'df_horeca_demand_scenarios.xlsx',
     index_col=0)
df_horeca_demand_scenarios.index = df_horeca_demand_scenarios.index.astype(str)
df_horeca_data_info = pd.read_excel(path + f'df_horeca_data_info.xlsx', index_col=0)
df_horeca_data_info.index = df_horeca_data_info.index.astype(str)
customer_locations = df_horeca_data_info.iloc[:,0]

if server == 'False':
    df_SE_shortest_dist_directed_False = pickle.load(open(path + '
     df_SE_shortest_dist_directed-False_nodes_all.pickle', 'rb'))
    df_SE_shortest_dist_directed_True_1 = pickle.load(open(path + '
     df_SE_shortest_dist_directed-True_nodes_all.pickle', 'rb'))
    df_SE_shortest_dist_directed_True = df_SE_shortest_dist_directed_True_1.fillna(1001)
    dict_FE_shortest_dist_directed_True_1 = pickle.load(open(path + '
     dict_FE_shortest_dist_directed-True_nodes_all.pickle', 'rb'))

if server == 'True':
    pickle_off = open(path + 'df_SE_shortest_dist_directed-True_nodes_all.pickle', 'rb')
    df_SE_shortest_dist_directed_True_1 = pd.read_pickle(pickle_off)
    df_SE_shortest_dist_directed_True = df_SE_shortest_dist_directed_True_1.fillna(1001)

    pickle_off = open(path + 'dict_FE_shortest_dist_directed-True_nodes_all.pickle', 'rb')
    dict_FE_shortest_dist_directed_True_1 = pd.read_pickle(pickle_off)
```

```
    df_SE_shortest_dist_directed_False = dict_FE_shortest_dist_directed_True_1
assigned = []
indices = []
customers = [[0]*3]*len(customer_locations)
for customer_id in df_horeca_data_info.index.tolist():
    if df_horeca_demand_scenarios.at[f'{customer_id}', f'set_{horeca_set}'] > 0:
        indices.append(customer_id)
        assigned.append({'road_node':int(df_horeca_data_info.at[customer_id, 'road_node']), 'demand':int(df_horeca_demand_scenarios.at[f'{customer_id}', f'set_{horeca_set}'])} )
customers = pd.DataFrame(assigned, index= indices)# df_horeca_data_info.index.tolist() )


satellite_locations = pd.read_excel(path + "satellite_nodes_storage_full.xlsx", index_col=0)
vehicles = pd.read_excel(path  +"Road_vehicles.xlsx", index_col=0)
road_nodes = pd.read_excel(path + "satellites_customers_road_nodes.xlsx", index_col = 0)

if directed == 'true':
    dist = df_SE_shortest_dist_directed_True
elif directed == 'false':
    dist = df_SE_shortest_dist_directed_False

#%% Parameters
speed_v = int(os.getenv('speed_v'))
tranship_s = int(os.getenv('tranship_s'))
tranship_c = int(os.getenv('tranship_c'))
fev_profile = 5
capacity_fe = int(os.getenv('capacity_fe'))
speed_fe_str =  os.getenv('speed_fe')
speed_fe = float(speed_fe_str)
service_time_fe = int(os.getenv('service_time_fe'))
capacity_s = int(os.getenv('capacity_s'))
capacity_se = int(os.getenv('capacity_se'))
Ns = int(os.getenv('NrSatellites'))
df_fe_distance_matrix = dict_FE_shortest_dist_directed_True_1[f'vessel_profile_{fev_profile}'
    ].copy()
dist_fe = df_fe_distance_matrix.fillna(99999)

# New distance matrix for multiple water vehicle depots:
dict_FE_new = pd.read_csv(path + 'distance_matrix_DCs.csv',sep=';',header=None)
dist_fe_new = pd.DataFrame(dict_FE_new)
dist_fe_new.index = dist_fe_new.index + 1
new_index = {old_index:old_index + 1 for old_index in dist_fe_new.columns}
dist_fe_new = dist_fe_new.rename(columns=new_index)
dist_fe = dist_fe_new.fillna(99999)

t_limits_VRP_E2_str = os.getenv('t_limits_VRP_E2')
t_limits_VRP_E2 = eval(t_limits_VRP_E2_str)
t_lim_VRP_E1 = int(os.getenv('t_lim_VRP_E1'))
t_lim_sched_road = int(os.getenv('t_lim_sched_road'))
t_lim_sched_water = int(os.getenv('t_lim_sched_water'))
t_lim_sched_total = int(os.getenv('t_lim_sched_total'))
time_span = int(os.getenv('time_span'))
mip_VRP_E2_str = os.getenv('mip_VRP_E2')
mip_VRP_E2 = float(mip_VRP_E2_str)
mip_VRP_E1_str = os.getenv('mip_VRP_E1')
mip_VRP_E1 = float(mip_VRP_E1_str)
mip_sched_r_str = os.getenv('mip_sched_r')
mip_sched_r = float(mip_sched_r_str)
mip_sched_w_str = os.getenv('mip_sched_w')
mip_sched_w = float(mip_sched_w_str)
mip_sched_t_str = os.getenv('mip_sched_t')
mip_sched_t = float(mip_sched_t_str)
storage_set = os.getenv('storage_set')
save_title = os.getenv('save_title')


capacity_s = {}
for i in satellite_locations.index.tolist():
    capacity_s[i] = satellite_locations.at[i,f'capacity_{storage_set}']

#%% Import initial solution
```

```python

N_s = []
results = []
for t_lim_VRP_E2 in t_limits_VRP_E2:
    print(save_title)                    #%%
    V_id = []
    W_id = []
    S_id = []
    X_W_init_s = {}
    Q_W_init_s = {}
    L_W_init_s = {}
    Z_WV_init_s = {}
    Y_init_s = {}
    B_init_s = {}
    A_WV_init_s = {}
    A_DD_init_s = {}
    S_init_s = {}
    LS_init_s = {}
    D_init_s = {}
    D_WV_init_s = {}
    W_init_s = {}
    P_V = {}
    LS_V = {}
    Z_V = {}
    v_s = {}
    L_V = {}
    D_T = {}
    v_d = {}
    canal_nodes_dict = {}
    D_r_VRP_E2 = None
    D_w_VRP_E1 = None
    MIP_VRP_E2 = None
    MIP_VRP_E1 = None
    DC = []
    with open(f'output_VRPs_{save_title}_{Ns}_{t_lim_VRP_E2}.txt', 'r') as f:
        current_var = None
        for line in f:
            line = line.strip()
            if line.endswith(':'):
                current_var = line[:-1]
            elif current_var is not None:
                parts = line.split(': ')
                if current_var == 'V_id':
                    V_id.append(line.strip())
                elif current_var == 'W_id':
                    W_id.append(line.strip())
                elif current_var == 'S_id':
                    S_id.append(line.strip())
                elif current_var == 'DC':
                    DC.append(line.strip())
                elif current_var == 'D_r_VRP_E2':
                    D_r_VRP_E2 = float(line)
                    print(D_r_VRP_E2)
                elif current_var == 'MIP_VRP_E2':
                    MIP_VRP_E2 = float(line)
                    print(MIP_VRP_E2)
                elif current_var == 'D_w_VRP_E1':
                    D_w_VRP_E1 = float(line)
                    print('D_w_VRP_E1')
                elif current_var == 'MIP_VRP_E1':
                    MIP_VRP_E1 = float(line)
                elif len(parts) == 2:
                    key, value = parts
                    if current_var == 'V_id':
                        V_id.append(value.strip())
                    elif current_var == 'LS_V':
                        key, value = line.split(': ')
                        key = key.strip()
                        value = value.strip()
                        LS_V[key] = int(value)
                    elif current_var == 'v_s':
```

```
                    key, value = line.split(': ')
                    key = key.strip()
                    value = value.strip()
                    v_s[key] = value
                elif current_var == 'canal_nodes_dict':
                    key, value = line.split(': ')
                    key = key.strip()
                    value = value.strip()
                    canal_nodes_dict[key] = int(value)
                elif current_var == 'v_d':
                    value = value.replace("'", "")
                    v_d[key] = int(value)
                else:
                    key_parts = line.split('(')[1].split(')')[0].split(', ')
                    key_parts = [part.strip("'") for part in key_parts]
                    indices = tuple(key_parts)
                    value = line.split(': ')[-1]
                    if current_var == 'X_W':
                        X_W_init_s[indices] = float(value)
                    elif current_var == 'Q_W':
                        Q_W_init_s[indices] = float(value)
                    elif current_var == 'L_W':
                        L_W_init_s[indices] = float(value)
                    elif current_var == 'Z_WV':
                        Z_WV_init_s[indices] = float(value)
                    elif current_var == 'Y':
                        Y_init_s[indices] = float(value)
                    elif current_var == 'B':
                        B_init_s[indices] = float(value)
                    elif current_var == 'A_WV':
                        A_WV_init_s[indices] = float(value)
                    elif current_var == 'A_DD':
                        A_DD_init_s[indices] = float(value)
                    elif current_var == 'S':
                        S_init_s[indices] = float(value)
                    elif current_var == 'LS':
                        LS_init_s[indices] = float(value)
                    elif current_var == 'D':
                        D_init_s[indices] = float(value)
                    elif current_var == 'D_WV':
                        D_WV_init_s[indices] = float(value)
                    elif current_var == 'W':
                        W_init_s[indices] = float(value)
                    elif current_var == 'P_V':
                        P_V[indices] = float(value)
                    elif current_var == 'Z_V':
                        Z_V[indices] = float(value)
                    elif current_var == 'D_T':
                        D_T[indices] = float(value)
                    elif current_var == 'L_V':
                        L_V[indices] = float(value)
zero = ['zero']
WV_id = W_id + V_id
WV0_id = zero + WV_id
W0_id = zero + W_id
V0_id = zero + V_id
DS_id = DC + S_id

#%% Create initial solution for scheduling road vehicles T_V[l,k,r]
print('Creating initial solution road scheduling for Ns:', Ns)
vehicles = pd.read_excel(path +"Road_vehicles.xlsx", index_col=0)
R_v_ = vehicles.index.tolist()
R_v = R_v_[0:len(V_id)]

T_V_init_s = {}
for r in R_v:
    for l in V0_id:
        for k in V0_id:
            T_V_init_s[l,k,r] = 0

numb = 0
```

```
    for v in V_id:
        T_V_init_s['zero',v,R_v[numb]] = 1
        T_V_init_s[v,'zero',R_v[numb]] = 1
        numb += 1

    T_V_init_s_new = {}
    for r in R_v:
        for l in V0_id:
            for k in V0_id:
                T_V_init_s_new[l,k,r] = 0
    R_v_new_init = []
    V_id_left = V_id.copy()
    for r in R_v:
        r_use = 0
        trip = 1
        for v in V_id_left:
            depot = v_d[v]
            T_V_init_s_new['zero',v,r] = 1
            s = v_s[v]
            trip = 0
            arrival = A_WV_init_s[s,v]
            for k in V_id_left:
                if depot == v_d[k]:
                    if Y_init_s[s,k,v] == 1:
                        if A_WV_init_s[s,k] >= arrival + P_V[v,k]:
                            T_V_init_s_new[v,k,r] = 1
                            T_V_init_s_new[k,'zero',r] = 1
                            V_id_left.remove(v)
                            V_id_left.remove(k)
                            trip = 1
                            r_use = 1
                            break
                    else:
                        continue
            if trip == 0:
                T_V_init_s_new[v,'zero',r] = 1
                V_id_left.remove(v)
                r_use = 1
            if r_use == 1:
                R_v_new_init.append(r)
            break
    #%%
    R_v = R_v_new_init.copy()
    T_V_init_s_new_1 = {}
    V_done = {}
    for r in R_v:
        for l in V0_id:
            for k in V0_id:
                T_V_init_s_new_1[l,k,r] = T_V_init_s_new[l,k,r]
                if T_V_init_s_new_1[l,k,r] == 1:
                    V_done[k] = 1
                    V_done[l] = 1



    #%% Schedule road vehicles
    print('Working on road scheduling for Ns:', Ns)
    start_sched_r = time.time()
    model = gb.Model('Scheduling_road')
    np.random.seed(123)
    time_limit = t_lim_sched_road
    K = 9999


    X_W =  X_W_init_s
    Q_W = Q_W_init_s


    total_load = 0
    for i in S_id:
            total_load += LS_V[i]
```

```
            load_delivered = 0
            load_delivered = quicksum(Q_W[i,w]  for w in W_id)
            print(i,'load delivered by w:', load_delivered, 'load required by v:',LS_V[i])
    print('total load required: ', total_load)

    # Binary variable, Y[i,k,l] = 1 if both k and l visit i
    Y = {}
    for k in WV_id:
        for l in WV_id:
            for i in DS_id:
                Y[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'Y')

    # Arrival time of vehicle r at i
    A_W = {}
    for i in DS_id:
        for w in W_id:
            A_W[i,w] = model.addVar(lb = -500, vtype = GRB.CONTINUOUS, name = 'A_W')

    # Arrival time of vehicle r at i
    A_WV = {}
    for i in DS_id:
        for k in WV_id:
            A_WV[i,k] = model.addVar(lb = -500, ub = 999999, vtype = GRB.CONTINUOUS, name = 'A_WV')

    # Difference in arrival times of vehicle
    A_D = {}
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                A_D[i,k,l] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'A_D')

    # Difference in arrival times of vehicle
    A_DD = {}
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                A_DD[i,k,l] = model.addVar(lb = -999999, ub = 999999, vtype = GRB.CONTINUOUS, name = 'A_DD')

    # Customer or satellite is visited by vehicle r: = 1, if not: = 0
    Z_WV = {}
    for k in WV_id:
        for i in DS_id:
            Z_WV[i,k] =  model.addVar(vtype = GRB.BINARY, name = 'Z_WV')

    # Accumulated load of road vehicle r at customer i
    L_W = {}
    for w in WV_id:
        for i in DS_id:
            L_W[i,w] =  model.addVar(lb=0.0, vtype = GRB.CONTINUOUS, name = 'L_W')


    # Accumulated load delivered to satellite i by vehicles before and including k
    LS = {}
    for i in S_id:
        for k in WV0_id:
            LS[i,k] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'LS')


    # Number of water vehicles used
    Nw = {}
    for w in W_id:
        Nw[w] = model.addVar(vtype = GRB.BINARY, name = 'Nw')

    # Stock at satellite i after arrival of vehicle k
    S = {}
    for i in S_id:
        for k in WV_id:
            S[i,k] = model.addVar(lb = -100, vtype = GRB.INTEGER, name = 'S')
```

```
    B = {}
    for i in S_id:
        for k in WV0_id:
            for l in WV0_id:
                B[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'B')


    D_w = {}
    for w in W0_id:
        D_w[w] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'D_w')


    #T is matrix per road vehicle r, with trips k,l in V0, if r first performs trip k and
    then l, T[k,l,r] = 1
    T_V = {}
    for r in R_v:
        for k in V0_id:
            for l in V0_id:
                T_V[k,l,r] = model.addVar(vtype = GRB.BINARY, name = 'T_V')

    A_R = {}
    for r in R_v:
        for v in V0_id:
            A_R[v,r] = model.addVar(lb = -500, vtype = GRB.CONTINUOUS, name = 'A_R')

