Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

Jacob Nitter
Shusheng Yang

The Static Ridesharing Routing Problem with Flexible Locations

An Adaptive Large Neighborhood Search Heuristic

Master's thesis in Industrial Economics and Technology Management
Supervisor: Kjetil Fagerholt
Co-supervisor: Andreas Breivik Ormevik
June 2023

NTNU
Norwegian University of Science and Technology

Jacob Nitter
Shusheng Yang

The Static Ridesharing Routing Problem with Flexible Locations

An Adaptive Large Neighborhood Search Heuristic

Master's thesis in Industrial Economics and Technology Management
Supervisor: Kjetil Fagerholt
Co-supervisor: Andreas Breivik Ormevik
June 2023

Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

NTNU
Norwegian University of
Science and Technology

Preface

This master's thesis represents the culmination of our Master of Science program at the Norwegian University of Science and Technology (NTNU), Department of Industrial Economics and Technology Management. The research work was conducted during the spring of 2023 and serves as a continuation of our specialization project in TIØ4500 Managerial Economics and Operations Research, which took place in the fall of 2022 (Nitter & Yang (2022)).

We would like to express our sincere appreciation to our supervisor, Professor Kjetil Fagerholt, and our co-supervisor, Ph.D. student Andreas Breivik Ormevik, for their invaluable guidance and advice throughout this endeavor. Additionally, we extend our gratitude to Telia and their Crowd Insights service for providing us with valuable data.

Jacob Nitter, Shusheng Yang

Trondheim, 5th June 2023

Abstract

Ridesharing has emerged as a promising solution to the transportation challenges faced by suburban and rural areas. At its core, it aims to reduce traffic congestion, decrease carbon emissions, and foster a sense of community by sharing travel expenses among participants. This potential is particularly pronounced in regions like Sotra and the greater Bergen area in Norway, where geographical and demographic characteristics result in an increased reliance on private vehicles. One notable area of congestion is Sotrabroen, the primary bridge connecting Sotra to Bergen, which experiences high traffic volumes, increasing the negative environmental impacts. Despite these advantages, ridesharing has not yet become a mainstream alternative due to challenges such as schedule coordination and convenience issues.

This thesis addresses these challenges by developing a coordination method for a ridesharing system, identified as the Static Ridesharing Routing Problem with Flexible Locations (SRRPFL). The primary objective of this method is to design efficient routes. By enabling vehicles to pick up and deliver passengers at various flexible locations, the method primarily aims to maximize passenger participation and, secondarily, to minimize the total travel time for drivers. The SRRPFL takes into account individual travel needs, such as origin and destination locations, along with preferences including maximum travel time and arrival time window.

To solve the SRRPFL, an exact mathematical arc-flow mixed integer programming model is formulated. However, for realistic and large-scale problem instances, using a commercial solver proves too time-consuming. Therefore, an adaptive large neighborhood search (ALNS) heuristic, including extensions of local search and a set-partitioning formulation, is implemented. This heuristic approach is tailored for the SRRPFL, and designed to deliver solutions quickly while still ensuring a high level of solution quality.

Tests on instances, based on real data provided by Telia, reveal that the ALNS heuristic produces high-quality solutions. These tests also demonstrate that the use of flexible locations notably increases passenger participation in the ridesharing scheme, thereby reducing the need for individual car usage and alleviating traffic congestion. By adapting to the needs and preferences of potential participants, the flexibility of locations reduces travel distance and travel time for drivers, subsequently resulting in lower CO_2 emissions. When comparing models, the one featuring flexible locations outperforms its inflexible counterpart not just in terms of the number of passengers participating, but also in operational metrics like reduced travel distance and travel time.

Sammendrag

Samkjøring fremstår som en lovende løsning på trafikkutfordringene man møter i forsteder og landlige områder. Hovedpoenget er å redusere trafikkbelastningen og karbonutslippene, og fremme fellesskapsfølelsen ved å dele reisekostnadene blant deltakerne. Potensialet for disse effektene er særlig uttalt i regioner som Sotra og området rundt Bergen, hvor både geografien og demografien fører til en økt avhengighet av private kjøretøy. Et område med en stor trafikkbelastning er Sotrabroen, broen som forbinder Sotra og Bergen. Til tross for de uttalte fordelene med samkjøring har dette ikke blitt et utbredt alternativ. Dette skyldes muligens utfordringer knyttet til koordrinering og hvordan man kan gjøre samkjøring lettvint for alle de involverte.

I denne masteroppgaven addresser vi disse utfordringene ved å utvikle en koordineringsmetode for et samkjøringssystem, kalt Static Ridesharing Routing Problem with Flexible Locations (SRRPFL). Hovedmålet med denne metoden er å bestemme effektive ruter. Ved å legge opp til at passasjerer kan bli plukket opp og levert på fleksible lokasjoner har SRRPFL som sitt primære mål å maksimere antall passasjerer som blir plukket opp og levert, mens det sekundære målet er å minimere den totale reisetiden for sjåførene. SRRPFL tar hensyn til individuelle reisebehov, som start- og sluttlokasjoner, maksimal reisetid for sjåfører og passsasjerer og ankomstvinduet til sjåfører og passasjerer.

For å løse SRRPFL introduseres en eksakt matematisk modell. Imidlertid, for realistiske probleminstanser blir denne modellen for tidkrevende å løse. Derfor er en adaptive large neighborhood search (ALNS) heuristikk, inkludert utvidelser med lokaløsk og settpartisjonsformulering, blitt implementert. Denne heuristikken er skreddersydd for SRRPFL og designet for å produsere løsninger raskt, samtidig som den sikrer høy kvalitet på løsnningene.

Etter at ALNS-heuristikken er testet på probleminstanser generert på ekte data fra Telia, kan man observere at ALNS-heuristikken produserer løsninger av høy kvalitet. Disse testene viser også at bruken av fleksible lokasjoner kan øke antaller passasjerer som blir plukket opp merkbart. Dette vil igjen kunne redusere behovet for individuell bilbruk og avlaster derfor trafikkbelastningen. Ved å tilpasse seg behovene og preferansene til sjåførene og passasjerene vil fleksible lokasjoner kunne bidra til å redusere antall kilometer kjørt, som igjen vil resultere i lavere CO_2-utslipp. Hvis man sammenligner de to scenariene med fleksible lokasjoner og uten fleksible lokasjoner, observerer man at scenariet med fleksible lokasjoner overgår sin motpart både i form av antall opplukkede passasjerer, men også i redusert reiseavstand og reisetid.

Table of Contents

List of Figures

List of Tables

Abbreviations and Frequently Used Terms

Abbreviations

SRRPFL	Static Ridesharing Routing Problem with Flexible Locations
ALNS	Adaptive large neighborhood search
LNS	Large neighborhood search
LS	Local search
RCP	Route combination problem

Definition of frequently used terms

Origin location	A starting location for a *driver* or a *passenger*, e.g. their home
Destination location	An ending location for a *driver* or a *passenger*, e.g. their workplace
Candidate location	A location a passenger can either be picked up at, or delivered at

Chapter 1

Introduction

Ridesharing is a transportation mode in which individual travelers share a vehicle, following similar itineraries and time schedules. At its core, ridesharing involves one or more passengers sharing a vehicle with a driver as they travel from their starting points to their respective destinations. The advantages of ridesharing include the sharing of travel expenses such as fuel, tolls, and parking fees among participants, the reduction of congestion and pollution for the general public, and the fostering of a sense of community among travelers. Moreover, ridesharing can contribute to a more sustainable transportation system by reducing the number of single-occupancy vehicles on the road, thereby decreasing traffic congestion and carbon emissions. However, despite these benefits, ridesharing has not yet become a mainstream transportation alternative due to the lack of efficient methods for coordinating schedules, as well as concerns surrounding trust and convenience.

In the case of Norway, the need for efficient and sustainable transportation solutions is further emphasized by the country's unique geographical and demographic characteristics. A considerable portion of the population resides in suburban areas and on the outskirts of cities and towns, often requiring long-distance commutes to urban centers for work or other activities. The Norwegian landscape, with its numerous fjords and islands, adds complexity to transportation networks and can lead to a reliance on ferries or bottleneck bridges for commuting between regions. This situation results in a higher dependency on private vehicles, leading to increased traffic congestion and associated negative environmental impacts. Additionally, public transportation options may be limited or inconvenient for these commuters.

One particular region that highlights these challenges is the island of Sotra and the Bergen municipality on the west coast of Norway. Sotra, situated just outside of Bergen, exemplifies a region that faces transportation difficulties due to its bottleneck bridge called Sotrabroen connecting the island to the mainland. With nearly 30,000 car movements crossing the bridge daily (Statens Vegvesen, 2023), traffic congestion is a common occurrence, causing delays and frustration for commuters. A majority of residents on Sotra who work in Bergen must rely on this bridge to access their workplaces.

A successful implementation of a ridesharing system in Sotra could alleviate some of the pressure on the bridge, reduce the number of vehicles on the road, and ultimately decrease traffic congestion. By optimizing the use of available resources and promoting a culture of shared mobility, ridesharing has the potential to create a more efficient and sustainable transportation network for the residents of Sotra and the greater Bergen area. Furthermore, a well-designed ridesharing system could serve as a model for other similar

regions facing similar transportation challenges, both in Norway and beyond. Ridesharing has the potential to offer a viable solution to these challenges by connecting individuals with similar travel needs and enabling them to share resources, thereby reducing the number of cars on the road and mitigating environmental impacts. Moreover, ridesharing can help bridge the gap in public transportation services, providing a more convenient, cost-effective, and sustainable alternative for commuters living in suburban or rural areas.