    N_R = {}
    for r in R_v:
        N_R[r] = model.addVar(vtype = GRB.BINARY, name = 'N_R')

    Z_RV = {}
    for r in R_v:
        for v in V_id:
            Z_RV[v,r] = model.addVar(vtype = GRB.BINARY, name = 'Z_RV')

    C_R = {}
    for r in R_v:
        C_R[r] = model.addVar(vtype = GRB.CONTINUOUS, name = 'C_R')

    # New for departure times
    D = {}
    for i in S_id:
        for k in WV0_id:
            for l in WV0_id:
                D[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'D')


    D_WV = {}
    for i in DS_id:
        for k in WV_id:
            D_WV[i,k] = model.addVar(lb = 0.0, ub = 999999, vtype = GRB.CONTINUOUS, name = '
    D_WV')

    W = {}
    for i in DS_id:
        for w in W0_id:
            W[i,w] = model.addVar(vtype = GRB.CONTINUOUS, name = 'W')

    D_r = {}
    for r in R_v:
        D_r[r] = model.addVar(vtype = GRB.CONTINUOUS, name = 'D_r')

    # Objective function
    model.setObjective( 0.1* quicksum(D_r[r] for r in R_v) + 500* quicksum(N_R[r] for r in
    R_v)) # + 500 * quicksum(N_F[f] for f in F))


    model.modelSense = GRB.MINIMIZE
    model.update()

    # Constraints
    # 1. A vehicle never goes from i to i
```

```
    for w in W_id:
        for i in DS_id:
            for j in DS_id:
                if i == j:
                    constr_w_1 = model.addConstr(X_W[i,j,w] == 0, name='Constr_1')

    # 2. Vehicle r can only leave node if it also arrived there
    for w in W_id:
        for i in DS_id:
            # if i != j:
                constr_w_2 = model.addConstr(quicksum(X_W[i,j,w] for j in DS_id) == quicksum(X_W[j,i,w] for j in DS_id), name='Constr_2')

    # 3. Nodes that are visited by vehicle w
    for w in W_id:
        for i in DS_id:
            constr_w_3b = model.addConstr(Z_WV[i,w] == quicksum(X_W[i,j,w] for j in DS_id), name='Constr_3')

    # 4b. Nodes that are visited by vehicle r
    for v in V_id:
        for i in DS_id:
            constr_w_4c = model.addConstr(Z_WV[i,v] == Z_V[i,v], name='Constr_4')

    # New
    # 5. The demand delivered to i is zero if vehicle r does not visit i
    for w in W_id:
        for i in DS_id:
            constr_w_5 = model.addGenConstrIndicator (Z_WV[i,w], False, Q_W[i,w], GRB.EQUAL, 0, name='Constr_5')

    # 6. Demand satisfaction constraint
    for i in S_id:
            constr_w_6 = model.addConstr(quicksum(Q_W[i,w] for w in W_id) == LS_V[i], name='Constr_6') #s_v[i])) #
            constr_w_6b = model.addConstr(Q_W[i,'zero'] == 0, name='Constr_6b')

    # New
    # 7. No load is delivered to DC
    # 8. The accumulated load at the DC is zero
    for w in W_id:
        DC = DC if isinstance(DC, list) else [DC]
        for i in DC:
            constr_w_7 = model.addConstr (Q_W[i,w] == 0, name='Constr_7')
            constr_w_8 = model.addConstr(L_W[i,w] == 0, name='Constr_8')

    # 8b. No load delivered by road vehicles
    for v in V_id:
        for i in DS_id:
            constr_w_8b = model.addConstr(Q_W[i,v] == 0, name='Constr_8b')
            constr_w_8c = model.addConstr(L_W[i,v] == 0, name='Constr_8c')

    # 9a_new. With X_W as an input, the constraint can be rewritten as:
    for w in W_id:
        for i in DS_id:
            for j in S_id:
                if X_W[i,j,w] == 1:
                    constr_9a_new = model.addConstr(L_W[j,w] - L_W[i,w] - Q_W[j,w] == 0, name = 'Constr_9a_new')

    # 9b. No L_R if not visited
    for w in W_id:
        for i in DS_id:
            constr_w_9b = model.addGenConstrIndicator (Z_WV[i,w], False, L_W[i,w], GRB.EQUAL, 0, name='Constr_9b')

    # 9c. The load delivered to customer i by vehicle r is always less than or equal to the accumulated load of r at customer i:
    for w in W_id:
        for i in S_id:
            constr_w_9c = model.addConstr(Q_W[i,w] <= L_W[i,w], name='Constr_9c')
```

```
    # 9d. The accumulated load of vehicle r at customer i is always less than or equal to the maximum capacity of vehicle r:
    for w in W_id:
        for i in S_id:
            constr_w_9d = model.addConstr( L_W[i,w] <= capacity_fe, name='Constr_9d')


    # # Arrival time constraints:

    # 10_new. With X_W as input
    for w in W_id:
        for i in DS_id:
            for j in S_id:
                if X_W[i,j,w] == 1:
                    constr_10_new = model.addConstr(A_WV[j,w] - A_WV[i,w] - dist_fe.at[canal_nodes_dict[i],canal_nodes_dict[j]]  / (speed_fe * 60)  - W[i,w] - Q_W[i,w] * 0.2 >= 0, name='Constr_10_new' )

    # 10b. The arrival time at the first satellite of trip w is the arrival time at the depot - travel time - service time
    for w in W_id:
        for j in S_id:
            DC = DC if isinstance(DC, list) else [DC]
            for i in DC:
                if X_W[i,j,w] == 1:
                    constr_time_10b = model.addConstr(A_WV[j, w] - A_WV[i, w] - dist_fe.at[canal_nodes_dict[i], canal_nodes_dict[j]] / (speed_fe * 60) - service_time_fe / 60 == 0, name='Constr_10b')

    # 11. Binary variable Y[i,k,l] is one if both k and l visit i
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                if k != l:
                    constr_Y_11 = model.addConstr(Y[i,k,l] == gb.and_(Z_WV[i,k], Z_WV[i,l]), name='Constr_11')


    # 12. Arrival times of vehicles at satellites cannot be the same
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                constr_time_12a = model.addConstr(A_DD[i,k,l] == A_WV[i,k] - A_WV[i,l], name='Constr_12a')
                constr_time_12b = model.addConstr(A_D[i,k,l] == gb.abs_(A_DD[i,k,l]), name='Constr_12b')


    # 13a. Arrival times of road vehicles at satellites cannot be the same
    for i in S_id:
        for k in V_id:
            for l in V_id:
                if k != l:
                    constr_time_13a = model.addGenConstrIndicator(Y[i,k,l], True, A_D[i,k,l], GRB.GREATER_EQUAL, transship_s, name='Constr_13a') #180)
                    constr_time_13a_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13a_1')

    # 13b. Arrival times of a water vehicles is later than the departure time of another water vehicle
    for i in S_id:
        for k in W_id:
            for l in W_id:
                if k != l:
                    constr_time_13b = model.addGenConstrIndicator(B[i,k,l], True, A_WV[i,k] - D_WV[i,l] , GRB.GREATER_EQUAL, 0, name='Constr_13b') #600)
                    constr_time_13b_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13b_1')

    # 13b. Arrival times of water and road vehicles at satellites cannot be the same
```

```
    for i in S_id:
        for k in W_id:
            for l in V_id:
                if k != l:
                    constr_time_13c = model.addGenConstrIndicator(Y[i,k,l], True, A_D[i,k,l], GRB.GREATER_EQUAL, 0.01, name='Constr_13c') #600)
                    constr_time_13c_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13c_1')

    #13c. Arrival times at satellites cannot be later than the maximum time span
    for i in S_id:
        for k in WV_id:
            constr_time_13d = model.addConstr(D_WV[i,k] <= time_span, name='Constr_13d')


    # 14. Arrival time is infinite if a vehicle does not visit satellite i
    for i in S_id:
        for k in WV_id:
            constr_time_14 = model.addGenConstrIndicator(Z_WV[i,k], False, A_WV[i,k], GRB.EQUAL, 0)

    # # Satellite synchronisation constraints:

    # 15. Binary variable = 1 if vehicle k arrives at the same time or after vehicle l
    for i in S_id:
        for k in WV_id:
            constr_binary_150 = model.addGenConstrIndicator(Z_WV[i,k], True, B[i,k,'zero'], GRB.EQUAL, 1 )
            for l in WV_id:
                constr_binary_15a = model.addGenConstrIndicator(Y[i,k,l], True, A_WV[i,k] - K * B[i,k,l] - A_WV[i,l], GRB.LESS_EQUAL, 0 )
                constr_binary_15b = model.addGenConstrIndicator(Y[i,k,l], True, B[i,k,l] + B[i,l,k], GRB.EQUAL, 1)
                constr_binary_15c = model.addConstr(B[i,k,l] + B[i,l,k] <= 1)
                constr_binary_15d = model.addGenConstrIndicator(Z_WV[i,k], False, B[i,k,l], GRB.EQUAL, 0 )
                constr_binary_15e = model.addGenConstrIndicator(Z_WV[i,k], False, B[i,l,k], GRB.EQUAL, 0 )

    # 16. Load delivered to satellite i by all vehicles before k and k
    for i in S_id:
        for k in WV0_id:
            for l in WV0_id:
                    constr_load_16a = model.addGenConstrIndicator(B[i,k,l], True, LS[i,k] - LS[i,l] - Q_W[i,k], GRB.GREATER_EQUAL,0)

    for i in S_id:
        for k in WV_id:
            constr_load_16b = model.addConstr(LS[i,k] <= quicksum(Q_W[i,w] for w in W_id))
            constr_load_16c = model.addGenConstrIndicator(Z_WV[i,k], False, LS[i,k], GRB.EQUAL, 0 )

    # 17. New Stock at satellites constraints
    for i in S_id:
        for k in WV_id:
            constr_stock_17a = model.addGenConstrIndicator (Z_WV[i,k], True, S[i,k] + quicksum(L_V[i,l] * B[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.EQUAL, 0)

    for i in S_id:
        for k in WV_id:
            constr_stock_17b = model.addConstr(S[i,k] >= 0)
            constr_stock_17c = model.addConstr(S[i,k] <= capacity_s[i] + capacity_fe)

    for w in W_id:
        constr_water_km = model.addConstr(D_w[w] == quicksum(dist_fe.at[canal_nodes_dict[i], canal_nodes_dict[j]] * X_W[i,j,w] for i in DS_id for j in DS_id))


    for w in W_id:
        for i in S_id:
            constr_Nw = model.addGenConstrIndicator(Z_WV[i,w], True, Nw[w], GRB.EQUAL, 1)
```

```
    # Road vehicles scheduling

    # Each vehicle r can only leave the depot once
    for r in R_v:
        constr_18f = model.addConstr(quicksum(T_V['zero',k,r] for k in V0_id) <= 1)

    # Each trip is performed once
    for k in V_id:
        constr_18b = model.addConstr(quicksum(T_V[l,k,r] for l in V0_id for r in R_v) == 1)

    # Trip k can be performed by vehicle r if the start time of trip k is later than the end time of trip l
    for r in R_v:
        for k in V_id:
            for l in V0_id:
                constr_18c = model.addGenConstrIndicator(T_V[l,k,r], True, A_R[k,r] - A_R[l,r] - P_V[l,k], GRB.GREATER_EQUAL, 0 )

    # A trip can never be performed after itself
    for r in R_v:
        for l in V0_id:
            constr_18d = model.addConstr(T_V[l,l,r] == 0)

    # Vehicle r can only end trip l if it also started it
    for r in R_v:
        for l in V0_id:
            # if i != j:
                constr_18e = model.addConstr(quicksum(T_V[l,k,r] for k in V0_id) == quicksum(T_V[k,l,r] for k in V0_id))

    # Number of road vehicles used
    for r in R_v:
        for k in V_id:
            constr_19 = model.addGenConstrIndicator(T_V['zero',k,r], True, N_R[r], GRB.EQUAL, 1)


    for r in R_v:
        constr_19d = model.addConstr(quicksum(T_V['zero',k,r] for k in V_id) >= N_R[r], name = 'constr_19d')

    # Z_RV = 1 if r performs trip v
    for k in V_id:
        for r in R_v:
            constr_20a = model.addConstr(Z_RV[k,r] == quicksum(T_V[l,k,r] for l in V0_id))

    # Set A_R to zero if r does not perform trip
    for v in V_id:
        for r in R_v:
            constr_20b = model.addGenConstrIndicator(Z_RV[v,r], False, A_R[v,r], GRB.EQUAL, 0)

    # Connect A_R with A_WV
    for v in V_id:
        constr_20c = model.addConstr(A_WV[v_s[v],v] == quicksum(A_R[v,r] for r in R_v))

    # Completion time for vehicle r is the start time of the last trip + the time to perform the last trip
    for r in R_v:
        for v in V_id:
            constr_21a = model.addGenConstrIndicator(T_V[v,'zero',r], True, C_R[r] - A_R[v,r] - P_V[v,'zero'], GRB.EQUAL, 0)


    for k in WV_id:
        for i in S_id:
            constr_time_span = model.addConstr(A_WV[i,k] >= 0, name = 'constr_time_span')
```

```
    # New for departure times
    for i in S_id:
        for v in V_id:
            constr_departure_1 = model.addGenConstrIndicator(Z_WV[i,v], True, D_WV[i,v] - A_WV[i,v] - transship_s / 60, GRB.EQUAL, 0, name = 'constr_dep_1')
        for w in W_id:
            constr_departure_2 = model.addGenConstrIndicator(Z_WV[i,w], True, D_WV[i,w] - A_WV[i,w] - W[i,w] - Q_W[i,w] * 0.2, GRB.EQUAL, 0, name = 'constr_dep_2' )


    for i in S_id:
        for k in V0_id:
            for l in V0_id:
                constr_departure_3 = model.addGenConstrIndicator(B[i,k,l], True, D[i,k,l], GRB.EQUAL, 1, name = 'constr_dep_3')
        for k in W0_id:
            for l in W0_id:
                constr_departure_4 = model.addGenConstrIndicator(B[i,k,l], True, D[i,k,l], GRB.EQUAL, 1, name = 'constr_dep_4')

        for k in WV_id:
            constr_departure_8 =  model.addGenConstrIndicator (Z_WV[i,k], False, quicksum(D[i,k,l] for l in WV_id) + quicksum(D[i,l,k] for l in WV_id), GRB.EQUAL, 0, name = 'constr_dep_8')
            for l in WV_id:
                constr_departure_5 = model.addGenConstrIndicator(Y[i,k,l], True, D_WV[i,k] - K* D[i,k,l] - D_WV[i,l], GRB.LESS_EQUAL, 0, name = 'constr_dep_5')
                constr_departure_6 = model.addGenConstrIndicator(Y[i,k,l], True, D[i,k,l] + D[i,l,k], GRB.EQUAL, 1, name = 'constr_dep_6')


    for i in S_id:
        for k in W_id:
            constr_departure_7 = model.addGenConstrIndicator (Z_WV[i,k], True, quicksum(L_V[i,l] * D[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.LESS_EQUAL, 0, name = 'constr_dep_7')
            constr_departure_7_b = model.addGenConstrIndicator (Z_WV[i,k], True, quicksum(L_V[i,l] * D[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.GREATER_EQUAL, - capacity_s[i], name = 'constr_dep_7_b')


    # Distance on the road per vehicle
    for r in R_v:
        constr_distance_r = model.addConstr(D_r[r] == quicksum(T_V[l,k,r] * D_T[l,k] for l in V0_id for k in V0_id))


    for (i,w), value in L_W_init_s.items():
        if w in WV_id:
            L_W[i,w].start = value

    for (i,w), value in Z_WV_init_s.items():
        if w in WV_id:
            Z_WV[i,w].start = value

    for (i,k,l), value in Y_init_s.items():
        if k in WV_id:
            if l in WV_id:
                Y[i,k,l].start = value

    for (i,k,l), value in B_init_s.items():
        if k in WV0_id:
            if l in WV0_id:
                B[i,k,l].start = value

    for (i, k), value in A_WV_init_s.items():
        if k in WV_id:
            A_WV[i,k].start = value

    for (i,k,l), value in A_DD_init_s.items():
        if k in WV_id:
```

```
            if l in WV_id:
                A_DD[i,k,l].start = value

    for (i,w), value in S_init_s.items():
        if w in WV_id:
            S[i,w].start = value

    for (i,k), value in LS_init_s.items():
        if k in WV0_id:
            LS[i,k].start = value

    for (i,k,l), value in T_V_init_s_new_1.items():
        T_V[i,k,l].start = value

    for (i,k,l), value in D_init_s.items():
        if k in WV0_id:
            if l in WV0_id:
                D[i,k,l].start = value

    for (i, k), value in D_WV_init_s.items():
        if k in WV_id:
            D_WV[i,k].start = value

    for (i, k), value in W_init_s.items():
        if k in W0_id:
            W[i,k].start = value

    # Start optimisation
    print("start optimizing")
    model.setParam( 'OutputFlag', True)
    model.setParam ('MIPGap', mip_sched_r);
    model.setParam('FeasibilityTol', 1e-5)
    model.setParam('MIPFocus', 0)
    model.setParam('SubMIPNodes', 20000)
    model.setParam('Seed', 123)
    model.setParam('SoftMemLimit', 120)
    model.setParam('Threads', 40)

    if time_limit:
        model.setParam('Timelimit', time_limit)
    model._obj = None
    model._bd = None
    model._obj_value = []
    model._time = []
    model._start = time.time()
    model.optimize()
    MIP_sched_r = model.MIPGap
    end_sched_r = time.time()
    time_sched_r = end_sched_r - start_sched_r



    #%% Save solutions

    r_used_road = 0
    for r in R_v:
        if N_R[r].X == 1:
            r_used_road += 1
    print (r_used_road)

    Nr_R_r = r_used_road


    max_complete = 0
    for r in R_v:
        for k in V0_id:
            for l in V0_id:
                if T_V[l,k,r].X == 1:
                    if C_R[r].X > max_complete:
                        max_complete = C_R[r].X
    max_start_R_r = max_complete
```

```
    W_used_VRP = []
    for w in W_id:
        w_visits = 0
        for i in S_id:
            if Z_WV[i,w].X == 1:
                    w_visits += 1
        if w_visits >= 1:
            W_used_VRP.append(w)

    Nr_w_r = len(W_used_VRP)

    road_km_R_r = {}
    total_road_km = 0
    for r in R_v:
        save_road_km = 0
        for l in V0_id:
            for k in V0_id:
                if T_V[l,k,r].X == 1:
                    save_road_km += D_T[l,k]
                    total_road_km += D_T[l,k]
        road_km_R_r[r] = save_road_km
    D_r_r = total_road_km


    X_W_init_sw = X_W
    Y_init_sw = model.getAttr('X', Y)
    A_WV_init_sw = model.getAttr('X', A_WV)
    A_D_init_sw = model.getAttr('X', A_D)
    A_DD_init_sw = model.getAttr('X', A_DD)
    Q_W_init_sw = Q_W
    Z_WV_init_sw = model.getAttr('X', Z_WV)
    L_W_init_sw = model.getAttr('X', L_W)
    LS_init_sw = model.getAttr('X', LS)
    S_init_sw = model.getAttr('X', S)
    B_init_sw = model.getAttr('X', B)
    T_V_init_sw = model.getAttr('X', T_V)
    A_R_init_sw = model.getAttr('X', A_R)
    D_init_sw = model.getAttr('X', D)
    D_WV_init_sw = model.getAttr('X', D_WV)
    W_init_sw = model.getAttr('X', W)
    D_w_init_sw = model.getAttr('X',D_w)

    # Select only R_v that are used for in road vehicle scheduling
    R_v_new = []
    for r in R_v:
        g = []
        for k in V_id:
            G = T_V['zero',k,r].X
            if G > 0:
                g = [r]
                break
        R_v_new = R_v_new + g

    #Select only values of T_V for new R_V
    T_V_new_init_sw = {}

    for r in R_v_new:
        for l in V0_id:
            for k in V0_id:
                T_V_new_init_sw[k,l,r] = T_V_init_sw[k,l,r]

    R_v = R_v_new.copy()
    R_sched_r = len(R_v)
    print('Distance on the road after road scheduling: ', D_r_r)

    #%%
    with open(f'output_road_{save_title}_{Ns}_{t_lim_VRP_E2}.txt', 'w') as f:
        f.write(f'D_r_VRP_E2:\n{D_r_VRP_E2}\n')
        f.write(f'MIP_VRP_E2:\n{MIP_VRP_E2}\n')
        f.write(f'D_w_VRP_E1:\n{D_w_VRP_E1}\n')
```

```python
        f.write(f'MIP_VRP_E1:\n{MIP_VRP_E1}\n')
        f.write(f'R_sched_r:\n{R_sched_r}\n')
        f.write(f'D_r_r:\n{D_r_r}\n')
        f.write(f'MIP_sched_r:\n{MIP_sched_r}\n')
        for var_name, var_values in [
            ('X_W', X_W_init_sw),
            ('Y', Y_init_sw),
            ('A_WV', A_WV_init_sw),
            ('A_D', A_D_init_sw),
            ('A_DD',A_DD_init_sw),
            ('Q_W', Q_W_init_sw),
            ('Z_WV', Z_WV_init_sw),
            ('L_W', L_W_init_sw),
            ('LS', LS_init_sw),
            ('S', S_init_sw),
            ('B', B_init_sw),
            ('D', D_init_sw),
            ('D_WV', D_WV_init_sw),
            ('W', W_init_sw),
            ('T_V', T_V_new_init_sw),
            ('A_R', A_R_init_sw),
            ('D_w', D_w_init_sw),
            ('P_V', P_V),
            ('L_V', L_V),
            ('LS_V', LS_V),
            ('Z_V', Z_V),
            ('v_s', v_s),
            ('L_V', L_V),
            ('D_T', D_T),
            ('v_d', v_d),
            ('canal_nodes_dict', canal_nodes_dict)
        ]:
            f.write(f'{var_name}:\n')
            for key, value in var_values.items():
                if isinstance(value, gb.LinExpr):
                    value = value.getValue()
                f.write(f'  {key}: {value}\n')
        f.write('V_id:\n')
        for v in V_id:
            f.write(f'{v}\n')
        f.write('W_id:\n')
        for w in W_id:
            f.write(f'{w}\n')
        f.write('S_id:\n')
        for s in S_id:
            f.write(f'{s}\n')
        f.write('R_v:\n')
        for r in R_v:
            f.write(f'{r}\n')
        f.write('DC:\n')
        for d in DC:
            f.write(f'{d}\n')
    #%%
    model.dispose()
```

B.4.2. Vessel Scheduling

```python
#%% Import libraries
import gurobipy as gb
import time
import os
import numpy as np
import pandas as pd
import pickle
import math
import copy
import sys
import matplotlib.pyplot as plt
from openpyxl import load_workbook
from gurobipy import quicksum, GRB

```

```python
#%% Set path
server = 'True'

if server == 'False':
    path = os.getcwd() + "\Inputs\\"
    path_out = os.getcwd() + "\Outputs\\"
    from FLP_solver_definition_number_customers_horeca_sets_Laudy import FLP_num_cust
    from FLP_solver_definition_horeca_sets_capacity_assignment import FLP_capacity

if server == 'True':
    path = os.getcwd() + "/Inputs/"
    path_out = os.getcwd() + "/Outputs/"


#%% Scenario inputs
directed = 'true'                # Indicate wether to use directed or undirected distance matrix
FLP_constraint = 'num_cust'      # Which FLP constraint to use, either capacity or num_cust
Nc = 750                         # Insert the number of customers to consider
horeca_sets = np.arange(1,11)    # Which horeca sets to evaluate
horeca_set = 1                   # If not testing all horeca sets, insert one to evaluate


#%% Import network and scenario data
df_horeca_demand_scenarios = pd.read_excel(path + f'df_horeca_demand_scenarios.xlsx', index_col=0)
df_horeca_demand_scenarios.index = df_horeca_demand_scenarios.index.astype(str)
df_horeca_data_info = pd.read_excel(path + f'df_horeca_data_info.xlsx', index_col=0)
df_horeca_data_info.index = df_horeca_data_info.index.astype(str)
customer_locations = df_horeca_data_info.iloc[:,0]

if server == 'False':
    df_SE_shortest_dist_directed_False = pickle.load(open(path + 'df_SE_shortest_dist_directed-False_nodes_all.pickle', 'rb'))
    df_SE_shortest_dist_directed_True_1 = pickle.load(open(path + 'df_SE_shortest_dist_directed-True_nodes_all.pickle', 'rb'))
    df_SE_shortest_dist_directed_True = df_SE_shortest_dist_directed_True_1.fillna(1001)
    dict_FE_shortest_dist_directed_True_1 = pickle.load(open(path + 'dict_FE_shortest_dist_directed-True_nodes_all.pickle', 'rb'))

if server == 'True':
    pickle_off = open(path + 'df_SE_shortest_dist_directed-True_nodes_all.pickle', 'rb')
    df_SE_shortest_dist_directed_True_1 = pd.read_pickle(pickle_off)
    df_SE_shortest_dist_directed_True = df_SE_shortest_dist_directed_True_1.fillna(1001)

    pickle_off = open(path + 'dict_FE_shortest_dist_directed-True_nodes_all.pickle', 'rb')
    dict_FE_shortest_dist_directed_True_1 = pd.read_pickle(pickle_off)
    df_SE_shortest_dist_directed_False = dict_FE_shortest_dist_directed_True_1
assigned = []
indices = []
customers = [[0]*3]*len(customer_locations)
for customer_id in df_horeca_data_info.index.tolist():
    if df_horeca_demand_scenarios.at[f'{customer_id}', f'set_{horeca_set}'] > 0:
        indices.append(customer_id)
        assigned.append({'road_node':int(df_horeca_data_info.at[customer_id, 'road_node']), 'demand':int(df_horeca_demand_scenarios.at[f'{customer_id}', f'set_{horeca_set}'])} )
customers = pd.DataFrame(assigned, index= indices)# df_horeca_data_info.index.tolist() )


satellite_locations = pd.read_excel(path + "satellite_nodes_storage_full.xlsx", index_col=0)
vehicles = pd.read_excel(path +"Road_vehicles.xlsx", index_col=0)
road_nodes = pd.read_excel(path + "satellites_customers_road_nodes.xlsx", index_col = 0)

if directed == 'true':
    dist = df_SE_shortest_dist_directed_True
elif directed == 'false':
    dist = df_SE_shortest_dist_directed_False

#%% Parameters

speed_v = int(os.getenv('speed_v'))
```

```
transship_s = int(os.getenv('transship_s'))
transship_c = int(os.getenv('transship_c'))
fev_profile = 5
capacity_fe = int(os.getenv('capacity_fe'))
speed_fe_str =  os.getenv('speed_fe')
speed_fe = float(speed_fe_str)
service_time_fe = int(os.getenv('service_time_fe'))
capacity_s = int(os.getenv('capacity_s'))
capacity_se = int(os.getenv('capacity_se'))

t_limits_VRP_E2_str = os.getenv('t_limits_VRP_E2')
t_limits_VRP_E2 = eval(t_limits_VRP_E2_str)
t_lim_VRP_E1 = int(os.getenv('t_lim_VRP_E1'))
t_lim_sched_road = int(os.getenv('t_lim_sched_road'))
t_lim_sched_water = int(os.getenv('t_lim_sched_water'))
t_lim_sched_total = int(os.getenv('t_lim_sched_total'))
time_span = int(os.getenv('time_span'))
mip_VRP_E2_str = os.getenv('mip_VRP_E2')
mip_VRP_E2 = float(mip_VRP_E2_str)
mip_VRP_E1_str = os.getenv('mip_VRP_E1')
mip_VRP_E1 = float(mip_VRP_E1_str)
mip_sched_r_str = os.getenv('mip_sched_r')
mip_sched_r = float(mip_sched_r_str)
mip_sched_w_str = os.getenv('mip_sched_w')
mip_sched_w = float(mip_sched_w_str)
mip_sched_t_str = os.getenv('mip_sched_t')
mip_sched_t = float(mip_sched_t_str)
storage_set = os.getenv('storage_set')
save_title = os.getenv('save_title')  #

Ns = int(os.getenv('NrSatellites'))



df_fe_distance_matrix = dict_FE_shortest_dist_directed_True_1[f'vessel_profile_{fev_profile}'
    ].copy()
dist_fe = df_fe_distance_matrix.fillna(99999)

# New distance matrix for multiple water vehicle depots:
dict_FE_new = pd.read_csv(path + 'distance_matrix_DCs.csv',sep=';',header=None)
dist_fe_new = pd.DataFrame(dict_FE_new)
dist_fe_new.index = dist_fe_new.index + 1
new_index = {old_index:old_index + 1 for old_index in dist_fe_new.columns}
dist_fe_new = dist_fe_new.rename(columns=new_index)
dist_fe = dist_fe_new.fillna(99999)
mip_s_r = 0.1


capacity_s = {}
for i in satellite_locations.index.tolist():
    capacity_s[i] = satellite_locations.at[i,f'capacity_{storage_set}']


#%% Import initial solution

N_s = []
results = []
for t_lim_VRP_E2 in t_limits_VRP_E2:
    V_id = []
    W_id = []
    S_id = []
    R_v = []
    X_W_init_sw = {}
    Q_W_init_sw = {}
    L_W_init_sw = {}
    Z_WV_init_sw = {}
    Y_init_sw = {}
    B_init_sw = {}
    A_WV_init_sw = {}
    A_DD_init_sw = {}
    S_init_sw = {}
```

```
    LS_init_sw = {}
    T_V_new_init_sw = {}
    D_init_sw = {}
    D_WV_init_sw = {}
    W_init_sw = {}
    A_R_init_sw = {}
    D_w_init_sw = {}