Despite the potential benefits of ridesharing, implementing a successful system in regions such as Sotra and the greater Bergen area presents several challenges that must be addressed. One key challenge is the coordination of passengers and drivers, which requires an effective platform that can match individuals with similar travel needs while considering factors such as timing, pick-up and delivery locations, and vehicle capacity. Additionally, fostering trust among users is crucial in encouraging widespread adoption, as individuals may be hesitant to share rides with strangers due to concerns about reliability. Another challenge lies in overcoming the deeply ingrained car-centric culture and mindset, which may lead individuals to prefer the privacy and convenience of their private vehicles over participating in ridesharing initiatives. Public awareness campaigns and incentives may be necessary to shift attitudes and promote the adoption of shared mobility options. Lastly, addressing the potential for unintended consequences, such as increased vehicle distance traveled due to additional trips generated by ridesharing services or the cannibalization of public transit commuters, is essential to ensure that ridesharing contributes positively to the overall transportation system.

Recognizing the need for innovative transportation solutions, the Bergen municipality has recently engaged in discussions with local administration to explore potential strategies for addressing these challenges. Consequently, a pilot project for ridesharing in the region is planned to be announced for tender soon. This development is taking place in real-time while this thesis is being written, underlining the urgency and relevance of the topic.

This thesis aims to address the challenge of effectively coordinating passengers and drivers in a ridesharing system by developing an approach that takes into account the various factors influencing the ridesharing experience. Through the consideration of individual travel needs, preferences, and constraints, the thesis seeks to create a more efficient transportation in regions such as Sotra and the greater Bergen area. The approach taken in this thesis acknowledges the importance of flexible pick-up and delivery locations, allowing for a greater degree of adaptability and customization in ridesharing arrangements. This flexibility involves passengers not necessarily being picked up directly at their homes; instead, they can travel a short distance to another pick-up location. Similarly, passengers do not have to be delivered right at their destination, but can be delivered at a nearby location from which they can travel to their final destination. This flexibility is important in improving the overall efficiency of the system, enabling better matches between drivers and passengers, and accommodating diverse travel requirements.

The Static Ridesharing Routing Problem with Flexible Locations (SRRPFL) is a transportation optimization problem, aiming to design efficient routes for a set of drivers to pick-up and deliver passengers at different, flexible locations. The SRRPFL addresses the challenge of coordinating multiple drivers and passengers in a ridesharing system, while considering each participant's individual preferences and constraints. The SRRPFL aims to optimize the ridesharing experience by maximizing the number of participating passengers and minimizing the total travel time for all drivers. By incorporating flexible pick-up and delivery locations, the SRRPFL allows for a higher degree of adaptability and customization, enabling better matches between drivers and passengers. The solution takes

into account the participants' travel information, such as origin and destination locations, time windows, maximum travel times, and vehicle capacities. The output generated by the SRRPFL is an optimized set of routes and schedules for each driver and passenger, ensuring that the objectives are met while adhering to the underlying assumptions and operational constraints of the problem.

In solving the SRRPFL, we first formulate an arc-flow mixed integer programming model. This model is solved using an exact method with a commercial solver for small-sized instances. For tackling larger and more realistic problem instances, we implement an adaptive large neighborhood search (ALNS) heuristic. To further enhance the performance of the ALNS, we integrate local search extensions and a set partitioning formulation. We put this ALNS heuristic to the test on instances that are based on real-world data supplied by Telia, and it proves to deliver high-quality solutions.

Thus, this thesis aims to investigate how ridesharing can contribute to reducing traffic congestion in Sotra and the greater Bergen area. The main contributions of this report are: 1) the introduction of a ridesharing optimization problem that incorporates both pick-up and delivery of passengers with the utilization of flexible pick-up and delivery locations, 2) the presentation of an adaptive large neighborhood search (ALNS) as a solution methodology for our proposed ridesharing problem, which includes local neighborhood search heuristics and a set-partitioning approach, and 3) the application of a real-world case study to reveal the effects of flexible locations.

Chapter 2 reviews related literature on ridesharing. The problem is described in Chapter 3. Chapter 4 presents the mathematical formulation of the problem. Furthermore, Chapter 5 presents the adaptive large neighborhood search (ALNS) heuristic. Chapter 6 details the case study considered in this thesis and test instance generation. The computational study of the ALNS heuristic performance is detailed in Chapter 7. Chapter 8 discusses the managerial insights gained from the computational study. Chapter 9 concludes this thesis, and Chapter 10 provides suggestions for future work.

Chapter 2

Literature Review

This chapter reviews existing literature found relevant for the Static Ridesharing Routing Problem with Flexible Locations (SRRPFL), building on the review presented by Nitter & Yang (2022). Section 2.1 presents literature on ridesharing optimization problems. Section 2.2 presents literature on Dial-A-Ride problems (DARP) and Pick-Up and Delivery problems (PDP) relevant to ridesharing.

2.1 Ridesharing

The field of ridesharing research can benefit from a more precise classification of its scenarios. In our study, we propose two distinct categories, coined as *organic* and *inorganic ridesharing*, based on the degree of structure and formal organization present in the ridesharing arrangements. It is important to clarify that this classification is our proposition and, to our knowledge, has not been explicitly stated in the existing literature. *Inorganic ridesharing* represents scenarios where an entity such as a company or workplace coordinates and manages the ridesharing service for its customers, employees and members. These arrangements typically involve predetermined drivers and passengers, where the sole purpose of a driver is to pick-up and deliver passengers. Examples of inorganic ridesharing includes Uber, taxis and arrangements made by companies for its employees. On the other hand, *organic ridesharing* represents a more flexible, decentralized approach where individuals independently rideshare based on their personal schedules and preferences. In this case, there is no prearranged plan or designated driver decided by a company or workplace. Instead, an individual, such as an employee, may decide to pick up colleagues on their way to work, requiring a more structured decision-making process to determine which drivers will pick up which passengers. An example of organic ridesharing is colleagues with a common workplace deciding to rideshare. Understanding these differences is essential as we delve further into the literature, as each type of ridesharing has unique characteristics, challenges, and objectives.

Ridesharing, as a relatively recent transportation mode, has gained attention in both research and practice due to its potential to reduce traffic congestion, lower emissions, and offer a more cost-effective alternative to traditional transportation options. Agatz et al. (2012) provide a comprehensive review of various types of ridesharing optimization problems, each with distinct characteristics and challenges. One key aspect of ridesharing systems is the underlying network structure, which determines the relationships between pick-up and delivery locations, as well as the routes that drivers can take. The authors

discuss the use of both static and dynamic network representations, noting that dynamic networks better capture the real-time nature of ridesharing services. The degree of dynamism in problem settings is another important aspect highlighted by the authors, which can range from fully static (all information is known beforehand) to highly dynamic (information about riders and drivers becomes available over time). Agatz et al. (2012) emphasize that dynamic problem settings present greater computational challenges. Their review also explores various optimization objectives that can be pursued in ridesharing problems, such as minimizing travel time, reducing the number of vehicles on the road, or maximizing the overall system utility. These objectives often involve trade-offs, which can be addressed using single or multi-objective optimization approaches. Furthermore, Agatz et al. (2012) explore different algorithmic techniques employed in ridesharing optimization research, including exact methods, heuristics, and metaheuristics, such as genetic algorithms, simulated annealing, and ant colony optimization. Each method has its strengths and limitations, making it important to choose the appropriate approach based on the specific problem characteristics and computational requirements.

Carpooling, as a specific form of ridesharing, has been the subject of various optimization approaches aimed at improving its efficiency and attractiveness. Baldacci et al. (2004) present an exact method for solving the carpooling problem using a Lagrangean column generation approach. Their research focuses on matching passengers with drivers organized by companies that encourage their employees to pick up colleagues while driving to and from work in an inorganic setting. The objective is to minimize the sum of the costs of the paths used to reach the workplace and the cost of deriving from the penalties of the unserviced clients, while taking into account constraints such as vehicle capacities, time windows, and maximum travel times for each participant. The authors formulate both an mixed integer formulation and a set-partitioning formulation. Baldacci et al. (2004) propose both an exact and a heuristic method to solve this problem. The exact method uses a bounding procedure that involves different Lagrangian relaxations of the problem. For the heuristic method, the authors propose a set-partitioning formulation for the problem, employing the column generation approach. Baldacci et al. (2004) demonstrate that their approach can provide high-quality solutions within reasonable computational times. Their research contributes to the development of exact methods for carpooling optimization and provides insights into the use of column generation and Lagrangean relaxation techniques in transportation problems.

The vanpool assignment problem, a specific form of ridesharing, has been studied by Kaan & Olinick (2013). In their paper, the authors focus on inorganically assigning passengers to vanpools, which are larger capacity vehicles typically provided by employers or public transportation agencies to facilitate commuting among groups of individuals with similar travel patterns. Kaan & Olinick (2013) take various constraints into consideration such as vehicle capacities, time windows, and alternative meeting locations. The inclusion of alternative meeting points for passengers, called park-and-ride locations, allows for more flexible arrangements that can accommodate a wider range of passenger preferences, potentially increasing the adoption of vanpooling as a sustainable transportation alternative. The objective is minimizing total travel cost. Kaan & Olinick (2013) propose three heuristics to solve the vanpool assignment problem, referred to as the restricted allowance heuristic (RAH), the relaxed restricted allowance heuristic (RRAH), and the greedy cover heuristic (GCH). The RAH reduces the number of possible park-and-ride combinations by eliminating those unlikely to be used in an optimal solution. The RRAH relaxes the vanpool assignment problem into a linear program (LP), using the LP solution to identify the most useful park-and-ride locations. The GCH selects a minimal set of park-and-ride combinations that cover all passengers that can be covered. Kaan & Olinick (2013)

demonstrate the effectiveness of their approach in producing high-quality solutions within reasonable computational times.

He et al. (2023) address a transportation problem to intercity transportation hubs such as railway stations and airports. Recognizing the need for reliable and cost-effective solutions, the authors propose a ridesharing approach using a mixed integer linear programming (MILP) model. In this study, the focus is on first-mile ridesharing to intercity transportation hubs (FMRITH), offering services within a defined radius from the hub. This is under the consideration that passengers can book these services based on their specific travel requirements. The authors use a rolling horizon method to dynamically group requests based on their arrival times and constraints such as the presence of large luggage, specific arrival time windows and maximum travel time requirements, and travel time uncertainties. The proposed MILP model aims to minimize the total transportation cost for ridesharing service providers. For small-scale cases, the FMRITH problem can be solved using commercial solvers like CPLEX and Gurobi. However, solving large-scale instances is time-consuming. To address this, He et al. (2023) design an algorithm based on the adaptive large neighborhood search by Ropke & Pisinger (2006) for a more efficient solution to large-scale FMRITH problems. The quality of the solution is assessed using a column generation algorithm and a greedy algorithm. Moreover, the authors emphasizes the need to consider large luggage and travel time uncertainty when planning ridesharing services to intercity transportation hubs. The models and algorithms proposed by the authors are shown to effectively tackle the FMRITH problem.

Auad-Perez & Hentenryck (2022) explore the concept of on-demand multimodal transit systems (ODMTS), which integrate bus or rail routes between transit hubs with on-demand shuttles. The study specifically targets the development of an ODMTS network design aimed at minimizing total travel costs, bus operation costs, and travel times. This allows potential passengers passengers to share shuttle trips, potentially leading to a reduction in the number of shuttles and the total variable cost associated with shuttle rides. The authors extend the traditional ODMTS by incorporating ridesharing into shuttle rides and introduce new fleet-sizing algorithms to determine the necessary number of shuttles to transport a set of passengers. Auad-Perez & Hentenryck (2022) propose a mixed integer programming (MIP) model in which the ODMTS incorporates pick-up and delivery routes, encapsulating ridesharing by grouping riders traveling to and from the same hub. The fleet-sizing optimization is framed as a minimum flow problem with covering constraints. To overcome computational issues brought about by the dense graph in the natural formulation, a reformulation operates on a sparse graph. The study is validated through a real case study of the public transit system in Michigan. When compared to the existing system, the proposed ODMTS design reduces costs by 35% and transit times by 38%.

Zheng & Pantuso (2023) focus on a ridesharing problem, which involves determining optimal routes for a fleet of vehicles to inorganically transport customers to a common destination via shared trips. The objective functions aim to minimize transportation costs and maximize service rates. Constraints include vehicles to leave their current positions at most once, each journey terminates at the station, and each customer is picked up at most once. Zheng & Pantuso (2023) present an evolutionary algorithm (EA) based on Pareto dominance to solve this problem. The algorithm simulates the evolution of species and operates with a population of individuals, where each individual represents a solution to the problem. The algorithm works iteratively, with each iteration referred to as a generation. In each generation, appropriate measures of fitness are used to select individuals that will reproduce and survive to the next generation. Then, an initialization phase is carried out,

which involves generating an initial population using a semi-random procedure. During the evolution phase, the algorithm generates offspring by applying crossover and mutation operators. The resulting offspring undergo a route rearrangement process to ensure that each vehicle takes the shortest distance to the destination. The evaluation step involves assessing the fitness of each individual in the population and offspring. Finally, the algorithm performs elite and survival selection. After testing the algorithm on real-life data, the results indicate that the algorithm is capable of delivering high-quality Pareto fronts.

On-demand ridesharing services have gained popularity in recent years, presenting new challenges and opportunities for optimization. Fielbaum et al. (2021) focus on optimizing pick-up and delivery locations in on-demand ridesharing systems in an inorganic way. They define an on-demand ridesharing service in which passengers with similar routes can share a vehicle. In their paper, they consider a system in which users might be requested to walk to and from nearby pick-up and delivery points. Their objective is to minimize routing cost, passenger walking cost and maximize passenger participation. Their consideration of alternative pick-up and delivery locations provides additional flexibility for both drivers and passengers. Fielbaum et al. (2021) develop specialized heuristics with local search to solve their MILP model. They propose four heuristics to accelerate the computation of the graph structure called groups-vehicles (GV). The first heuristic, called "optimization of the sequence and the pick-up and delivery (PUDO) points," employ an insertion heuristic, which inserts requests one by one into the vehicle in an arbitrary order. The second heuristic, "filtering vehicles," discards some feasible arcs in the GV-graph, reducing the number of trips. The third heuristic, "limiting sequences," uses a local search approach to decide the sequence of pick-ups and deliveries, limiting the number of sequences. Finally, the fourth heuristic, "searching for PUDO nodes," performs a local search to find a limited set of candidate PUDO points rather than an exhaustive computation over all possible points. To evaluate the effectiveness of their proposed method, Fielbaum et al. (2021) conduct computational experiments, comparing the performance of their approach to a baseline method without optimized walking locations. The results demonstrate that the proposed method can achieve reductions in passenger walking distances while maintaining comparable route efficiency for drivers.

Ghandeharioun & Kouvelas (2023) explore on-demand ridesharing by creating a real-time simulation framework and an optimization algorithm aimed at enhancing ridesharing operations. Their objective is to develop a matching algorithm to address the on-demand ridesharing task in a real-time setting. The model seeks to solve the problem of assigning requests to shuttles in a manner that satisfies both the operator's and passengers' needs. Each request is characterized by three crucial variables: desired pick-up time, desired pick-up location, and delivery location. Their approach incorporates dynamic congestion, ensuring that travel times on road segments are consistently updated throughout the simulation period. The study's contributions include the development of a modular real-time simulation framework, the framing of the ridesharing problem as a dynamic deterministic on-demand matching issue, the implementation of dynamic congestion, and the consideration of multiple stakeholders' objectives. By using a New York City taxi dataset, the authors demonstrate that their algorithm outperforms the current taxi fleet in terms of service rate. Moreover, the developed simulation framework provides insights into cost functions and operational policies.

Pelzer et al. (2015) present a partition-based matchmaking algorithm for dynamic ridesharing, to match passengers and drivers in real-time. The objective is to maximize mileage savings by sharing rides. The partition-based matchmaking algorithm divides the service area into smaller regions, enabling a more efficient search for feasible matches between pas-

sengers and drivers. Pelzer et al. (2015) also employ a two-phase approach to solve this problem. The first phase involves clustering the requests and vehicles in the partitions, while the second phase focuses on finding the optimal matches within each partition. The authors develop a heuristic algorithm based on local search and insertion techniques to find feasible matches. Their results demonstrate that the partition-based matchmaking algorithm is capable of providing high-quality solutions in a relatively short amount of time, highlighting its effectiveness and potential for application in dynamic ridesharing systems.

In a compelling response to the ongoing challenges of optimizing ridesharing, Lin et al. (2019) explore a probabilistic, demand-aware approach. The central issue in ridesharing optimization, as identified by the authors, lies in the joint optimization of request-vehicle assignment and routing. Traditional methodologies typically fail to consider the probabilities of future demands, leading to sub-optimal solutions. To address this, the authors suggest a probabilistic performance index intended to improve ride-sharing optimization by incorporating the probabilities of future requests. Lin et al. (2019) propose a non-linear and combinatorial optimization problem geared towards maximizing the expected number of new or ridesharing passenger pickups, considering the probability distributions of future requests. The optimization problem is subject to several constraints, such as vehicle capacity, passenger waiting time limits, and passenger time windows. To address the complexity of the problem, the authors propose a dual-subgradient heuristic approach. To handle the inherent complexity of the problem, the authors adapt an approach of reformulating the original problem into a linear-combinatorial one, which despite still being NP-hard, can produce an approximate solution with an approximation ratio of $1 - \frac{1}{e}$. In terms of the heuristic, the authors provide conditions under which it generates an optimal solution to the reformulated problem, and thus an approximate solution to the original problem. To evaluate the efficiency of their proposed approach, they conduct numerical experiments based on real-world travel request traces in Manhattan. The results indicate that their demand-aware solution substantially outperforms a conventional demand-oblivious scheme, boosting total passenger pickups by up to 46%. Furthermore, joint optimization at the fleet level led to 19% more pickups than individual vehicles optimizing separately.

Sun et al. (2020) provide insights and algorithmic solutions for optimization problems related to nonprofit peer-to-peer (P2P) ridesharing services. The authors aim to augment the societal benefits and cost-saving opportunities by enhancing ridesharing optimization. These services, particularly government or non-profit organization-run matching agencies, are becoming increasingly popular due to their efficiency over solo driving and public transit. The study examines two versions of the nonprofit ridesharing problem. The static version has all driver offers and rider requests received in advance, while the dynamic version accommodates real-time requests. The authors identify the main differences between these problems and pick-up and delivery problems with time windows (PDPTW), which include differences in vehicles and requests, divergent objectives, and the scale of the problem. The authors propose an exact solution algorithm for the static problem, which utilizes route-based variables in a set packing formulation. By leveraging the specific characteristics of the problem, they present an efficient graph-based approach to quickly generate all necessary vehicle routes. For larger problem instances, a column generation based heuristic approach is suggested. The dynamic version of the problem is tackled with two dynamic dispatching policies. These strategies are designed to generate near-optimal solutions quickly, which is particularly useful in a real-time context. The algorithms proposed by Sun et al. (2020) are found to solve very large problem instances optimally, even up to 600 drivers and 1,800 passengers in the case of the static problem.