    P_V = {}
    LS_V = {}
    Z_V = {}
    v_s = {}
    L_V = {}
    D_T = {}
    v_d = {}
    D_r_VRP_E2 = None
    D_w_VRP_E1 = None
    D_r_r = None
    MIP_VRP_E2 = None
    MIP_VRP_E1 = None
    MIP_sched_r = None
    R_sched_r = None
    canal_nodes_dict = {}
    DC = []
    with open(f'output_road_{save_title}_{Ns}_{t_lim_VRP_E2}.txt', 'r') as f:
        current_var = None
        for line in f:
            line = line.strip()
            if line.endswith(':'):
                current_var = line[:-1]
            elif current_var is not None:
                parts = line.split(': ')
                if current_var == 'V_id':
                    V_id.append(line.strip())
                elif current_var == 'W_id':
                    W_id.append(line.strip())
                elif current_var == 'S_id':
                    S_id.append(line.strip())
                elif current_var == 'R_v':
                    R_v.append(line.strip())
                elif current_var == 'DC':
                    DC.append(line.strip())
                elif current_var == 'D_r_VRP_E2':
                    D_r_VRP_E2 = float(line)
                elif current_var == 'MIP_VRP_E2':
                    MIP_VRP_E2 = float(line)
                elif current_var == 'D_w_VRP_E1':
                    D_w_VRP_E1 = float(line)
                elif current_var == 'MIP_VRP_E1':
                    MIP_VRP_E1 = float(line)
                elif current_var == 'D_w_VRP_E1':
                    D_w_VRP_E1 = float(line)
                elif current_var == 'R_sched_r':
                    R_sched_r = float(line)
                elif current_var == 'D_r_r':
                    D_r_r = float(line)
                elif current_var == 'MIP_sched_r':
                    MIP_sched_r = float(line)
                elif len(parts) == 2:
                    key, value = parts
                    if current_var == 'LS_V':
                        key, value = line.split(': ')
                        key = key.strip()
                        value = value.strip()
                        LS_V[key] = int(value)
                    elif current_var == 'v_s':
                        key, value = line.split(': ')
                        key = key.strip()
                        value = value.strip()
                        v_s[key] = value
                    elif current_var == 'canal_nodes_dict':
```

```
                        key, value = line.split(': ')
                        key = key.strip()
                        value = value.strip()
                        canal_nodes_dict[key] = int(value)
                    elif current_var == 'D_w':
                        key, value = line.split(': ')
                        key = key.strip()
                        value = value.strip()
                        D_w_init_sw[key] = value
                    elif current_var == 'v_d':
                        key, value = line.split(': ')
                        value = value.replace("'", "")
                        v_d[key] = int(value)
                    else:
                        key_parts = line.split('(')[1].split(')')[0].split(', ')
                        key_parts = [part.strip("'") for part in key_parts]
                        indices = tuple(key_parts)
                        value = line.split(': ')[-1]
                        if current_var == 'X_W':
                            X_W_init_sw[indices] = float(value)
                        elif current_var == 'Q_W':
                            Q_W_init_sw[indices] = float(value)
                        elif current_var == 'L_W':
                            L_W_init_sw[indices] = float(value)
                        elif current_var == 'Z_WV':
                            Z_WV_init_sw[indices] = float(value)
                        elif current_var == 'Y':
                            Y_init_sw[indices] = float(value)
                        elif current_var == 'B':
                            B_init_sw[indices] = float(value)
                        elif current_var == 'A_WV':
                            A_WV_init_sw[indices] = float(value)
                        elif current_var == 'A_DD':
                            A_DD_init_sw[indices] = float(value)
                        elif current_var == 'S':
                            S_init_sw[indices] = float(value)
                        elif current_var == 'LS':
                            LS_init_sw[indices] = float(value)
                        elif current_var == 'D':
                            D_init_sw[indices] = float(value)
                        elif current_var == 'D_WV':
                            D_WV_init_sw[indices] = float(value)
                        elif current_var == 'W':
                            W_init_sw[indices] = float(value)
                        elif current_var == 'P_V':
                            P_V[indices] = float(value)
                        elif current_var == 'Z_V':
                            Z_V[indices] = float(value)
                        elif current_var == 'D_T':
                            D_T[indices] = float(value)
                        elif current_var == 'L_V':
                            L_V[indices] = float(value)
                        elif current_var == 'T_V':
                            T_V_new_init_sw[indices] = float(value)
                        elif current_var == 'A_R':
                            A_R_init_sw[indices] = float(value)
    zero = ['zero']
    WV_id = W_id + V_id
    WV0_id = zero + WV_id
    W0_id = zero + W_id
    V0_id = zero + V_id
    DS_id = DC + S_id


    Nr_vessels = np.arange(1,len(W_id) + 1)
    F = []
    for n in Nr_vessels:
        f = [f'Vessel_{n}']
        F = F + f

    T_W_init_sw = {}
```

```
    for f in F:
        for l in W0_id:
            for k in W0_id:
                T_W_init_sw[l,k,f] = 0

    numb = 0
    for w in W_id:
        T_W_init_sw['zero',w,F[numb]] = 1
        T_W_init_sw[w,'zero',F[numb]] = 1
        numb += 1

    # Determine closest DC for each satellite
    DC_S = {}
    for s in S_id:
        dist_DC = 999999
        for dc in DC:
            dist_s_DC = dist_fe.at[canal_nodes_dict[dc], canal_nodes_dict[s]]
            if dist_s_DC < dist_DC:
                dist_DC = dist_s_DC
                dc_s = dc
        DC_S[s] = dc_s


    S_DC = {'DC_1': [], 'DC_2': [], 'DC_3': []}

    for s in S_id:
        dc = DC_S.get(s)
        if dc in S_DC:
            S_DC[dc].append(s)


    #%% Schedule water vehicles
    print('Working on water scheduling for Ns:', Ns)
    start_sched_w = time.time()
    model = gb.Model('Scheduling_water')
    np.random.seed(123)
    time_limit = t_lim_sched_water
    K = 9999

    X_W =  X_W_init_sw
    Q_W = Q_W_init_sw

    total_load = 0
    for i in S_id:
            total_load += LS_V[i]
            load_delivered = 0
            load_delivered = quicksum(Q_W[i,w]  for w in W_id)
            print(i,'load delivered by w:', load_delivered, 'load required by v:',LS_V[i])
    print('total load required: ', total_load)

    DC_W = {}
    for w in W_id:
        for d in DC:
            for i in S_id:
                if round(X_W_init_sw[i,d,w]) == 1:
                    DC_W[w] = d

    # Binary variable, Y[i,k,l] = 1 if both k and l visit i
    Y = {}
    for k in WV_id:
        for l in WV_id:
            for i in DS_id:
                Y[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'Y')

    # Arrival time of vehicle r at i
    A_WV = {}
    for i in DS_id:
        for k in WV_id:
            A_WV[i,k] = model.addVar(lb = -500, ub = 999999, vtype = GRB.CONTINUOUS, name = '
    A_WV')

```

```
    # Difference in arrival times of vehicle
    A_D = {}
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                A_D[i,k,l] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'A_D')

    # Difference in arrival times of vehicle
    A_DD = {}
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                A_DD[i,k,l] = model.addVar(lb = -999999, ub = 999999, vtype = GRB.CONTINUOUS,
     name = 'A_DD')

    # Customer or satellite is visited by vehicle r: = 1, if not: = 0
    Z_WV = {}
    for k in WV_id:
        for i in DS_id:
            Z_WV[i,k] =  model.addVar(vtype = GRB.BINARY, name = 'Z_WV')

    # Accumulated load of road vehicle r at customer i
    L_W = {}
    for w in WV_id:
        for i in DS_id:
            L_W[i,w] =  model.addVar(lb=0.0, vtype = GRB.CONTINUOUS, name = 'L_W')


    # Accumulated load delivered to satellite i by vehicles before and including k
    LS = {}
    for i in S_id:
        for k in WV0_id:
            LS[i,k] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'LS')

    # Number of water vehicles used
    Nw = {}
    for w in W_id:
        Nw[w] = model.addVar(vtype = GRB.BINARY, name = 'Nw')

    # Stock at satellite i after arrival of vehicle k
    S = {}
    for i in S_id:
        for k in WV_id:
            S[i,k] = model.addVar(lb = -100, vtype = GRB.INTEGER, name = 'S')

    B = {}
    for i in S_id:
        for k in WV0_id:
            for l in WV0_id:
                B[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'B')


    # New for scheduling
    D_w = {}
    for w in W0_id:
        D_w[w] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'D_w')


    # Scheduling water vehicles
    T_W = {}
    for f in F:
        for k in W0_id:
            for l in W0_id:
                T_W[k,l,f] = model.addVar(vtype = GRB.BINARY, name = 'T_W')

    A_F = {}
    for f in F:
        for w in W0_id:
            A_F[w,f] = model.addVar(lb = -500, vtype = GRB.CONTINUOUS, name = 'A_F')

    N_F = {}
```

```
    for f in F:
        N_F[f] = model.addVar(vtype = GRB.BINARY, name = 'N_F')

    P_W = {}
    for k in W0_id:
        for l in W0_id:
            P_W[l,k] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'P_W')

    # New for departure times
    D = {}
    for i in S_id:
        for k in WV0_id:
            for l in WV0_id:
                D[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'D')


    D_WV = {}
    for i in DS_id:
        for k in WV_id:
            D_WV[i,k] = model.addVar(lb = 0.0, ub = 999999, vtype = GRB.CONTINUOUS, name = 'D_WV')

    W = {}
    for i in DS_id:
        for w in W0_id:
            W[i,w] = model.addVar(vtype = GRB.CONTINUOUS, name = 'W')

    D_max = {}
    for l in W_id:
        D_max[l] = model.addVar(vtype = GRB.CONTINUOUS, name = 'D_max')

    Z_FW = {}
    for f in F:
        for w in W_id:
            Z_FW[w,f] = model.addVar(vtype = GRB.BINARY, name = 'Z_FW')


    #  Objective function
    model.setObjective(quicksum(N_F[f] for f in F))

    model.modelSense = GRB.MINIMIZE
    model.update()

    # Constraints
    # 3. Nodes that are visited by vehicle w
    for w in W_id:
        for i in DS_id:
            constr_w_3b = model.addConstr(Z_WV[i,w] == quicksum(X_W[i,j,w] for j in DS_id), name='Constr_3')

    # 4b. Nodes that are visited by vehicle r
    for v in V_id:
        for i in DS_id:
            constr_w_4c = model.addConstr(Z_WV[i,v] == Z_V[i,v], name='Constr_4')

    # New
    # 5. The demand delivered to i is zero if vehicle r does not visit i
    for w in W_id:
        for i in DS_id:
            constr_w_5 = model.addGenConstrIndicator (Z_WV[i,w], False, Q_W[i,w], GRB.EQUAL, 0, name='Constr_5')

    # 6. Demand satisfaction constraint
    for i in S_id:
            constr_w_6 = model.addConstr(quicksum(Q_W[i,w] for w in W_id) == LS_V[i], name='Constr_6') #s_v[i])) #
            constr_w_6b = model.addConstr(Q_W[i,'zero'] == 0, name='Constr_6b')

    # New
    # 7. No load is delivered to DC
    # 8. The accumulated load at the DC is zero
```

```
    for w in W_id:
        DC = DC if isinstance(DC, list) else [DC]
        for i in DC:
            constr_w_7 = model.addConstr (Q_W[i,w] == 0, name='Constr_7')
            constr_w_8 = model.addConstr(L_W[i,w] == 0, name='Constr_8')

    # 8b. No load delivered by road vehicles
    for v in V_id:
        for i in DS_id:
            constr_w_8b = model.addConstr(Q_W[i,v] == 0, name='Constr_8b')
            constr_w_8c = model.addConstr(L_W[i,v] == 0, name='Constr_8c')

    # 9a_new. With X_W as an input, the constraint can be rewritten as:
    for w in W_id:
        for i in DS_id:
            for j in S_id:
                if X_W[i,j,w] == 1:
                    constr_9a_new = model.addConstr(L_W[j,w] - L_W[i,w] - Q_W[j,w] == 0, name = 'Constr_9a_new')

    # 9b. No L_R if not visited
    for w in W_id:
        for i in DS_id:
            constr_w_9b = model.addGenConstrIndicator (Z_WV[i,w], False, L_W[i,w], GRB.EQUAL, 0, name='Constr_9b')

    # New
    # 9c. The load delivered to customer i by vehicle r is always less than or equal to the accumulated load of r at customer i:
    for w in W_id:
        for i in S_id:
            constr_w_9c = model.addConstr(Q_W[i,w] <= L_W[i,w], name='Constr_9c')

    # New
    # 9d. The accumulated load of vehicle r at customer i is always less than or equal to the maximum capacity of vehicle r:
    for w in W_id:
        for i in S_id:
            constr_w_9d = model.addConstr( L_W[i,w] <= capacity_fe, name='Constr_9d')


    # # Arrival time constraints:
    # 10_new. With X_W as input
    for w in W_id:
        for i in DS_id:
            for j in S_id:
                if X_W[i,j,w] == 1:
                    constr_10_new = model.addConstr(A_WV[j,w] - A_WV[i,w] - dist_fe.at[canal_nodes_dict[i],canal_nodes_dict[j]] / (speed_fe * 60) - W[i,w] - Q_W[i,w] * 0.2 >= 0, name='Constr_10_new' )

    for w in W_id:
        for j in S_id:
            DC = DC if isinstance(DC, list) else [DC]
            for i in DC:
                if X_W[i,j,w] == 1:
                    constr_time_10b = model.addConstr(A_WV[j, w] - A_WV[i, w] - dist_fe.at[canal_nodes_dict[i], canal_nodes_dict[j]] / (speed_fe * 60) - service_time_fe / 60 == 0, name='Constr_10b')


    # 11. Binary variable Y[i,k,l] is one if both k and l visit i
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                if k != l:
                    constr_Y_11 = model.addConstr(Y[i,k,l] == gb.and_(Z_WV[i,k], Z_WV[i,l]), name='Constr_11')

    # 12. Arrival times of vehicles at satellites cannot be the same
    for i in S_id:
```

```
        for k in WV_id:
            for l in WV_id:
                constr_time_12a = model.addConstr(A_DD[i,k,l] == A_WV[i,k] - A_WV[i,l], name='Constr_12a')
                constr_time_12b = model.addConstr(A_D[i,k,l] == gb.abs_(A_DD[i,k,l]), name='Constr_12b')

    # 13a. Arrival times of road vehicles at satellites cannot be the same
    for i in S_id:
        for k in V_id:
            for l in V_id:
                if k != l:
                    constr_time_13a = model.addGenConstrIndicator(Y[i,k,l], True, A_D[i,k,l], GRB.GREATER_EQUAL, transship_s, name='Constr_13a') #180)
                    constr_time_13a_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13a_1')

    # 13b. Arrival times of a water vehicles is later than the departure time of another water vehicle
    for i in S_id:
        for k in W_id:
            for l in W_id:
                if k != l:
                    constr_time_13b = model.addGenConstrIndicator(B[i,k,l], True, A_WV[i,k] - D_WV[i,l], GRB.GREATER_EQUAL, 0, name='Constr_13b') #600)
                    constr_time_13b_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13b_1')

    # 13b. Arrival times of water and road vehicles at satellites cannot be the same
    for i in S_id:
        for k in W_id:
            for l in V_id:
                if k != l:
                    constr_time_13c = model.addGenConstrIndicator(Y[i,k,l], True, A_D[i,k,l], GRB.GREATER_EQUAL, 0.01, name='Constr_13c') #600)
                    constr_time_13c_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13c_1')

    #13c. Arrival times at satellites cannot be later than the maximum time span
    for i in S_id:
        for k in WV_id:
            constr_time_13d = model.addConstr(D_WV[i,k] <= time_span, name='Constr_13d')


    # 14. Arrival time is infinite if a vehicle does not visit satellite i
    for i in S_id:
        for k in WV_id:
            constr_time_14 = model.addGenConstrIndicator(Z_WV[i,k], False, A_WV[i,k], GRB.EQUAL, 0, name = 'Constr_14')

    # # Satellite synchronisation constraints:

    # 15. Binary variable = 1 if vehicle k arrives at the same time or after vehicle l
    for i in S_id:
        for k in WV_id:
            constr_binary_150 = model.addGenConstrIndicator(Z_WV[i,k], True, B[i,k,'zero'], GRB.EQUAL, 1, name = 'Constr_150' )
            for l in WV_id:
                constr_binary_15a = model.addGenConstrIndicator(Y[i,k,l], True, A_WV[i,k] - K * B[i,k,l] - A_WV[i,l], GRB.LESS_EQUAL, 0, name = 'Constr_15a' )
                constr_binary_15b = model.addGenConstrIndicator(Y[i,k,l], True, B[i,k,l] + B[i,l,k], GRB.EQUAL, 1, name = 'Constr_15b')
                constr_binary_15c = model.addConstr(B[i,k,l] + B[i,l,k] <= 1)
                constr_binary_15d = model.addGenConstrIndicator(Z_WV[i,k], False, B[i,k,l], GRB.EQUAL, 0, name = 'Constr_15d' )
                constr_binary_15e = model.addGenConstrIndicator(Z_WV[i,k], False, B[i,l,k], GRB.EQUAL, 0, name = 'Constr_15e' )

    # 16. Load delivered to satellite i by all vehicles before k and k
    for i in S_id:
        for k in WV0_id:
```

```
            for l in WV0_id:
                    constr_load_16a = model.addGenConstrIndicator(B[i,k,l], True, LS[i,k] - LS[i,l] - Q_W[i,k], GRB.GREATER_EQUAL,0, name = 'Constr_16a')

    for i in S_id:
        for k in WV_id:
            constr_load_16b = model.addConstr(LS[i,k] <= quicksum(Q_W[i,w] for w in W_id), name = 'Constr_16b')
            constr_load_16c = model.addGenConstrIndicator(Z_WV[i,k], False, LS[i,k], GRB.EQUAL, 0, name = 'Constr_16c' )

    # 17. New Stock at satellites constraints
    for i in S_id:
        for k in WV_id:
            # for l in WV_id:
                constr_stock_17a = model.addGenConstrIndicator (Z_WV[i,k], True, S[i,k] + quicksum(L_V[i,l] * B[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.EQUAL, 0, name = 'Constr_17a')

    for i in S_id:
        for k in WV_id:
            constr_stock_17b = model.addConstr(S[i,k] >= 0, name = 'Constr_17b')
            constr_stock_17c = model.addConstr(S[i,k] <= capacity_s[i] + capacity_fe, name = 'Constr_17c')

    for w in W_id:
        constr_water_km = model.addConstr(D_w[w] == quicksum(dist_fe.at[canal_nodes_dict[i], canal_nodes_dict[j]] * X_W[i,j,w] for i in DS_id for j in DS_id), name = 'Constr_water_km')


    for w in W_id:
        for i in S_id:
            constr_Nw = model.addGenConstrIndicator(Z_WV[i,w], True, Nw[w], GRB.EQUAL, 1, name = 'Constr_Nw')

    constr_trips_performed = model.addConstr(quicksum(T_W[i,j,f] for i in W_id for j in W0_id for f in F) == len(W_id), name = 'constr_trips')


    for k in WV_id:
        for i in S_id:
            constr_time_span = model.addConstr(A_WV[i,k] >= 0, name = 'constr_time_span')

    for r in R_v:
        for k in V_id:
            for l in V_id:
                    if k!= l:
                        if T_V_new_init_sw[l,k,r] == 1:
                            constr_relax_init = model.addConstr(A_WV[v_s[k],k] - A_WV[v_s[l],l] - P_V[l,k]  >= 0, name = 'Constr_relax_init')

    # New for multiple water vehicle depots:
    for l in W_id:
        for k in W_id:
            constr_21 = model.addConstr(P_W[l,k] == D_w[l]/(speed_fe*60) + (dist_fe.at[canal_nodes_dict[DC_W[l]],canal_nodes_dict[DC_W[k]]] / (speed_fe *60)) + service_time_fe / 60 + quicksum(Z_WV[i,l] * W[i,l] for i in S_id), name = 'Constr_21' )#quicksum(Z_WV[i,l] for i in S_id) * (transship_s / 60) + service_time_fe / 60)

    # Each vehicle f can only leave the depot once
    for f in F:
        constr_22f = model.addConstr(quicksum(T_W['zero',k,f] for k in W_id) <= 1, name = 'Constr_22f')

    # Each trip is performed once
    for k in W_id:
        constr_22b = model.addConstr(quicksum(T_W[l,k,f] for l in W0_id for f in F) == 1, name = 'Constr_22b')

```

```
    #A vessel can only perform trips in the same neighbourhood:
    for f in F:
        for k in W_id:
            depot_k = DC_W[k]
            for l in W_id:
                if DC_W[l] != depot_k:
                    constr_neighbour = model.addGenConstrIndicator(Z_FW[k,f], True, Z_FW[l,f], GRB.EQUAL, 0, name = 'Constr_neighbour' )
                    constr_neighbour_b = model.addGenConstrIndicator(Z_FW[k,f], True, T_W[l,k,f] + T_W[k,l,f], GRB.EQUAL, 0, name = 'Constr_neighbour_b' )


    # Z_FW = 1 if f performs trip w
    for k in W_id:
        for f in F:
            constr_22b_1 = model.addConstr(Z_FW[k,f] == quicksum(T_W[l,k,f] for l in W0_id), name = 'Constr_22b_1')

    for f in F:
        for l in W_id:
            constr_22b_2 = model.addGenConstrIndicator(Z_FW[l,f], True, quicksum(T_W['zero',k,f] for k in W_id), GRB.EQUAL, 1, name = 'Constr_22b_2')

    for l in W_id:
        constr_22c = model.addConstr(D_max[l] == gb.max_(D_WV[i,l] for i in S_id), name = 'Constr_22c') # gb.max_(D_WV[i,l] for i in S_id)

    # New: Trip k can be performed by vehicle f if the start time of trip k is later than the end of trip l
    for f in F:
        for k in W_id:
            for l in W_id:
                constr_22c_new = model.addGenConstrIndicator(T_W[l,k,f], True, A_F[k,f] - D_max[l] - quicksum(X_W[i,d,l] * dist_fe.at[canal_nodes_dict[i],canal_nodes_dict[d]] for i in S_id for d in DC) / (speed_fe * 60), GRB.GREATER_EQUAL, 0, name = 'Constr_22c_new')

    # A trip can never be performed after itself
    for f in F:
        for l in W0_id:
            constr_22d = model.addConstr(T_W[l,l,f] == 0, name = 'Constr_22d')

    # Vehicle f can only end trip l if it also started it
    for f in F:
        for l in W0_id:
            # if i != j:
                constr_22e = model.addConstr(quicksum(T_W[l,k,f] for k in W0_id) == quicksum(T_W[k,l,f] for k in W0_id), name = 'Constr_22e')

    # Number of water vehicles used
    for f in F:
        for k in W_id:
            constr_23 = model.addGenConstrIndicator(T_W['zero',k,f], True, N_F[f], GRB.EQUAL, 1, name = 'Constr_23')

    # Connect A_F with A_WV
    for w in W_id:
        constr_24 = model.addConstr(A_WV[DC_W[w],w] == quicksum(quicksum(T_W[k,w,f] for k in W0_id ) * A_F[w,f] for f in F), name = 'Constr_24')


    # New for departure times
    for i in S_id:
        for v in V_id:
            constr_departure_1 = model.addGenConstrIndicator(Z_WV[i,v], True, D_WV[i,v] - A_WV[i,v] - tranship_s / 60, GRB.EQUAL, 0, name = 'constr_dep_1')
        for w in W_id:
            constr_departure_2 = model.addGenConstrIndicator(Z_WV[i,w], True, D_WV[i,w] - A_WV[i,w] - W[i,w] - Q_W[i,w] * 0.2, GRB.EQUAL, 0, name = 'constr_dep_2' )


    for i in S_id:
```

```
    for k in V0_id:
        for l in V0_id:
            constr_departure_3 = model.addGenConstrIndicator(B[i,k,l], True, D[i,k,l], GRB.EQUAL, 1, name = 'constr_dep_3')
    for k in W0_id:
        for l in W0_id:
            constr_departure_4 = model.addGenConstrIndicator(B[i,k,l], True, D[i,k,l], GRB.EQUAL, 1, name = 'constr_dep_4')

    for k in WV_id:
        constr_departure_8 =  model.addGenConstrIndicator (Z_WV[i,k], False, quicksum(D[i,k,l] for l in WV_id) + quicksum(D[i,l,k] for l in WV_id), GRB.EQUAL, 0, name = 'constr_dep_8')
        for l in WV_id:
            constr_departure_5 = model.addGenConstrIndicator(Y[i,k,l], True, D_WV[i,k] - K* D[i,k,l] - D_WV[i,l], GRB.LESS_EQUAL, 0, name = 'constr_dep_5')
            constr_departure_6 = model.addGenConstrIndicator(Y[i,k,l], True, D[i,k,l] + D[i,l,k], GRB.EQUAL, 1, name = 'constr_dep_6')


for i in S_id:
    for k in W_id:
        constr_departure_7 = model.addGenConstrIndicator (Z_WV[i,k], True, quicksum(L_V[i,l] * D[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.LESS_EQUAL, 0, name = 'constr_dep_7')
        constr_departure_7_b = model.addGenConstrIndicator (Z_WV[i,k], True, quicksum(L_V[i,l] * D[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.GREATER_EQUAL, - capacity_s[i], name = 'constr_dep_7_b')


for (i,w), value in L_W_init_sw.items():
    L_W[i,w].start = value

for (i,w), value in Z_WV_init_sw.items():
    Z_WV[i,w].start = value

for (i,k,l), value in Y_init_sw.items():
    Y[i,k,l].start = value

for (i,k,l), value in B_init_sw.items():
    B[i,k,l].start = value

for (i, k), value in A_WV_init_sw.items():
    A_WV[i,k].start = value

for (i,k,l), value in A_DD_init_sw.items():
    A_DD[i,k,l].start = value

for (i,w), value in S_init_sw.items():
    S[i,w].start = value

for (i,k), value in LS_init_sw.items():
    LS[i,k].start = value

for (i,k,l), value in D_init_sw.items():
    D[i,k,l].start = value

for (i, k), value in D_WV_init_sw.items():
    D_WV[i,k].start = value

for (i, k), value in W_init_sw.items():
    W[i,k].start = value

for (i,k,l), value in T_W_init_sw.items():
    T_W[i,k,l].start = value


# Start optimisation

print("start optimizing")
model.setParam( 'OutputFlag', True)
```

```
    model.setParam ('MIPGap', mip_sched_w)
    model.setParam('FeasibilityTol', 1e-4)
    model.setParam('MIPFocus', 0)
    model.setParam('SubMIPNodes', 20000)
    model.setParam('Seed', 123)
    model.setParam('SoftMemLimit', 100)
    model.setParam('Threads', 40)
    if time_limit:
        model.setParam('Timelimit', time_limit)
    model._obj = None
    model._bd = None
    model._obj_value = []
    model._time = []
    model._start = time.time()
    model.optimize()
    mip_gap_water = model.MIPGap
    end_sched_w = time.time()
    time_sched_w = end_sched_w - start_sched_w

    #%% Save solutions
    water_km_w = 0
    for w in W_id:
        water_km_w += D_w[w].X


    W_used_w = []
    for w in W_id:
        w_visits = 0
        for i in S_id:
            if Z_WV[i,w].X == 1:
                    w_visits += 1
        if w_visits >= 1:
            W_used_w.append(w)
    Nr_w_w = len(W_used_w)

    f_used_water = 0
    for f in F:
        if N_F[f].X == 1:
            f_used_water += 1
    print (f_used_water)
    Nr_F_w = f_used_water


    max_start_F_ = 0
    for f in F:
        for w in W_id:
            if A_F[w,f].X > max_start_F_:
                max_start_F_ = A_F[w,f].X

    max_start_F_w = max_start_F_
    max_start = 0
    for v in V_id:
        for i in S_id:
            if A_WV[i,v].X > max_start:
                max_start = A_WV[i,v].X
    max_start_V_w = max_start

    F_new = []                    # Make set of water vehicles F the size used in scheduling water vehicles
    for f in F:
        if N_F[f].X == 1:
            F_new.append(f)
    F = F_new.copy()
    F_sched_w = len(F)
    print('Distance on waterways after water scheduling: ', water_km_w)

    X_W_init_wf = X_W
    Y_init_wf = model.getAttr('X', Y)
    A_WV_init_wf = model.getAttr('X', A_WV)
    A_D_init_wf = model.getAttr('X', A_D)
    A_DD_init_wf = model.getAttr('X', A_DD)
```

```
    Q_W_init_wf = Q_W
    Z_WV_init_wf = model.getAttr('X', Z_WV)
    L_W_init_wf = model.getAttr('X', L_W)
    LS_init_wf = model.getAttr('X', LS)
    S_init_wf = model.getAttr('X', S)
    B_init_wf = model.getAttr('X', B)
    A_F_init_wf = model.getAttr('X', A_F)
    T_W_init_wf = model.getAttr('X', T_W)
    T_V_init_wf = T_V_new_init_sw                    # T_V initial solution for total scheduling is still the found solution in road scheduling
    D_init_wf = model.getAttr('X', D)
    D_WV_init_wf = model.getAttr('X', D_WV)
    W_init_wf = model.getAttr('X', W)

    T_W_new_init_wf = {}

    for f in F:
        for l in W0_id:
            for k in W0_id:
                T_W_new_init_wf[k,l,f] = T_W_init_wf[k,l,f]