The authors' dynamic dispatching policies produce near-optimal solutions for the dynamic problem.

In an attempt to address the underutilization of ridesharing, Hsieh (2020) propose a monetary incentive optimization approach. The study frames ridesharing underutilization as an issue of insufficient motivation, with prior literature typically focusing on non-monetary performance indices, such as travel distance and successful matches, which may not provide strong enough incentives for widespread adoption of ridesharing. To resolve this, the author suggest a monetary incentive performance indicator intended to increase ridesharing incentives. Hsieh (2020) propose a non-linear integer programming optimization problem to optimize monetary incentives in ridesharing systems. The monetary incentive optimization problem involves finding the set of passengers and drivers that maximize the ratio between the cost savings and the original costs. This problem is subject to various constraints, including capacity constraints for each vehicle, cost-saving constraints for each user, and the winning bid constraint for each driver. The author introduce several discrete metaheuristic algorithms including discrete variants of particle swarm optimization algorithms, differential evolution algorithms, and the firefly algorithm. In handling the constraints, the authors adopted a method based on biasing feasible solutions over infeasible ones. To assess the effectiveness of their proposed metaheuristic algorithms, they conducted experiments on several real-world test cases. The results indicated that the discrete variant of the cooperative, co-evolving particle swarm optimization algorithm was significantly more effective than the other metaheuristic algorithms in solving a constrained optimization problem with a non-linear objective function and binary decision variables. Hsieh (2020) provides insights into incentivizing ridesharing through monetary rewards. This approach represents an innovative direction in ridesharing research, focusing on economic incentives as a key driver of user behavior.

Li et al. (2023) explore urban traffic management in terms of ridesharing. They approach this by developing a generalized stochastic user equilibrium model, which formulates travelers' mode and route choice behavior. Recognizing the impact of ridesharing compensation on individual travel choices and, subsequently, the authors emphasize the importance of compensation pricing in ridesharing services as a strategic approach to alleviating traffic congestion. In particular, they tackle the decision-making problem of ridesharing compensation from the perspective of traffic managers and policy-makers who aim to minimize total travel cost and CO_2 emissions. Li et al. (2023) provide a mathematical model that addresses travelers' mode and route choice behavior based on the average occupancy rate of each ridesharing vehicle. Notably, the constraints on the number of ridesharing drivers and riders are formulated in terms of path flow, thereby avoiding the issue of one ridesharing rider having to switch between multiple vehicles for a single trip. In addition, they incorporate the waiting time caused by excessive passengers and the limited capacity of public transit vehicles into their modeling, demonstrating a more comprehensive understanding of the real-world transportation network. The model integrates travelers' mode and route choice behavior, allowing for endogenous determination of the number of travelers choosing each travel mode and each path based on stochastic user equilibrium principle. The model employ the non-dominated sorting genetic algorithm to generate a set of Pareto-optimal solutions for policy-makers.

Minimizing CO_2 emissions has become a vital concern in various sectors, including transportation. Bruck et al. (2017) explore this issue in the context of carpooling by focusing on reducing CO_2 emissions in a practical daily organic carpooling problem, used by companies to organize carpooling for its employees on a daily basis to reach a common destination. Their study aims to provide an environmentally friendly approach to carpooling, emphas-

izing the importance of reducing the carbon footprint of transportation activities. The authors propose a MILP model that takes into account several constraints, vehicle capacities, maximum detour restriction, and alternative meeting locations. The objective is to minimize total CO_2 emissions by optimizing carpooling assignments, route selection, and meeting point choices. Bruck et al. (2017) also incorporate a flexible user preference structure, allowing for the customization of different user preferences, such as detour tolerance, preferred vehicle types, and preferred meeting locations. To solve the problem, the authors suggest the use of a construction heuristic based on capacitated minimum spanning trees to find an initial solution. Then, the initial solution is passed to two local search algorithms: a swap operator that interchanges pairs of vertices and a move operator that moves vertices to any other position in any route. Lastly, another heuristic is invoked to verify that each step in the construction heuristics and the two local search algorithms maintains direct routes. The results indicate that the proposed approach can substantially reduce CO_2 emissions compared to individual commuting, demonstrating the environmental benefits of carpooling.

Stiglic et al. (2015) investigate the advantages of incorporating meeting points in ridesharing systems, highlighting their potential benefits in improving overall efficiency and user satisfaction. The authors recognize that traditional door-to-door ride-sharing systems can result in significant detours for drivers and increased travel times for passengers. Stiglic et al. (2015) propose the concept of meeting points, which serve as intermediate pick-up and delivery locations, facilitating more efficient routes and minimizing detours for drivers. The authors develop a MILP model to create organic ridesharing assignments and routing while considering the use of meeting points. The model takes into account constraints such as vehicle capacities, time windows, and maximum travel times for passengers. The objective is to maximize number of matched participants and driving distance savings. To solve this model, the authors create insertion heuristics where, for each driver, they try to match a compatible passenger to that driver's route. To demonstrate the benefits of meeting points, Stiglic et al. (2015) compare the results of their meeting point-based approach to a scenario without meeting points. The results show that incorporating meeting points in ridesharing systems leads to substantial reductions in travel times and detours for drivers, as well as improved user satisfaction due to shorter and more direct routes. Furthermore, the use of meeting points also contributes to lower fuel consumption and emissions, supporting the environmental objectives of ridesharing systems.

Hou et al. (2018) investigate a ridesharing problem concerning the matching of passengers to drivers and the selection of optimal routes for each vehicle. They address this issue by proposing both models and a heuristic approach for optimizing ride-matching and routing in ridesharing systems. The objective is to optimize the assignment of passengers to vehicles and the routing of the vehicles in order to minimize total travelling distance for both drivers and passengers. To efficiently solve the problem, Hou et al. (2018) propose a large neighborhood search (LNS) heuristic. The algorithm destroys the current solution by removing some demands from the routes of selected drivers. Then, it repairs the solution by finding appropriate drivers and inserting the unserved demands back into their routes. The motivation to use the LNS in their problem is its capability in searching a large scale of neighborhood in each iteration. This implies that the LNS has the potential to return a better local optimal solution and hence is more efficient than other neighborhood search methods. The authors evaluate the effectiveness of their proposed method through computational experiments on a set of randomly generated instances and real-world data sets. The results demonstrate that the LNS heuristic can provide high-quality solutions within reasonable computational times.

Smet (2021) explores a large-scale organic ridesharing problem involving flexible drivers and the use of alternate locations, proposing a metaheuristic approach to address this challenge. In this study, the flexibility of drivers is considered. A flexible driver is considered one who can drive their own car but is also willing to be a passenger and ride in another car. The consideration of alternate locations plays a significant role in the optimization process, providing additional options for pick-up and delivery points, and enabling more efficient and convenient solutions for both passengers and drivers. After generating an initial solution using a construction heuristic, Smet (2021) proposes the late acceptance hill climbing (LAHC) algorithm as a metaheuristic approach. This LAHC differs from the classic hill climbing algorithm in its acceptance criterion: rather than comparing the neighboring solution to the previous solution, it is compared against the solution of several iterations before. Smet (2021) introduces a direct solution representation defined by two data structures: a set of unassigned users and, for each driver, an ordered list of tuples representing the different stops in the driver's route. The algorithm uses a combination of intra-route and inter-route local neighborhood search operators to explore the solution space. These operators include change order, change location, delete user, add user, replace user, move user, and swap users. Smet (2021) conclude that the use of flexible drivers can notably reduce costs. However, extensive use of flexible drivers show diminishing returns.

In this thesis, our contribution lies in addressing certain aspects of ridesharing, which we comprehensively review and compare with existing literature in the field. Key points of our study include organic ridesharing, static conditions, constraints like time windows and maximum travel time, as well as objectives such as maximizing assigned riders and minimizing routing costs. We also introduce a unique real-life case study. The ridesharing literature is summarized in Table 2.1. Our specific contribution is highlighted in the final row of the table.

Table 2.1: Reviewed literature table. RS = Ridesharing, CP = Car pooling, VP = Vanpooling, Org. = Organic ridesharing, Inorg. = Inorganic ridesharing, Stat. = Static, dyn. = dynamic, Con. = Constraints, FL = Flexible locations, Cap = Minimum capacity constraint, MD = Maximum detour constraint, MW = Maximum waiting time, TW = Time window constraint, TT = Travel time constraint, MP = Maximum number of preferred passengers, CS = Minimize cost savings, CO_2 = Minimize CO_2 emissions, RC = Minimize routing cost, AR = Maximize assigned riders, US = User satisfaction, WT = Minimize waiting time

Article	**Problem**	**Org. vs. inorg.**	**Stat. vs. dyn.**	**Con.**	**FL**	**Objective**	**Case study**	**Solution method**
Agatz et al. (2012)	Review: RS	Org., inorg.	Stat., dyn.	Multiple	✓	Multiple		Multiple
Baldacci et al. (2004)	RS/CP	Inorganic	Static	TW, TT		RC, AR		Lagrangean column generation
Kaan & Olinick (2013)	RS/VP	Inorganic	Static	TW	✓	RC	✓	Insertion heuristics
He et al. (2023)	RS	Inorganic	Static	TW, TT, Cap		RC	✓	ALNS
Auad-Perez & Hentenryck (2022)	RS	Inorganic	Static	TW, TT		RC	✓	Graph reformulation
Zheng & Pantuso (2023)	RS	Inorganic	Static	TW		RC, AR	✓	Evolutionary algorithm
Fielbaum et al. (2021)	RS	Inorganic	Static		✓	RC, WT, AR	✓	Graph heuristics
Ghandeharioun & Kouvelas (2023)	RS	Inorganic	Dynamic	TW		US	✓	Simulation
Pelzer et al. (2015)	RS	Inorganic	Dynamic			RC	✓	Partition-based
Lin et al. (2019)	RS	Inorganic	Static	TW, MT		AR	✓	Probabilistic approach
Sun et al. (2020)	RS	Inorganic	Stat., dyn.	TW, MP		RC, CS	✓	Column generation
Hsieh (2020)	RS	Inorganic	Static			RC	✓	Swarm, evolutionary and firefly algorithms
Li et al. (2023)	RS	Inorganic	Static			RC, CO_2		Genetic algorithm
Bruck et al. (2017)	RS/CP	Organic	Static	MD	✓	CO_2	✓	Insertion heuristic and local search
Stiglic et al. (2015)	RS	Organic	Static	TW	✓	RC, AR	✓	Insertion heuristic
Hou et al. (2018)	RS	Organic	Static	TW		RC		LNS
Smet (2021)	RS	Organic	Static	TW	✓	RC, AR	✓	Late acceptance hill climbing
Our contribution	**RS**	**Organic**	**Static**	**TW, TT**	✓	**RC, AR**	✓	**ALNS**

2.2 Dial-A-Ride Problems and Pick-Up and Delivery Problems

Pelzer et al. (2015) provide a unique perspective on the relationship between ridesharing, dial-a-ride problems (DARP) and pick-up and delivery problems (PDP). They note that ridesharing can be viewed as an extension of both dial-a-ride and pick-up and delivery problems. In the context of transportation and logistics, dial-a-ride problems involve serving passengers who request transportation between specific origin and destination pairs, while pick-up and delivery problems deal with transporting goods from one location to another. In both cases, the challenge is to efficiently route the vehicles while satisfying various constraints, such as time windows, vehicle capacities, and service quality. Ridesharing extends these concepts by allowing multiple passengers with different origin and destination pairs to share the same vehicle, thus reducing the overall transportation costs and environmental impact. The connection between these three problems highlights the potential for adapting solution approaches and techniques from one domain to another.

2.2.1 Dial-A-Ride Problems

Ho et al. (2018) provide a comprehensive review of the dial-a-ride problem (DARP) literature and discuss recent developments in the field. The authors focus on various aspects of the problem, including problem features, objective functions, problem classification and solution methods. In the problem description, the authors outline the typical features of DARP, such as visit, time window, depot(s), trip, vehicle capacity, travel time, and route duration. These features are crucial for understanding the problem's constraints and requirements. The objective functions of DARPs are discussed in detail, with the most popular objectives being minimizing the service provider's operating costs and users' inconvenience metrics. Other more problem-specific objectives are also considered, such as optimizing passenger occupancy rate, cost-effectiveness, operator's profit, staff workload, and the reliability of the system. The authors also highlight the different approaches to addressing multiple objectives, including the weighted sum, lexicographic objective functions, and Pareto frontier approaches. The problem classification section divides DARPs into four categories based on the availability of information (static vs. dynamic) and the certainty of the information (deterministic vs. stochastic). The authors describe the differences between these categories and provide examples of how these classifications apply to various DARPs. Solution methods for the DARP and its variants have been categorized into several techniques. Exact methods, which are mainly focused on deterministic and static problems, include branch-and-cut, branch-and-price, and branch-and-price-and-cut algorithms. These methods provide high-quality solutions but may not be suitable for dynamic or stochastic problems. In contrast, heuristic and metaheuristic methods are more efficient for solving larger, more complex problems. Some popular techniques in this category are construction insertion heuristics, tabu search, simulated annealing, variable neighborhood search, large neighborhood search, and genetic algorithms. Each of these methods has its own strengths and weaknesses, but they all aim to find good quality solutions within a reasonable time frame.

Psaraftis (1980) present an exact dynamic programming solution to a specific variant of the DARP, known as the single vehicle many-to-many immediate request dial-a-ride problem. In this problem, a single vehicle is tasked with providing transportation services for a set of passengers between multiple pick-up and delivery points. The immediate request aspect of the problem implies that passenger requests are made in real-time, and

the vehicle must respond to these requests without prior knowledge of future demand. In addressing this problem, Psaraftis (1980) propose a dynamic programming approach that takes into account various practical constraints, such as vehicle capacity, time windows, and passenger waiting times. The objective is to minimize the total travel time or distance of the vehicle while providing satisfactory service to the passengers. The dynamic programming algorithm developed by Psaraftis (1980) computes the optimal solution by recursively solving subproblems, showing that the dynamic programming approach can solve the problem optimally within reasonable computational times. Psaraftis (1980) also discusses the potential extensions and limitations of the proposed method, suggesting that the dynamic programming approach could be adapted for other variations of the dial-a-ride problem or used in conjunction with other optimization techniques.

Madsen et al. (1995) investigate a DARP that includes time windows, multiple vehicle capacities, and multiple objectives. In this variant, capacitated vehicles must be routed to satisfy a set of transportation requests, each involving the pick-up of passengers from an origin location and their delivery to a destination location within specified time windows. The problem aims to simultaneously minimize multiple objectives, such as total travel distance, number of vehicles, total waiting time, and deviation from promised service. To solve this problem, Madsen et al. (1995) propose the REBUS heuristic algorithm, which is an insertion algorithm. The algorithm starts by considering the next unallocated passenger in a set of passengers and then examines each vehicle in a set of vehicles. For every vehicle, the algorithm generates all feasible insertions of the passenger in the schedule and calculates the change in the objective function. If a feasible insertion exists, the one with the minimum change in the objective function is chosen, and the passenger is inserted into the schedule and removed from the set of passengers. If no feasible insertion exists for the passenger, it is returned to a list of passenger that cannot be served. The algorithm continues until all passengers in the set have been considered.

Parragh et al. (2010) address a DARP with an objective of minimizing the total routing cost, while taking into consideration constraints such as duration limits, time windows and maximum travel times. In their study, Parragh et al. (2010) develop a variable neighborhood search (VNS) heuristic. The VNS algorithm begins with constructing an initial solution and iteratively improves it by exploring the neighborhoods and applying local search procedures. If the new solution improves upon the incumbent solution, it replaces the incumbent, and the search continues in the first neighborhood. If the new solution is worse, the incumbent remains unchanged, and the next neighborhood is used in the subsequent iteration. The algorithm proceeds until a stopping criterion, such as a maximum number of iterations or a time limit, is reached. Parragh et al. (2010) validate the effectiveness of their VNS heuristic through computational experiments on various benchmark instances of the DARP. The authors report new best results for 16 out of 20 benchmark instances, demonstrating the competitive performance of the proposed VNS algorithm.

Cordeau & Laporte (2003) address the static multiple-vehicle DARP. The DARP is characterized by the need to satisfy multiple requests using a fleet of vehicles, while considering constraints such as vehicle capacities, time windows, and travel time limitations. The objective is to minimize the total travel distance. To tackle the static multiple-vehicle DARP, Cordeau & Laporte (2003) propose a tabu search heuristic. The tabu search algorithm developed by the authors is based on three main components: a neighborhood search strategy, a tabu list to store recently visited solutions and prevent cycling, and an aspiration criterion to override the tabu status of a solution if it shows potential for improvement. The neighborhood search strategy uses different types of moves, such as in-

sertions, exchanges, and relocations, to generate new candidate solutions from the current solution. The results indicate that the proposed algorithm is effective in finding high-quality solutions for the static multiple-vehicle DARP within reasonable computational times. Furthermore, the authors demonstrate that their tabu search heuristic outperforms other existing heuristics and optimization algorithms in terms of solution quality and computational efficiency.

2.2.2 Pick-Up and Delivery Problems

Parragh et al. (2008) provide a comprehensive survey of the various aspects and challenges of pick-up and delivery problems (PDP). In their survey, Parragh et al. (2008) highlight the various constraints and objectives typically encountered in pick-up and delivery problems, such as vehicle capacities, time windows, route length, and service quality. They also emphasize the importance of considering both static and dynamic aspects of these problems, as real-world transportation and logistics scenarios often involve changing demand patterns and evolving operational conditions. The authors present a thorough review of solution methods for pick-up and delivery problems, ranging from exact algorithms, such as branch-and-bound and dynamic programming, to heuristic and metaheuristic approaches, including local search, tabu search, simulated annealing, and genetic algorithms. Parragh et al. (2008) provide an analysis of the strengths and weaknesses of each method, as well as their suitability for specific problem types and settings.

Ropke & Pisinger (2006) focus on the pick-up and delivery problem with time windows (PDPTW), a variant of the classic pickup and delivery problem where each transportation request involves picking up goods or passengers from an origin location and delivering them to a destination location within specified time windows. The objective of the PDPTW is to minimize the total travel distance or time while adhering to constraints such as vehicle capacities and time windows. In their research, Ropke & Pisinger (2006) propose an adaptive large neighborhood search (ALNS) heuristic to solve the PDPTW. The ALNS is a metaheuristic that has shown promising results in solving various combinatorial optimization problems. The ALNS heuristic developed by the authors is based on a combination of several destroy and repair operators. The adaptive nature of the ALNS lies in its ability to dynamically adjust the selection probabilities of these operators based on their recent performance, guiding the search process towards more promising areas of the solution space. The computational results show that the ALNS is capable of finding high-quality solutions. Furthermore, the authors highlight the versatility of the ALNS, suggesting that it can be easily adapted to address other variants of the pickup and delivery problem, like ridesharing problems.