    #%%
    with open(f'output_water_{save_title}_{Ns}_{t_lim_VRP_E2}.txt', 'w') as f:
        f.write(f'D_r_VRP_E2:\n{D_r_VRP_E2}\n')
        f.write(f'MIP_VRP_E2:\n{MIP_VRP_E2}\n')
        f.write(f'D_w_VRP_E1:\n{D_w_VRP_E1}\n')
        f.write(f'MIP_VRP_E1:\n{MIP_VRP_E1}\n')
        f.write(f'R_sched_r:\n{R_sched_r}\n')
        f.write(f'D_r_r:\n{D_r_r}\n')
        f.write(f'MIP_sched_r:\n{MIP_sched_r}\n')
        f.write(f'F_sched_w:\n{F_sched_w}\n')
        f.write(f'D_w_w:\n{water_km_w}\n')
        f.write(f'MIP_sched_w:\n{mip_gap_water}\n')
        for var_name, var_values in [
            ('X_W', X_W_init_wf),
            ('Y', Y_init_wf),
            ('A_WV', A_WV_init_wf),
            ('A_D', A_D_init_wf),
            ('A_DD',A_DD_init_wf),
            ('Q_W', Q_W_init_wf),
            ('Z_WV', Z_WV_init_wf),
            ('L_W', L_W_init_wf),
            ('LS', LS_init_wf),
            ('S', S_init_wf),
            ('B', B_init_wf),
            ('D', D_init_wf),
            ('D_WV', D_WV_init_wf),
            ('W', W_init_wf),
            ('T_V', T_V_init_wf),
            ('A_F', A_F_init_wf),
            ('T_W', T_W_new_init_wf),
            ('P_V', P_V),
            ('L_V', L_V),
            ('LS_V', LS_V),
            ('Z_V', Z_V),
            ('v_s', v_s),
            ('L_V', L_V),
            ('D_T', D_T),
            ('v_d', v_d),
            ('canal_nodes_dict', canal_nodes_dict)
        ]:
            f.write(f'{var_name}:\n')
            for key, value in var_values.items():
                if isinstance(value, gb.LinExpr):
                    value = value.getValue()
                f.write(f'  {key}: {value}\n')
        f.write('V_id:\n')
        for v in V_id:
            f.write(f'{v}\n')
        f.write('W_id:\n')
        for w in W_id:
```

```
            f.write(f'{w}\n')
        f.write('S_id:\n')
        for s in S_id:
            f.write(f'{s}\n')
        f.write('R_v:\n')
        for r in R_v:
            f.write(f'{r}\n')
        f.write('F:\n')
        for k in F:
            f.write(f'{k}\n')
            print('F is written')
        f.write('DC:\n')
        for d in DC:
            f.write(f'{d}\n')
```

B.4.3. Integrated Scheduling

```
#%% Import libraries
import gurobipy as gb
import time
import os
import numpy as np
import pandas as pd
import pickle
import math
import copy
import sys
import matplotlib.pyplot as plt
import warnings
from openpyxl import load_workbook
from gurobipy import quicksum, GRB

#%% Set path
server = 'True'

if server == 'False':
    path = os.getcwd() + "\Inputs\\"
    path_out = os.getcwd() + "\Outputs\\"
    from FLP_solver_definition_number_customers_horeca_sets_Laudy import FLP_num_cust
    from FLP_solver_definition_horeca_sets_capacity_assignment import FLP_capacity

if server == 'True':
    path = os.getcwd() + "/Inputs/"
    path_out = os.getcwd() + "/Outputs/"


#%% Scenario inputs
directed = 'true'                # Indicate wether to use directed or undirected distance
    matrix
FLP_constraint = 'num_cust'      # Which FLP constraint to use, either capacity or num_cust
Nc = 750                         # Insert the number of customers to consider
horeca_sets = np.arange(1,11)    # Which horeca sets to evaluate
horeca_set = 1                   # If not testing all horeca sets, insert one to evaluate


#%% Import network and scenario data
df_horeca_demand_scenarios = pd.read_excel(path + f'df_horeca_demand_scenarios.xlsx',
    index_col=0)
df_horeca_demand_scenarios.index = df_horeca_demand_scenarios.index.astype(str)
df_horeca_data_info = pd.read_excel(path + f'df_horeca_data_info.xlsx', index_col=0)
df_horeca_data_info.index = df_horeca_data_info.index.astype(str)
customer_locations = df_horeca_data_info.iloc[:,0]

if server == 'False':
    df_SE_shortest_dist_directed_False = pickle.load(open(path + '
    df_SE_shortest_dist_directed-False_nodes_all.pickle', 'rb'))
    df_SE_shortest_dist_directed_True_1 = pickle.load(open(path + '
    df_SE_shortest_dist_directed-True_nodes_all.pickle', 'rb'))
    df_SE_shortest_dist_directed_True = df_SE_shortest_dist_directed_True_1.fillna(1001)
    dict_FE_shortest_dist_directed_True_1 = pickle.load(open(path + '
    dict_FE_shortest_dist_directed-True_nodes_all.pickle', 'rb'))
```

```python

if server == 'True':
    pickle_off = open(path + 'df_SE_shortest_dist_directed-True_nodes_all.pickle', 'rb')
    df_SE_shortest_dist_directed_True_1 = pd.read_pickle(pickle_off)
    df_SE_shortest_dist_directed_True = df_SE_shortest_dist_directed_True_1.fillna(1001)

    pickle_off = open(path + 'dict_FE_shortest_dist_directed-True_nodes_all.pickle', 'rb')
    dict_FE_shortest_dist_directed_True_1 = pd.read_pickle(pickle_off)
    df_SE_shortest_dist_directed_False = dict_FE_shortest_dist_directed_True_1
assigned = []
indices = []
customers = [[0]*3]*len(customer_locations)
for customer_id in df_horeca_data_info.index.tolist():
    if df_horeca_demand_scenarios.at[f'{customer_id}', f'set_{horeca_set}'] > 0:
        indices.append(customer_id)
        assigned.append({'road_node':int(df_horeca_data_info.at[customer_id, 'road_node']), '
    demand':int(df_horeca_demand_scenarios.at[f'{customer_id}', f'set_{horeca_set}'])} )
customers = pd.DataFrame(assigned, index= indices)

satellite_locations = pd.read_excel(path + "satellite_nodes_storage_full.xlsx", index_col=0)
vehicles = pd.read_excel(path  +"Road_vehicles.xlsx", index_col=0)
road_nodes = pd.read_excel(path + "satellites_customers_road_nodes.xlsx", index_col = 0)

if directed == 'true':
    dist = df_SE_shortest_dist_directed_True
elif directed == 'false':
    dist = df_SE_shortest_dist_directed_False

#%% Parameters
speed_v = int(os.getenv('speed_v'))
transship_s = int(os.getenv('transship_s'))
transship_c = int(os.getenv('transship_c'))
fev_profile = 5
capacity_fe = int(os.getenv('capacity_fe'))
speed_fe_str =  os.getenv('speed_fe')
speed_fe = float(speed_fe_str)
service_time_fe = int(os.getenv('service_time_fe'))
capacity_s = int(os.getenv('capacity_s'))
capacity_se = int(os.getenv('capacity_se'))
storage_set = os.getenv('storage_set')
df_fe_distance_matrix = dict_FE_shortest_dist_directed_True_1[f'vessel_profile_{fev_profile}'
    ].copy()
dist_fe = df_fe_distance_matrix.fillna(99999)

# New distance matrix for multiple water vehicle depots:
dict_FE_new = pd.read_csv(path + 'distance_matrix_DCs.csv',sep=';',header=None)
dist_fe_new = pd.DataFrame(dict_FE_new)
dist_fe_new.index = dist_fe_new.index + 1
new_index = {old_index:old_index + 1 for old_index in dist_fe_new.columns}
dist_fe_new = dist_fe_new.rename(columns=new_index)
dist_fe = dist_fe_new.fillna(99999)


t_limits_VRP_E2_str = os.getenv('t_limits_VRP_E2')
t_limits_VRP_E2 = eval(t_limits_VRP_E2_str)

t_lim_VRP_E1 = int(os.getenv('t_lim_VRP_E1'))
t_lim_sched_road = int(os.getenv('t_lim_sched_road'))
t_lim_sched_water = int(os.getenv('t_lim_sched_water'))
t_lim_sched_total = int(os.getenv('t_lim_sched_total'))
time_span = int(os.getenv('time_span'))
mip_VRP_E2_str = os.getenv('mip_VRP_E2')
mip_VRP_E2 = float(mip_VRP_E2_str)
mip_VRP_E1_str = os.getenv('mip_VRP_E1')
mip_VRP_E1 = float(mip_VRP_E1_str)
mip_sched_r_str = os.getenv('mip_sched_r')
mip_sched_r = float(mip_sched_r_str)
mip_sched_w_str = os.getenv('mip_sched_w')
mip_sched_w = float(mip_sched_w_str)
mip_sched_t_str = os.getenv('mip_sched_t')
mip_sched_t = float(mip_sched_t_str)
```

```
save_title = os.getenv('save_title')
Ns = int(os.getenv('NrSatellites'))

capacity_s = {}
for i in satellite_locations.index.tolist():
    capacity_s[i] = satellite_locations.at[i,f'capacity_{storage_set}']


#%% Import initial solution
N_s = []
results = []
for t_lim_VRP_E2 in t_limits_VRP_E2:
    V_id = []
    W_id = []
    S_id = []
    R_v = []
    F = []
    X_W_init_wf = {}
    Q_W_init_wf = {}
    L_W_init_wf = {}
    Z_WV_init_wf = {}
    Y_init_wf = {}
    B_init_wf = {}
    A_WV_init_wf = {}
    A_DD_init_wf = {}
    S_init_wf = {}
    LS_init_wf = {}
    T_V_init_wf = {}
    T_W_new_init_wf = {}
    D_init_wf = {}
    D_WV_init_wf = {}
    W_init_wf = {}
    A_F_init_wf = {}

    P_V = {}
    LS_V = {}
    Z_V = {}
    v_s = {}
    L_V = {}
    D_T = {}
    v_d = {}
    D_r_VRP_E2 = None
    D_w_VRP_E1 = None
    D_r_r = None
    D_w_w = None
    MIP_VRP_E2 = None
    MIP_VRP_E1 = None
    MIP_sched_r = None
    MIP_sched_w = None
    R_sched_r = None
    F_sched_w = None
    canal_nodes_dict = {}
    DC = []
    with open(f'output_water_{save_title}_{Ns}_{t_lim_VRP_E2}.txt', 'r') as f:
        current_var = None
        for line in f:
            line = line.strip()
            if line.endswith(':'):
                current_var = line[:-1]
            elif current_var is not None:
                parts = line.split(': ')
                if current_var == 'V_id':
                    V_id.append(line.strip())
                elif current_var == 'W_id':
                    W_id.append(line.strip())
                elif current_var == 'S_id':
                    S_id.append(line.strip())
                elif current_var == 'R_v':
                    R_v.append(line.strip())
                elif current_var == 'F':
                    F.append(line.strip())
```

```
            elif current_var == 'DC':
                DC.append(line.strip())
            elif current_var == 'D_r_VRP_E2':
                D_r_VRP_E2 = float(line)
            elif current_var == 'MIP_VRP_E2':
                MIP_VRP_E2 = float(line)
            elif current_var == 'D_w_VRP_E1':
                D_w_VRP_E1 = float(line)
            elif current_var == 'MIP_VRP_E1':
                MIP_VRP_E1 = float(line)
            elif current_var == 'D_w_VRP_E1':
                D_w_VRP_E1 = float(line)
            elif current_var == 'R_sched_r':
                R_sched_r = float(line)
            elif current_var == 'D_r_r':
                D_r_r = float(line)
            elif current_var == 'MIP_sched_r':
                MIP_sched_r = float(line)
            elif current_var == 'F_sched_w':
                F_sched_w = float(line)
            elif current_var == 'D_w_w':
                D_w_w = float(line)
            elif current_var == 'MIP_sched_w':
                MIP_sched_w = float(line)
            elif len(parts) == 2:
                key, value = parts
                if current_var == 'LS_V':
                    key, value = line.split(': ')
                    key = key.strip()
                    value = value.strip()
                    LS_V[key] = int(value)
                elif current_var == 'v_s':
                    key, value = line.split(': ')
                    key = key.strip()
                    value = value.strip()
                    v_s[key] = value
                elif current_var == 'canal_nodes_dict':
                    key, value = line.split(': ')
                    key = key.strip()
                    value = value.strip()
                    canal_nodes_dict[key] = int(value)
                elif current_var == 'v_d':
                    key, value = line.split(': ')
                    value = value.replace("'", "")
                    v_d[key] = int(value)
                else:
                    key_parts = line.split('(')[1].split(')')[0].split(', ')
                    key_parts = [part.strip("'") for part in key_parts]
                    indices = tuple(key_parts)
                    value = line.split(': ')[-1]
                    if current_var == 'X_W':
                        X_W_init_wf[indices] = float(value)
                    elif current_var == 'Q_W':
                        Q_W_init_wf[indices] = float(value)
                    elif current_var == 'L_W':
                        L_W_init_wf[indices] = float(value)
                    elif current_var == 'Z_WV':
                        Z_WV_init_wf[indices] = float(value)
                    elif current_var == 'Y':
                        Y_init_wf[indices] = float(value)
                    elif current_var == 'B':
                        B_init_wf[indices] = float(value)
                    elif current_var == 'A_WV':
                        A_WV_init_wf[indices] = float(value)
                    elif current_var == 'A_DD':
                        A_DD_init_wf[indices] = float(value)
                    elif current_var == 'S':
                        S_init_wf[indices] = float(value)
                    elif current_var == 'LS':
                        LS_init_wf[indices] = float(value)
                    elif current_var == 'D':
```

```
                            D_init_wf[indices] = float(value)
                        elif current_var == 'D_WV':
                            D_WV_init_wf[indices] = float(value)
                        elif current_var == 'W':
                            W_init_wf[indices] = float(value)
                        elif current_var == 'P_V':
                            P_V[indices] = float(value)
                        elif current_var == 'Z_V':
                            Z_V[indices] = float(value)
                        elif current_var == 'D_T':
                            D_T[indices] = float(value)
                        elif current_var == 'L_V':
                            L_V[indices] = float(value)
                        elif current_var == 'T_V':
                            T_V_init_wf[indices] = float(value)
                        elif current_var == 'A_F':
                            A_F_init_wf[indices] = float(value)
                        elif current_var == 'T_W':
                            T_W_new_init_wf[indices] = float(value)

    zero = ['zero']
    WV_id = W_id + V_id
    WV0_id = zero + WV_id
    W0_id = zero + W_id
    V0_id = zero + V_id
    DS_id = DC + S_id

    #%%
    # Determine closest DC for each satellite
    DC_S = {}
    for s in S_id:
        dist_DC = 999999
        for dc in DC:
            dist_s_DC = dist_fe.at[canal_nodes_dict[dc], canal_nodes_dict[s]]
            if dist_s_DC < dist_DC:
                dist_DC = dist_s_DC
                dc_s = dc
        DC_S[s] = dc_s


    S_DC = {'DC_1': [], 'DC_2': [], 'DC_3': []}

    for s in S_id:
        dc = DC_S.get(s)
        if dc in S_DC:
            S_DC[dc].append(s)

    #%% Scheduling total
    print('Working on total scheduling for Ns:', Ns)
    start_sched_wr = time.time()
    model = gb.Model('Scheduling_total')
    np.random.seed(123)
    time_limit = t_lim_sched_total
    K = 9999

    X_W =  X_W_init_wf
    Q_W = Q_W_init_wf

    DC_W = {}
    for w in W_id:
        for d in DC:
            for i in S_id:
                if round(X_W_init_wf[i,d,w]) == 1:
                    DC_W[w] = d

    total_load = sum(Q_W_init_wf[i,w] for i in S_id for w in W_id)
    print('Q_W', sum(Q_W_init_wf[i,w] for i in S_id for w in W_id))