Lu & Dessouky (2004) present an exact algorithm for solving the multiple vehicle pick-up and delivery problem without time windows (MVPDP), and the problem with time windows (MVPDPTW), which are variants of the classic pick-up and delivery problem in which multiple vehicles are available to serve a set of transportation requests. In the MVPDP, each request consists of picking up goods or passengers from an origin location and delivering them to a destination location. The problem entails assigning requests to vehicles, determining the route of each vehicle, and scheduling the pick-up and delivery activities to minimize the total travel cost and the fixed vehicle cost, while considering various constraints such as vehicle capacities and time windows. To address the MVPDP, Lu & Dessouky (2004) propose a branch-and-cut heuristic. In their approach, they employ bounding and branching strategies to effectively tackle the problem. At each node of the search tree, a linear relaxation is defined, and they treat certain constraints iteratively,

only adding them to the problem if they are violated when not included. This iterative procedure reduces the solution time for solving the LP-relaxation. Lu & Dessouky (2004) demonstrate the effectiveness of their branch-and-cut heuristic through computational experiments. The results show that the proposed method can find high-quality solutions for small to medium-sized instances within reasonable computational times. The authors also discuss potential extensions and limitations of their approach, suggesting that their heuristic could be adapted for other variants of the pick-up and delivery problem.

Ropke et al. (2006) investigate the pick-up and delivery problem with time windows (PDPTW). The objective is to minimize the total travel distance or time. The authors make three contributions in their paper. First, they propose two formulations for the PDPTW and the closely related DARP, which impose a limit on the elapsed time between pick-up and delivery of a request. Second, Ropke et al. (2006) introduce new valid inequalities that combine the pick-up and delivery structure of the problem with either vehicle capacity constraints or time window constraints. These inequalities are used to strengthen the formulations, making them more effective in solving the problem. Finally, the authors develop branch-and-cut algorithms that incorporate the new formulations and valid inequalities to solve the PDPTW and the DARP. The branch-and-cut algorithms merge the branch-and-bound method with cutting plane techniques, generating valid inequalities or cuts to enhance the linear programming relaxation. Through computational experiments on various test instance sets, Ropke et al. (2006) demonstrate the effectiveness of their approach. The results indicate that their algorithms can optimally solve instances with up to eight vehicles and 96 requests (194 nodes), showcasing the potential of the new formulations, valid inequalities, and branch-and-cut algorithms for addressing the PDPTW and the DARP.

Lu & Dessouky (2006) address the pick-up and delivery problem with time windows (PDPTW) by presenting a insertion-based construction heuristic to solve the problem. In the PDPTW, capacitated vehicles must be routed to satisfy a set of transportation requests, each involving the pickup of goods or passengers from an origin location and their delivery to a destination location within specified time windows. The objective is to minimize the total travel distance and fixed vehicle cost. The authors' proposed insertion-based construction heuristic consists of two main components: a parallel insertion procedure and a sequential insertion procedure. The parallel insertion procedure is employed to build initial solutions by iteratively inserting requests into the routes of multiple vehicles simultaneously. The requests are inserted according to a cost-based criterion, which prioritizes those with the least increase in the total cost of the solution. The sequential insertion procedure refines the initial solution by examining each request in sequence and reinserting it into the best position within the existing routes. This process continues until no further improvement in the solution can be achieved.

Dahle et al. (2019) focus on the pick-up and delivery problem with time windows and occasional drivers (PDPTW-OD), an extension of the classic PDPTW. In this variant, a company not only has its own fleet of vehicles to service requests but may also use the services of occasional drivers. These drivers are willing to take a detour to serve one or more transportation requests for a small compensation. The objective is to minimize the routing costs of the regular vehicles plus the compensation given to the ODs. Furthermore, the model incorporates different compensation schemes for occasional drivers and personal threshold constraints that reflect the minimum acceptable compensation for each occasional driver. Dahle et al. (2019) discuss the impact of occasional drivers and different ways of compensating them. The authors present two mathematical formulations of the problem: reduction tests and symmetry-breaking constraints. They also propose a model

for optimizing the parameters in the compensation schemes. The computational study conducted by the authors shows that utilizing occasional drivers may lead to large cost savings, with the model yielding cost savings of about 10–15%. Additionally, they discuss the effects of different compensation schemes and the role of personal threshold constraints in reflecting the expectations of occasional drivers.

Wang et al. (2020) present a multi-objective version of the multitrip pickup and delivery problem with time windows and manpower planning (MTPDPTW-MP) to describe a real-life healthcare problem that originated from the application of public hospitals in Hong Kong, China. These transportation services are provided to disabled or elderly patients between their residences and clinics. The problem requires designing ambulance routes that satisfy a series of constraints, as well as staff assignment. The authors introduce a multi-objective MTPDPTW-MP (MO-MTPDPTW-MP) with three objectives to be minimized: the number of unserved requests, the total traveling cost, and the workload deviation. These objectives take into account the interests of customers, the hospital, and the staff. Constraints include vehicle capacity, time windows, maximum travel times, and a staff break time constraint, where each staff member must take a 30-minute break after each trip. To solve the MO-MTPDPTW-MP, Wang et al. (2020) propose a multi-objective iterated local search algorithm with adaptive neighborhood selection (MOILS-ANS). The algorithm generates a diverse set of alternative solutions for decision-makers to meet their requirements. MOILS-ANS employs problem-specific neighborhood structures and an adaptive neighborhood selection strategy to explore the search space. The experimental results show that the proposed MOILS-ANS substantially outperforms other multi-objective algorithms. The authors also analyze the nature of objective functions and the properties of the problem. In addition, the study compares the proposed MOILS-ANS with the previous single-objective algorithm, highlighting the benefits of multi-objective optimization.

Chapter 3

Problem Definition

In this chapter, the Static Ridesharing Routing Problem with Flexible Locations (SRRPFL) is introduced. The SRRPFL involves multiple individuals participating in a ridesharing system, taking on one of two roles: a driver or a passenger. A driver is a participant who drives their own car and is willing to pick up and deliver passengers. A passenger, on the other hand, is a participant who does not drive a car and therefore needs to be picked up and delivered by a driver. If no drivers are available to pick up the passenger, they must find alternative ways to reach their destination.

The SRRPFL is solved daily in a static setting. Each day, all drivers and passengers submit their travel information, which is used as the input to the SRRPFL. The travel information for passengers includes an origin and a destination location, a time window and a maximum travel time. Furthermore, passengers submit how far they are willing to travel to/from alternative pick-up/delivery locations, which are referred to as candidate locations. The same travel information is included for the drivers, except for candidate locations, but the capacity of their car is added. The origin location for both drivers and passengers is considered their residence, while the destination location is considered their workplace. Ridesharing can occur from a passenger's home to their workplace or on a portion of this route between two candidate locations.

The set of candidate locations is split into candidate pick-up locations and candidate delivery locations. Each passenger has an individual set of candidate pick-up and delivery locations. Candidate pick-up locations are locations that a passenger can travel to and be picked up by a driver. Candidate delivery locations are locations where a driver can deliver a passenger before the passenger travels to their destination. These candidate locations are used to reduce the total travel time for drivers, increase the degree of ridesharing, provide more flexibility, and create additional possible solutions or combinations of ridesharing routes. Each time window includes the earliest and the latest time a driver or passenger can be at their destination location. The maximum travel time is the maximum time each driver or passenger is willing to spend traveling from their origin location to their destination location. If a passenger is either picked up or delivered at a candidate location, the time it takes to travel to or from the candidate location is included in the calculation of the total travel time. For drivers, the capacity is the number of free seats they have in their car or the number of passengers they are willing to pick up.

The SRRPFL determines the optimal route for each driver and passenger. This process involves making four sets of decisions displayed in Table 3.1:

Table 3.1: Four sets of decisions involved in the SRRPFL

1. Determine which passengers to be assigned to a suitable driver or, alternatively, establish that no driver is available to pick them up
2. Determine the pick-up location for each designated passenger and inform the respective driver
3. Determine the delivery locations for each driver and passenger
4. Determine the order of pick-up and delivery for each passenger

The SRRPFL makes all decisions while keeping in mind its two primary objectives. The first objective is to maximize the number of passengers participating in ridesharing. This is achieved by measuring the number of passengers picked up by each driver, ultimately reducing the number of cars on the road. The second objective is to minimize the total travel time for all drivers. This ensures that the most effective route is chosen, with the same number of passengers being picked up.

The rational for the second objective, minimizing drivers' travel time, is the expectation that it will also indirectly benefit passengers' experience. By reducing travel time for drivers, the overall system becomes more efficient, which can lead to a decrease in passengers' travel time as well. As drivers spend less time on the road, passengers' overall travel time may be reduced as a consequence. In this way, minimizing the total travel time for drivers inherently contributes to the minimization of travel time for passengers.

There are also several underlying assumptions for the SRRPFL. Firstly, it operates in a static manner where drivers and passengers submit their travel information ahead of time, and the SRRPFL determines the optimal solution for a specific time interval. The solution remains fixed for that time interval. Secondly, a trip is split into two distinct phases: a pick-up phase and a delivery phase. During the pick-up phase, all the pick-ups are made, and all deliveries are made during the delivery phase. This means that all pick-ups are made before the first delivery, and after the first delivery, there can be no more pick-ups.

Figure 3.1 illustrates an example problem of the SRRPFL with a corresponding possible solution. This example problem considers the problem with one driver and two passengers. Driver 1 starts driving from its own origin in *OD1* in Blomvåg. Driver 1 picks up Passenger 1 before picking up Passenger 2. The SRRPFL dictates that both passengers must travel to one of their candidate locations. Passenger 1 travels to *CP2P1* in Ågotnes from Sollsvika, while Passenger 2 travels to *CP3P2* in Tellnes from Skogsvåg. After the pick-up phase, Passenger 2 is the first to be delivered. Passenger 2 is delivered at *CD1P2* in Fyllingsdalen, before it travels to its destination, *DP2* in Bønes. Further, Passenger 1 is delivered at its destination, *DP1* in Laksevåg. Lastly, Driver 1 travels to its destination, *DD1* in Bergen sentrum.

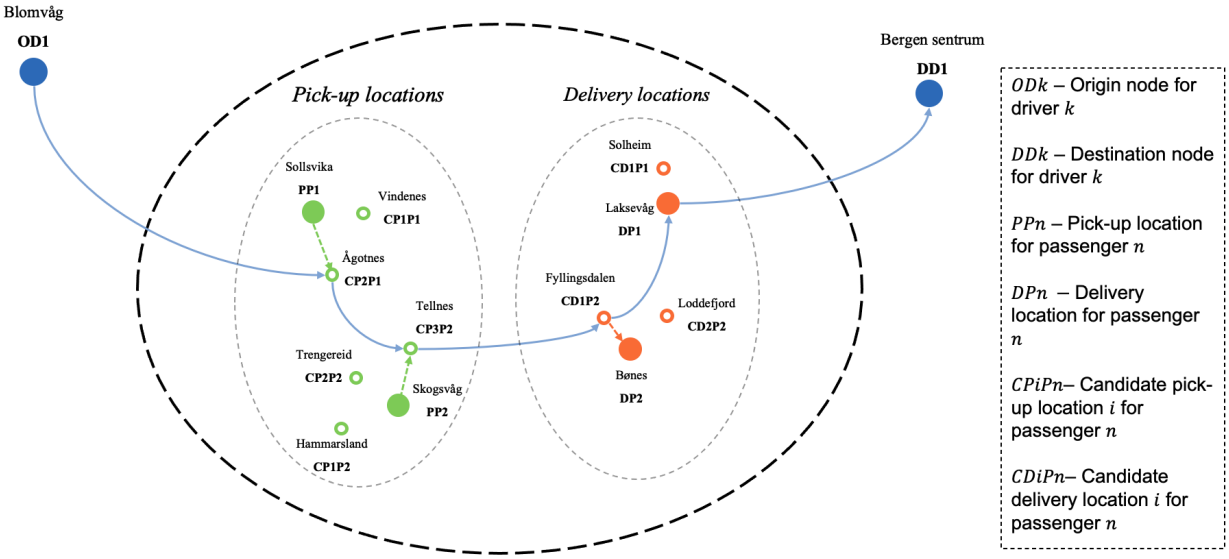


Figure 3.1: Illustration of an example SRRPFL problem featuring one driver and two passengers, with location names based on the case study *(Chapter 6)*. The driver picks up and delivers passengers at specified locations before reaching their final destination

Chapter 4

Mathematical Model

This chapter presents the mathematical model of SRRPFL presented in this thesis, and is formulated as a mixed integer programming (MIP) model. Section 4.1 provides the mathematical notations used, including sets, parameters, and decision variables. Section 4.2 presents descriptions of the objective functions and constraints of the SRRPFL.

4.1 Mathematical Notation

The following section presents the notation used in our mathematical model. We introduce the sets in Subsection 4.1.1. The parameters are introduced in Subsection 4.1.2. The decision variables are introduced in Subsection 4.1.3.

4.1.1 Sets

The set of drivers is defined as $\mathcal{D}$, where $|\mathcal{D}|$ is the total number of drivers. This set also represents drivers' origin locations, i.e., their residencies. The set of passengers is represented as the set of different passenger origin locations, $\mathcal{P}^P$, i.e., the residency of each passenger. Each passenger also has a designated destination location, e.g., their workplace, and this is represented as the set $\mathcal{P}^D$. The size of the set $\mathcal{P}^P$ is equal to the size of $\mathcal{P}^D$, and represents the total number of passengers N. Each passenger has a set of candidate pick-up locations and a set of candidate delivery locations. The set of candidate pick-up locations is represented as $\mathcal{M}_i^P$ and the set of candidate delivery locations is represented as $\mathcal{M}_i^D$ for passenger $i \in \mathcal{P}^P$.

Each node in the model is represented as a combination of an origin/destination location for a driver/passenger and a candidate pick up/delivery location. The set of passenger pick-up nodes is represented as $(i, m) \in \mathcal{N}^P$, where $i \in \mathcal{P}^P$ and $m \in \mathcal{M}_i^P$. The representation of the origin of a passenger $i \in \mathcal{P}^P$ as a pick-up node is $(i, 0)$. The set of passenger delivery nodes is represented as $(j, n) \in \mathcal{N}^D$, where $j \in \mathcal{P}^D$ and $n \in \mathcal{M}_i^D$, where i is the corresponding origin location to j. The representation of the destination of a passenger $j \in \mathcal{P}^D$ as a delivery node is $(j, 0)$. Furthermore, the set $\mathcal{N}^R$ represents all ridesharing nodes where a passenger is either picked up or delivered. Note that the origin node $o(k)$ and destination node $d(k)$ for driver $k \in \mathcal{D}$ is not a part of this set, thus, $\mathcal{N}^R = \mathcal{N}^P \cup \mathcal{N}^D$. $o(k)$ and $d(k)$ are described in more detail in Subsection 4.1.2. Lastly, $\mathcal{A}_k$ is defined as the set of all possible arcs driver $k \in \mathcal{D}$ can travel. An arc for a specific driver k is defined

as the direct travel between two nodes (i, m) and (j, n), where $(i, m) \in \mathcal{N}^{\mathcal{R}} \cup \{o(k)\}$ to node $(j, n) \in \mathcal{N}^R \cup \{d(k)\}$. Table 4.1 summarizes all sets used in the SRRPFL.

Table 4.1: All sets defined for the mathematical formulation of the SRRPFL

Notation	Explanation
$\mathcal{D}$	Set of drivers $k \in \{0, 1, ..., \lvert\mathcal{D}\rvert\}$
$\mathcal{P}^P$	Set of passenger origin locations $i \in \{1, 2, ..., N\}$
$\mathcal{P}^D$	Set of passenger destination locations $j \in \{N+1, N+2, ..., 2N\}$
$\mathcal{M}_i^P$	Set of candidate pick-up locations $m \in \{0, 1, ..., \lvert\mathcal{M}_i^P\rvert\}$ for passengers $i \in \mathcal{P}^P$
$\mathcal{M}_i^D$	Set of candidate delivery locations $n \in \{0, 1, ..., \lvert\mathcal{M}_i^D\rvert\}$ for passengers $i \in \mathcal{P}^P$
$\mathcal{N}^P$	Set of passenger pick-up nodes $\mathcal{N}^P \in \{(i, m) \lvert i \in \mathcal{P}^P, m \in \mathcal{M}_i^P\}$
$\mathcal{N}^D$	Set of passenger delivery nodes $\mathcal{N}^D \in \{(j, n) \lvert j \in \mathcal{P}^D, n \in \mathcal{M}_j^D\}$
$\mathcal{N}^R$	Set of all ridesharing nodes $\mathcal{N}^R = \mathcal{N}^P \cup \mathcal{N}^D$
$\mathcal{N}$	Set of all nodes $\mathcal{N} = \mathcal{N}^R \cup \{o(k)\} \cup \{d(k)\}$
$\mathcal{A}_k$	Set of possible arcs driver $k \in \mathcal{D}$ can travel. An arc represents the direct travel from node $(i, m) \in \mathcal{N}^R \cup \{o(k)\}$ to node $(j, n) \in \mathcal{N}^R \cup \{d(k)\}$

Figure 4.1, inspired by Figure 3.1 from Chapter 3, visually demonstrates the sets $\mathcal{N}^P$, $\mathcal{N}^D$, and $\mathcal{N}^R$. In this illustration, the set $\mathcal{N}^P$ encompasses all pick-up nodes a driver k, originating from $o(k)$, can visit. Subsequently, after collecting all passengers from the nodes in $\mathcal{N}^P$, the driver delivers them to their respective delivery nodes in the set $\mathcal{N}^D$. Finally, the set $\mathcal{N}^R$ all ridesharing nodes of the process of picking up and delivering passengers.