    # Binary variable, Y[i,k,l] = 1 if both k and l visit i
    Y = {}
    for k in WV_id:
```

```
        for l in WV_id:
            for i in DS_id:
                Y[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'Y')

    # Arrival time of vehicle r at i
    A_WV = {}
    for i in DS_id:
        for k in WV_id:
            A_WV[i,k] = model.addVar(lb = -500, ub = 999999, vtype = GRB.CONTINUOUS, name = 'A_WV')

    # Difference in arrival times of vehicle
    A_D = {}
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                A_D[i,k,l] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'A_D')

    # Difference in arrival times of vehicle
    A_DD = {}
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                A_DD[i,k,l] = model.addVar(lb = -999999, ub = 999999, vtype = GRB.CONTINUOUS, name = 'A_DD')

    # Customer or satellite is visited by vehicle r: = 1, if not: = 0
    Z_WV = {}
    for k in WV_id:
        for i in DS_id:
            Z_WV[i,k] =  model.addVar(vtype = GRB.BINARY, name = 'Z_WV')


    # New
    # Accumulated load of road vehicle r at customer i
    L_W = {}
    for w in WV_id:
        for i in DS_id:
            L_W[i,w] =  model.addVar(lb=0.0, vtype = GRB.CONTINUOUS, name = 'L_W')


    # Accumulated load delivered to satellite i by vehicles before and including k
    LS = {}
    for i in S_id:
        for k in WV0_id:
            LS[i,k] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'LS')

    # Number of water vehicles used
    Nw = {}
    for w in W_id:
        Nw[w] = model.addVar(vtype = GRB.BINARY, name = 'Nw')

    # Stock at satellite i after arrival of vehicle k
    S = {}
    for i in S_id:
        for k in WV_id:
            S[i,k] = model.addVar(lb = -100, vtype = GRB.INTEGER, name = 'S')

    B = {}
    for i in S_id:
        for k in WV0_id:
            for l in WV0_id:
                B[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'B')

    # New for scheduling!!!
    D_w = {}
    for w in W0_id:
        D_w[w] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'D_w')


    #T is matrix per road vehicle r, with trips k,l in V0, if r first performs trip k and
```

```
 then l, T[k,l,r] = 1
T_V = {}
for r in R_v:
    for k in V0_id:
        for l in V0_id:
            T_V[k,l,r] = model.addVar(vtype = GRB.BINARY, name = 'T_V')

A_R = {}
for r in R_v:
    for v in V0_id:
        A_R[v,r] = model.addVar(lb = -500, vtype = GRB.CONTINUOUS, name = 'A_R')

N_R = {}
for r in R_v:
    N_R[r] = model.addVar(vtype = GRB.BINARY, name = 'N_R')

Z_RV = {}
for r in R_v:
    for v in V_id:
        Z_RV[v,r] = model.addVar(vtype = GRB.BINARY, name = 'Z_RV')


# Scheduling water vehicles
T_W = {}
for f in F:
    for k in W0_id:
        for l in W0_id:
            T_W[k,l,f] = model.addVar(vtype = GRB.BINARY, name = 'T_W')

A_F = {}
for f in F:
    for w in W0_id:
        A_F[w,f] = model.addVar(lb = -500, vtype = GRB.CONTINUOUS, name = 'A_F')

N_F = {}
for f in F:
    N_F[f] = model.addVar(vtype = GRB.BINARY, name = 'N_F')

P_W = {}
for k in W0_id:
    for l in W0_id:
        P_W[l,k] = model.addVar(lb = 0.0, vtype = GRB.CONTINUOUS, name = 'P_W')


C_R = {}
for r in R_v:
    C_R[r] = model.addVar(vtype = GRB.CONTINUOUS, name = 'C_R')

# New for departure times
D = {}
for i in S_id:
    for k in WV0_id:
        for l in WV0_id:
            D[i,k,l] = model.addVar(vtype = GRB.BINARY, name = 'D')


D_WV = {}
for i in DS_id:
    for k in WV_id:
        D_WV[i,k] = model.addVar(lb = 0.0, ub = 999999, vtype = GRB.CONTINUOUS, name = 'D_WV')

W = {}
for i in DS_id:
    for w in W0_id:
        W[i,w] = model.addVar(vtype = GRB.CONTINUOUS, name = 'W')

D_max = {}
for l in W_id:
    D_max[l] = model.addVar(vtype = GRB.CONTINUOUS, name = 'D_max')
```

```
    Z_FW = {}
    for f in F:
        for w in W_id:
            Z_FW[w,f] = model.addVar(vtype = GRB.BINARY, name = 'Z_FW')

    D_r = {}
    for r in R_v:
        D_r[r] = model.addVar(vtype = GRB.CONTINUOUS, name = 'D_r')


    # Objective function
    model.setObjective(0.01* quicksum(D_r[r] for r in R_v) + 400* quicksum(N_R[r] for r in R_v) + 500 * quicksum(N_F[f] for f in F))

    model.modelSense = GRB.MINIMIZE
    model.update()

    # Constraints
    # 1. A vehicle never goes from i to i
    for w in W_id:
        for i in DS_id:
            for j in DS_id:
                if i == j:
                    constr_w_1 = model.addConstr(X_W[i,j,w] == 0, name='Constr_1')


    # 2. Vehicle r can only leave node if it also arrived there
    for w in W_id:
        for i in DS_id:
            # if i != j:
                constr_w_2 = model.addConstr(quicksum(X_W[i,j,w] for j in DS_id) == quicksum(X_W[j,i,w] for j in DS_id), name='Constr_2')

    # 3. Nodes that are visited by vehicle w
    for w in W_id:
        for i in DS_id:
            constr_w_3b = model.addConstr(Z_WV[i,w] == quicksum(X_W[i,j,w] for j in DS_id), name='Constr_3')

    # 4b. Nodes that are visited by vehicle r
    for v in V_id:
        for i in DS_id:
            constr_w_4c = model.addConstr(Z_WV[i,v] == Z_V[i,v], name='Constr_4')

    # New
    # 5. The demand delivered to i is zero if vehicle r does not visit i
    for w in W_id:
        for i in DS_id:
            constr_w_5 = model.addGenConstrIndicator (Z_WV[i,w], False, Q_W[i,w], GRB.EQUAL, 0, name='Constr_5')

    # 6. Demand satisfaction constraint
    for i in S_id:
            constr_w_6 = model.addConstr(quicksum(Q_W[i,w] for w in W_id) == LS_V[i], name='Constr_6')
            constr_w_6b = model.addConstr(Q_W[i,'zero'] == 0, name='Constr_6b')

    # 7. No load is delivered to DC
    # 8. The accumulated load at the DC is zero
    for w in W_id:
        DC = DC if isinstance(DC, list) else [DC]
        for i in DC:
            constr_w_7 = model.addConstr (Q_W[i,w] == 0, name='Constr_7')
            constr_w_8 = model.addConstr(L_W[i,w] == 0, name='Constr_8')

    # 8b. No load delivered by road vehicles
    for v in V_id:
        for i in DS_id:
            constr_w_8b = model.addConstr(Q_W[i,v] == 0, name='Constr_8b')
            constr_w_8c = model.addConstr(L_W[i,v] == 0, name='Constr_8c')

```

```
    # 9a_new. With X_W as an input, the constraint can be rewritten as:
    for w in W_id:
        for i in DS_id:
            for j in S_id:
                if X_W[i,j,w] == 1:
                    constr_9a_new = model.addConstr(L_W[j,w] - L_W[i,w] - Q_W[j,w] == 0, name = 'Constr_9a_new')

    # 9b. No L_R if not visited
    for w in W_id:
        for i in DS_id:
            constr_w_9b = model.addGenConstrIndicator (Z_WV[i,w], False, L_W[i,w], GRB.EQUAL, 0, name='Constr_9b')

    # New
    # 9c. The load delivered to customer i by vehicle r is always less than or equal to the accumulated load of r at customer i:
    for w in W_id:
        for i in S_id:
            constr_w_9c = model.addConstr(Q_W[i,w] <= L_W[i,w], name='Constr_9c')

    # New
    # 9d. The accumulated load of vehicle r at customer i is always less than or equal to the maximum capacity of vehicle r:
    for w in W_id:
        for i in S_id:
            constr_w_9d = model.addConstr( L_W[i,w] <= capacity_fe, name='Constr_9d')

    # # Arrival time constraints:

    # 10_new. With X_W as input
    for w in W_id:
        for i in DS_id:
            for j in S_id:
                if X_W[i,j,w] == 1:
                    constr_10_new = model.addConstr(A_WV[j,w] - A_WV[i,w] - dist_fe.at[canal_nodes_dict[i],canal_nodes_dict[j]] / (speed_fe * 60) - W[i,w] - Q_W[i,w] * 0.2 >= 0, name='Constr_10_new' )


    for w in W_id:
        for j in S_id:
            DC = DC if isinstance(DC, list) else [DC]
            for i in DC:
                if X_W[i,j,w] == 1:
                    constr_time_10b = model.addConstr(A_WV[j, w] - A_WV[i, w] - dist_fe.at[canal_nodes_dict[i], canal_nodes_dict[j]] / (speed_fe * 60) - service_time_fe / 60 == 0, name='Constr_10b')

    # 11. Binary variable Y[i,k,l] is one if both k and l visit i
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                if k != l:
                    constr_Y_11 = model.addConstr(Y[i,k,l] == gb.and_(Z_WV[i,k], Z_WV[i,l]), name='Constr_11')


    # 12. Arrival times of vehicles at satellites cannot be the same
    for i in S_id:
        for k in WV_id:
            for l in WV_id:
                constr_time_12a = model.addConstr(A_DD[i,k,l] == A_WV[i,k] - A_WV[i,l], name='Constr_12a')
                constr_time_12b = model.addConstr(A_D[i,k,l] == gb.abs_(A_DD[i,k,l]), name='Constr_12b')


    # 13a. Arrival times of road vehicles at satellites cannot be the same
    for i in S_id:
```

```
        for k in V_id:
            for l in V_id:
                if k != l:
                    constr_time_13a = model.addGenConstrIndicator(Y[i,k,l], True, A_D[i,k,l], GRB.GREATER_EQUAL, transship_s, name='Constr_13a') #180)
                    constr_time_13a_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13a_1')

    # 13b. Arrival times of a water vehicles is later than the departure time of another water vehicle
    for i in S_id:
        for k in W_id:
            for l in W_id:
                if k != l:
                    constr_time_13b = model.addGenConstrIndicator(B[i,k,l], True, A_WV[i,k] - D_WV[i,l] , GRB.GREATER_EQUAL, 0, name='Constr_13b') #600)
                    constr_time_13b_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13b_1')


    # 13b. Arrival times of water and road vehicles at satellites cannot be the same
    for i in S_id:
        for k in W_id:
            for l in V_id:
                if k != l:
                    constr_time_13c = model.addGenConstrIndicator(Y[i,k,l], True, A_D[i,k,l], GRB.GREATER_EQUAL, 0.01, name='Constr_13c') #600)
                    constr_time_13c_1 = model.addGenConstrIndicator(Y[i,k,l], False, A_D[i,k,l], GRB.GREATER_EQUAL, 0, name='Constr_13c_1')

    #13c. Arrival times at satellites cannot be later than the maximum time span
    for i in S_id:
        for k in WV_id:
            constr_time_13d = model.addConstr(D_WV[i,k] <= time_span, name='Constr_13d')


    # 14. Arrival time is infinite if a vehicle does not visit satellite i
    for i in S_id:
        for k in WV_id:
            constr_time_14 = model.addGenConstrIndicator(Z_WV[i,k], False, A_WV[i,k], GRB.EQUAL, 0, name = 'Constr_14')

    # # Satellite synchronisation constraints:

    # 15. Binary variable = 1 if vehicle k arrives at the same time or after vehicle l
    for i in S_id:
        for k in WV_id:
            constr_binary_150 = model.addGenConstrIndicator(Z_WV[i,k], True, B[i,k,'zero'], GRB.EQUAL, 1 , name = 'Constr_150')
            for l in WV_id:
                constr_binary_15a = model.addGenConstrIndicator(Y[i,k,l], True, A_WV[i,k] - K * B[i,k,l] - A_WV[i,l], GRB.LESS_EQUAL, 0 , name = 'Constr_15a')
                constr_binary_15b = model.addGenConstrIndicator(Y[i,k,l], True, B[i,k,l] + B[i,l,k], GRB.EQUAL, 1, name = 'Constr_15b')
                constr_binary_15c = model.addConstr(B[i,k,l] + B[i,l,k] <= 1, name = 'Constr_15c')
                constr_binary_15d = model.addGenConstrIndicator(Z_WV[i,k], False, B[i,k,l], GRB.EQUAL, 0, name = 'Constr_15d')
                constr_binary_15e = model.addGenConstrIndicator(Z_WV[i,k], False, B[i,l,k], GRB.EQUAL, 0 , name = 'Constr_15e')

    # 16. Load delivered to satellite i by all vehicles before k and k
    for i in S_id:
        for k in WV0_id:
            for l in WV0_id:
                constr_load_16a = model.addGenConstrIndicator(B[i,k,l], True, LS[i,k] - LS[i,l] - Q_W[i,k], GRB.GREATER_EQUAL,0 , name = 'Constr_load_16a')

    for i in S_id:
        for k in WV_id:
            constr_load_16b = model.addConstr(LS[i,k] <= quicksum(Q_W[i,w] for w in W_id) ,
```

```
name = 'Constr_load_16b')
            constr_load_16c = model.addGenConstrIndicator(Z_WV[i,k], False, LS[i,k], GRB.EQUAL, 0 , name = 'Constr_load_16c')

    # 17. New Stock at satellites constraints
    for i in S_id:
        for k in WV_id:
            # for l in WV_id:
                constr_stock_17a = model.addGenConstrIndicator (Z_WV[i,k], True, S[i,k] + quicksum(L_V[i,l] * B[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.EQUAL, 0, name = 'Constr_stock_17a')

    for i in S_id:
        for k in WV_id:
            constr_stock_17b = model.addConstr(S[i,k] >= 0, name = 'Constr_stock_17b')
            constr_stock_17c = model.addConstr(S[i,k] <= capacity_s[i] + capacity_fe, name = 'Constr_stock_17c')

    for w in W_id:
        constr_water_km = model.addConstr(D_w[w] == quicksum(dist_fe.at[canal_nodes_dict[i], canal_nodes_dict[j]] * X_W[i,j,w] for i in DS_id for j in DS_id), name = 'Constr_water_km')


    for w in W_id:
        for i in S_id:
            constr_Nw = model.addGenConstrIndicator(Z_WV[i,w], True, Nw[w], GRB.EQUAL, 1, name = 'Constr_Nw')

    # New for scheduling

    # Each vehicle r can only leave the depot once
    for r in R_v:
        constr_18f = model.addConstr(quicksum(T_V['zero',k,r] for k in V0_id) <= 1, name = 'Constr_18f')

    # Each trip is performed once
    for k in V_id:
        constr_18b = model.addConstr(quicksum(T_V[l,k,r] for l in V0_id for r in R_v) == 1, name = 'Constr_18b')

    # Trip k can be performed by vehicle r if the start time of trip k is later than the end time of trip l
    for r in R_v:
        for k in V_id:
            for l in V0_id:
                constr_18c = model.addGenConstrIndicator(T_V[l,k,r], True, A_R[k,r] - A_R[l,r] - P_V[l,k], GRB.GREATER_EQUAL, 0, name = 'Constr_18c')

    # A trip can never be performed after itself
    for r in R_v:
        for l in V0_id:
            constr_18d = model.addConstr(T_V[l,l,r] == 0, name = 'Constr_18d')

    # Vehicle r can only end trip l if it also started it
    for r in R_v:
        for l in V0_id:
            # if i != j:
                constr_18e = model.addConstr(quicksum(T_V[l,k,r] for k in V0_id) == quicksum(T_V[k,l,r] for k in V0_id), name = 'Constr_18e')


    # Number of road vehicles used
    for r in R_v:
        for k in V_id:
            constr_19 = model.addGenConstrIndicator(T_V['zero',k,r], True, N_R[r], GRB.EQUAL, 1, name = 'Constr_19')

    # Z_RV = 1 if r performs trip v
    for k in V_id:
        for r in R_v:
```

```
            constr_20a = model.addConstr(Z_RV[k,r] == quicksum(T_V[l,k,r] for l in V0_id), name = 'Constr_20a')

    # Set A_R to zero if r does not perform trip
    for v in V_id:
        for r in R_v:
            constr_20b = model.addGenConstrIndicator(Z_RV[v,r], False, A_R[v,r], GRB.EQUAL, 0, name = 'Constr_20b')

    # Connect A_R with A_WV
    for v in V_id:
        constr_20c = model.addConstr(A_WV[v_s[v],v] == quicksum(A_R[v,r] for r in R_v), name = 'Constr_20c')

    # Completion time for vehicle r is the start time of the last trip + the time to perform the last trip
    for r in R_v:
        for v in V_id:
            constr_20d = model.addGenConstrIndicator(T_V[v,'zero',r], True, C_R[r] - A_R[v,r] - P_V[v,'zero'], GRB.EQUAL, 0, name = 'Constr_20d')


    for k in WV_id:
        for i in S_id:
            constr_time_span = model.addConstr(A_WV[i,k] >= 0 , name = 'constr_time_span')

    # Each vehicle f can only leave the depot once
    for f in F:
        constr_22f = model.addConstr(quicksum(T_W['zero',k,f] for k in W_id) <= 1, name = 'Constr_22f')

    # Each trip is performed once
    for k in W_id:
        constr_22b = model.addConstr(quicksum(T_W[l,k,f] for l in W0_id for f in F) == 1, name = 'Constr_22b')

    #A vessel can only perform trips in the same neighbourhood:
    for f in F:
        for k in W_id:
            depot_k = DC_W[k]
            for l in W_id:
                if DC_W[l] != depot_k:
                    constr_neighbour = model.addGenConstrIndicator(Z_FW[k,f], True, Z_FW[l,f], GRB.EQUAL, 0, name = 'Constr_neighbour' )
                    constr_neighbour_b = model.addGenConstrIndicator(Z_FW[k,f], True, T_W[l,k,f] + T_W[k,l,f], GRB.EQUAL, 0, name = 'Constr_neighbour_b' )


    # Z_FW = 1 if f performs trip w
    for k in W_id:
        for f in F:
            constr_22b_1 = model.addConstr(Z_FW[k,f] == quicksum(T_W[l,k,f] for l in W0_id), name = 'Constr_22b_1')

    for f in F:
        for l in W_id:
            constr_22b_2 = model.addGenConstrIndicator(Z_FW[l,f], True, quicksum(T_W['zero',k,f] for k in W_id), GRB.EQUAL, 1)

    for l in W_id:
        model.addConstr(D_max[l] == gb.max_(D_WV[i,l] for i in S_id)) # gb.max_(D_WV[i,l] for i in S_id)

    # New: Trip k can be performed by vehicle f if the start time of trip k is later than the end of trip l
    for f in F:
        for k in W_id:
            for l in W_id:
                constr_22c_new = model.addGenConstrIndicator(T_W[l,k,f], True, A_F[k,f] - D_max[l] - quicksum(X_W[i,d,l] * dist_fe.at[canal_nodes_dict[i],canal_nodes_dict[d]] for i in S_id for d in DC) / (speed_fe * 60), GRB.GREATER_EQUAL, 0, name = 'Constr_22c_new')
```

```python

    # A trip can never be performed after itself
    for f in F:
        for l in W0_id:
            constr_22d = model.addConstr(T_W[l,l,f] == 0, name = 'Constr_22d')

    # Vehicle f can only end trip l if it also started it
    for f in F:
        for l in W0_id:
            # if i != j:
                constr_22e = model.addConstr(quicksum(T_W[l,k,f] for k in W0_id) == quicksum(T_W[k,l,f] for k in W0_id), name = 'Constr_22e')

    # Number of water vehicles used
    for f in F:
        for k in W_id:
            constr_23 = model.addGenConstrIndicator(T_W['zero',k,f], True, N_F[f], GRB.EQUAL, 1, name = 'Constr_23')


    # New for multiple depots water vehicles:
    # Connect A_F with A_WV
    for w in W_id:
        constr_24 = model.addConstr(A_WV[DC_W[w],w] == quicksum(quicksum(T_W[k,w,f] for k in W0_id ) * A_F[w,f] for f in F), name = 'Constr_24')

    # New for departure times
    for i in S_id:
        for v in V_id:
            constr_departure_1 = model.addGenConstrIndicator(Z_WV[i,v], True, D_WV[i,v] - A_WV[i,v] - transship_s / 60, GRB.EQUAL, 0, name = 'constr_dep_1')
        for w in W_id:
            constr_departure_2 = model.addGenConstrIndicator(Z_WV[i,w], True, D_WV[i,w] - A_WV[i,w] - W[i,w] - Q_W[i,w] * 0.2, GRB.EQUAL, 0, name = 'constr_dep_2' )


    for i in S_id:
        for k in V0_id:
            for l in V0_id:
                constr_departure_3 = model.addGenConstrIndicator(B[i,k,l], True, D[i,k,l], GRB.EQUAL, 1, name = 'constr_dep_3')
        for k in W0_id:
            for l in W0_id:
                constr_departure_4 = model.addGenConstrIndicator(B[i,k,l], True, D[i,k,l], GRB.EQUAL, 1, name = 'constr_dep_4')

        for k in WV_id:
            constr_departure_8 =  model.addGenConstrIndicator (Z_WV[i,k], False, quicksum(D[i,k,l] for l in WV_id) + quicksum(D[i,l,k] for l in WV_id), GRB.EQUAL, 0, name = 'constr_dep_8')
            for l in WV_id:
                constr_departure_5 = model.addGenConstrIndicator(Y[i,k,l], True, D_WV[i,k] - K* D[i,k,l] - D_WV[i,l], GRB.LESS_EQUAL, 0, name = 'constr_dep_5')
                constr_departure_6 = model.addGenConstrIndicator(Y[i,k,l], True, D[i,k,l] + D[i,l,k], GRB.EQUAL, 1, name = 'constr_dep_6')

    for i in S_id:
        for k in W_id:
            constr_departure_7 = model.addGenConstrIndicator (Z_WV[i,k], True, quicksum(L_V[i,l] * D[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.LESS_EQUAL, 0, name = 'constr_dep_7')
            constr_departure_7_b = model.addGenConstrIndicator (Z_WV[i,k], True, quicksum(L_V[i,l] * D[i,k,l] for l in V_id) + L_V[i,k] - LS[i,k], GRB.GREATER_EQUAL, - capacity_s[i], name = 'constr_dep_7_b')

    # New new
    # Distance on the road per vehicle
    for r in R_v:
        constr_distance_r = model.addConstr(D_r[r] == quicksum(T_V[l,k,r] * D_T[l,k] for l in V0_id for k in V0_id))

```

```
    for (i,w), value in L_W_init_wf.items():
        L_W[i,w].start = value

    for (i,w), value in Z_WV_init_wf.items():
        Z_WV[i,w].start = value

    for (i,k,l), value in Y_init_wf.items():
        Y[i,k,l].start = value

    for (i,k,l), value in B_init_wf.items():
        B[i,k,l].start = value

    for (i, k), value in A_WV_init_wf.items():
        A_WV[i,k].start = value

    for (i,w), value in S_init_wf.items():
        S[i,w].start = value

    for (i,k), value in LS_init_wf.items():
        LS[i,k].start = value

    for (i,k,l), value in T_V_init_wf.items():
        if k in V0_id:
            if i in V0_id:
                if l in R_v:
                    T_V[i,k,l].start = value

    for (i,k,l), value in T_W_new_init_wf.items():
        T_W[i,k,l].start = value

    for (i,k,l), value in D_init_wf.items():
        D[i,k,l].start = value

    for (i, k), value in D_WV_init_wf.items():
        D_WV[i,k].start = value

    for (i, k), value in W_init_wf.items():
        W[i,k].start = value

    model.update()

    print("start optimizing")
    model.setParam( 'OutputFlag', True)
    model.setParam ('MIPGap', mip_sched_t)
    model.setParam('FeasibilityTol', 1e-3)
    model.setParam('MIPFocus', 1)
    model.setParam('SubMIPNodes', 20000)
    model.setParam('SoftMemLimit', 120)
    model.setParam('Seed', 123)
    if time_limit:
        model.setParam('Timelimit', time_limit)
    model._obj = None
    model._bd = None
    model._obj_value = []
    model._time = []
    model._start = time.time()
    model.optimize()
    mip_gap_total = model.MIPGap
    end_sched_wr = time.time()
    time_sched_wr = end_sched_wr - start_sched_wr


    #%% Save solutions total scheduling
    road_km_R_wr = {}
    total_road_km = 0
    for r in R_v:
        save_road_km = 0
        for l in V0_id:
            for k in V0_id:
                if T_V[l,k,r].X == 1:
```

```
                    save_road_km += D_T[l,k]
                    total_road_km += D_T[l,k]
        road_km_R_wr[r] = save_road_km

    road_km_wr = total_road_km


    water_km_wr = 0
    for w in W_id:
        water_km_wr += D_w[w].X

    r_used = 0
    R_final = []
    for r in R_v:
        if N_R[r].X == 1:
            r_used += 1
            R_final.append(r)
    print (r_used)
    Nr_R_wr = r_used
    print('distance on the roads wr: ',road_km_wr, 'distance on the waterways wr: ',
    water_km_wr)
    f_used = 0
    F_final = []
    for f in F:
        if N_F[f].X == 1:
            f_used += 1
            F_final.append(f)
    print (f_used)
    Nr_F_wr = f_used

    W_used_wr = []
    for w in W_id:
        w_visits = 0
        for i in S_id:
            if Z_WV[i,w].X == 1:
                    w_visits += 1
        if w_visits >= 1:
            W_used_wr.append(w)
    Nr_w_wr = len(W_used_wr)
    #%%
    max_complete = 0
    for r in R_v:
        for k in V0_id:
            for l in V0_id:
                if T_V[l,k,r].X == 1:
                    if C_R[r].X > max_complete:
                        max_complete = C_R[r].X

    max_start_R_wr = max_complete

    max_start = 0
    for f in F:
        for w in W_id:
            if A_F[w,f].X > max_start:
                max_start = A_F[w,f].X
    max_start_F_wr = max_start

    N_s.append(t_lim_VRP_E2)

    results.append({'road_km_wr': road_km_wr,
                    'water_km_wr': water_km_wr,
                    'Nr_R_wr': Nr_R_wr,
                    'Nr_F_wr': Nr_F_wr,
                    'MIP_VRP_E2': MIP_VRP_E2,
                    'MIP_VRP_E1': MIP_VRP_E1,
                    'MIP_sched_r': MIP_sched_r,
                    'MIP_sched_w': MIP_sched_w,
                    'MIP_sched_wr': mip_gap_total,
                    'D_r_VRP_E2': D_r_VRP_E2,
                    'D_r_r': D_r_r,
                    'D_w_VRP_E1': D_w_VRP_E1,
```

```
                     'D_w_w': D_w_w,
                     'Nr_R_r': R_sched_r,
                     'Nr_F_w': F_sched_w,
                     'Nr_w_wr': Nr_w_wr,
                     'time_sched_wr': time_sched_wr,
                     'max_complete_R_wr': max_start_R_wr, 'max_start_F_wr': max_start_F_wr})
    print(results)

    X_W_final = X_W
    Y_final = model.getAttr('X', Y)
    A_WV_final = model.getAttr('X', A_WV)
    A_D_final = model.getAttr('X', A_D)
    A_DD_final = model.getAttr('X', A_DD)
    Q_W_final = Q_W
    Z_WV_final = model.getAttr('X', Z_WV)
    L_W_final = model.getAttr('X', L_W)
    LS_final = model.getAttr('X', LS)
    S_final = model.getAttr('X', S)
    B_final = model.getAttr('X', B)
    A_F_final = model.getAttr('X', A_F)
    T_W_final = model.getAttr('X', T_W)
    T_V_final = model.getAttr('X', T_V)
    D_final = model.getAttr('X', D)
    D_WV_final = model.getAttr('X', D_WV)
    W_final = model.getAttr('X', W)
    with open(f'output_total_{save_title}_{Ns}_{t_lim_VRP_E2}.txt', 'w') as f:
        for var_name, var_values in [
            ('X_W', X_W_final),
            ('Y', Y_final),
            ('A_WV', A_WV_final),
            ('A_D', A_D_final),
            ('A_DD',A_DD_final),
            ('Q_W', Q_W_final),
            ('Z_WV', Z_WV_final),
            ('L_W', L_W_final),
            ('LS', LS_final),
            ('S', S_final),
            ('B', B_final),
            ('D', D_final),
            ('D_WV', D_WV_final),
            ('W', W_final),
            ('T_V', T_V_final),
            ('A_F', A_F_final),
            ('T_W', T_W_final),
            ('P_V', P_V),
            ('L_V', L_V),
            ('LS_V', LS_V),
            ('Z_V', Z_V),
            ('v_s', v_s),
            ('L_V', L_V),
            ('D_T', D_T),
            ('canal_nodes_dict', canal_nodes_dict)
        ]:
            f.write(f'{var_name}:\n')
            for key, value in var_values.items():
                if isinstance(value, gb.LinExpr):
                    value = value.getValue()
                f.write(f'  {key}: {value}\n')
        f.write('V_id:\n')
        for v in V_id:
            f.write(f'{v}\n')
        f.write('W_id:\n')
        for w in W_id:
            f.write(f'{w}\n')
        f.write('S_id:\n')
        for s in S_id:
            f.write(f'{s}\n')
        f.write('R_final:\n')
        for r in R_final:
            f.write(f'{r}\n')
        f.write('F_final:\n')
```

```
        for k in F_final:
            f.write(f'{k}\n')
            print('F is written')
        f.write('DC:\n')
        for d in DC:
            f.write(f'{d}\n')
#%%
results_NS = pd.DataFrame(results, index = N_s)
print(results_NS)
with pd.ExcelWriter(path_out + 'Results_full_storage.xlsx', engine = 'openpyxl',mode = 'a')
    as writer:

    results_NS.to_excel(writer, sheet_name = f'Final_{save_title}_{Ns}')
```