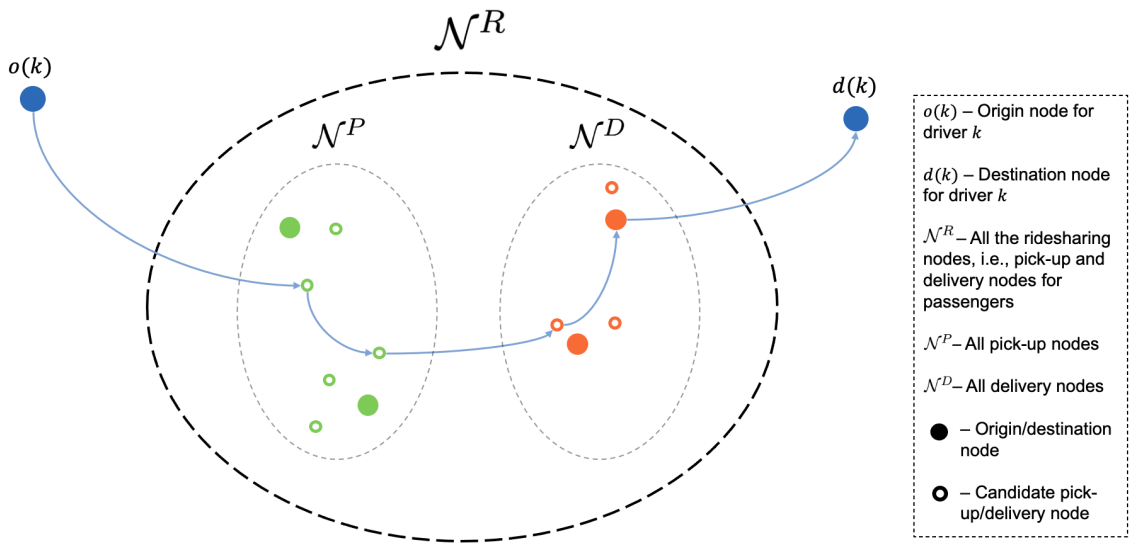


Figure 4.1: Visual illustration of $\mathcal{N}^P$, $\mathcal{N}^D$, and $\mathcal{N}^R$

Figure 4.2 visually illustrates the sets $\mathcal{M}_i^P$ and $\mathcal{M}_i^D$. In this example, when the driver is en

route to pick up the first passenger, the driver has three distinct candidate pick-up nodes ($\mathcal{M}_1^P$) to choose from: one node representing the passenger's origin and two candidate nodes. For the second passenger, the driver has the option to pick up at four different nodes ($\mathcal{M}_2^P$): the origin node or at any of the three other candidate pick-up nodes. When it comes to the first delivery, the driver can choose between the destination node and two candidate delivery nodes ($\mathcal{M}_2^D$). Lastly, for the final delivery, the driver can either deliver at the destination location or at one candidate delivery node ($\mathcal{M}_1^D$).

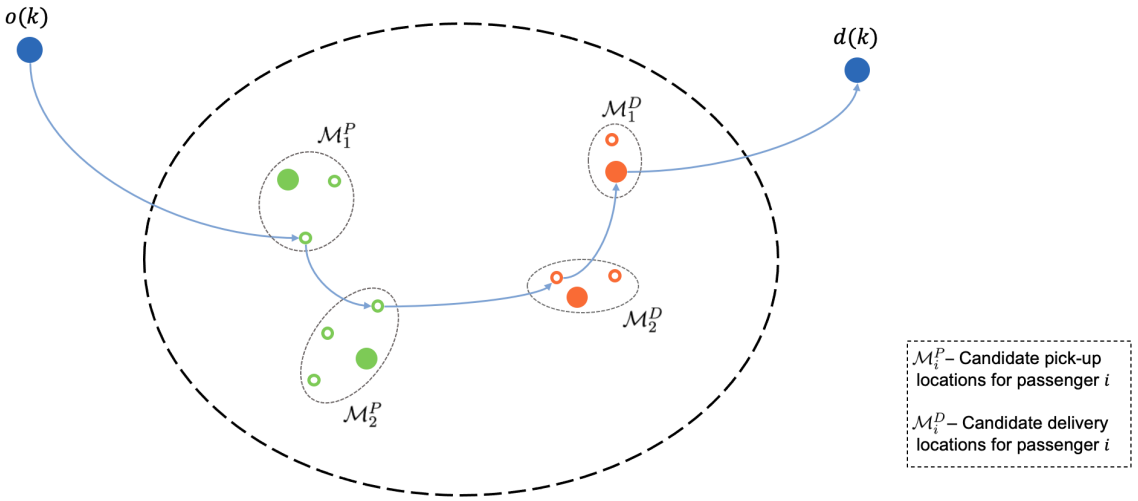


Figure 4.2: Visual illustration of $\mathcal{M}_i^P$ and $\mathcal{M}_i^D$. Here, we have passenger $i = 1$ and $i = 2$

4.1.2 Parameters

As mentioned in Subsection 4.1.1, we define the origin node and destination node as $o(k)$ and $d(k)$ for driver $k \in \mathcal{D}$, respectively. Drivers will start their ride at their origin and end the ride at their destination node. Thus, drivers will not have any other candidate locations.

T_{imjn}^D is defined as the direct travel time between node $(i, m) \in \mathcal{N}^R \cup \{o(k)\}$ and node $(j, n) \in \mathcal{N}^R \cup \{d(k)\}$. T_{im}^C is defined as the travel time between a passenger's origin/destination $i \in \mathcal{P}^P \cup \mathcal{P}^D$ and a candidate pick up/delivery location $m \in \mathcal{M}_i^P \cup \mathcal{M}_i^D$. T_{im}^C is zero minutes for passengers picked up at their origin location ($(i, 0), i \in \mathcal{P}^P$), or delivered at their destination location ($(j, 0), j \in \mathcal{P}^D$). T_k^M is defined as the maximum travel time driver/passenger $k \in \mathcal{D} \cup \mathcal{P}^P$ is willing to spend from their origin to their destination locations.

Each driver/passenger has a time window for when they can arrive at their destination location. The earliest and latest arrival time for driver/passenger $k \in \mathcal{D} \cup \mathcal{P}^P$ is represented as $\underline{A}_k$ and $\overline{A}_k$, respectively. Finally, Q_k represents the number of passengers each driver $k \in \mathcal{D}$ can transport. Table 4.2 summarizes all parameters.

Figure 4.3 provides a closer look at Figure 4.2, focusing on the second and third nodes the driver visits. In Figure 4.3, a node is depicted with parentheses (i, m) for readability purposes. Additionally, the node representations are displayed. The origin node for passenger 2 is depicted as node $(2, 0)$, and the candidate pick-up nodes $\{(2, 1), (2, 2), (2, 3)\}$. Moreover, the destination node for passenger 2 is represented as node $(3, 0)$, and the candidate delivery nodes $\{(3, 1), (3, 2)\}$. $T_{(2,3)}^C$ denotes the time it takes for passenger 2 to

Table 4.2: All parameters defined for the mathematical formulation of the model

Notation	Explanation
$o(k)$	Origin node for driver $k \in \mathcal{D}$
$d(k)$	Destination node for driver $k \in \mathcal{D}$
T^D_{imjn}	Direct travel time from node $(i,m) \in \mathcal{N}^R \cup \{o(k)\}$ to node $(j,n) \in \mathcal{N}^R \cup \{d(k)\}$
T^C_{im}	Direct travel time between origin/destination location for passenger $i \in \mathcal{P}^P$ and its candidate pick up/delivery location $(i,m) \in \mathcal{N}^P \cup \mathcal{N}^D$
T^M_k	Maximum travel time for driver/passenger $k \in \mathcal{D} \cup \mathcal{P}^P$
$\underline{A}_k$	Earliest arrival time at destination for driver/passenger $k \in \mathcal{D} \cup \mathcal{P}^P$
$\overline{A}_k$	Latest arrival time at the destination for driver/passenger $k \in \mathcal{D} \cup \mathcal{P}^P$
Q_k	Maximum capacity for driver $k \in \mathcal{D}$

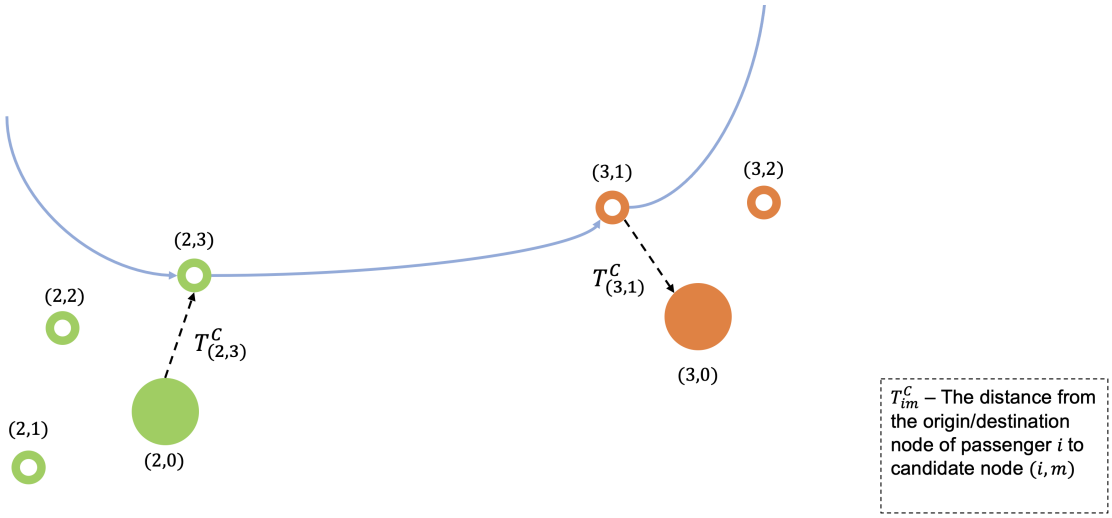


Figure 4.3: Visual representation of nodes and T^C_{im}

travel from their origin location to the candidate location $(2,3)$. $T^C_{(3,1)}$ signifies the time it takes for passenger 2 to travel from the candidate delivery location $(3,1)$ to their destination location.

4.1.3 Decision Variables

The binary variable x_{kimjn}, called the ridesharing arc, is 1 if driver $k \in \mathcal{D}$ travels directly from node $(i,m) \in \mathcal{N}^R$ to node $(j,n) \in \mathcal{N}^R$, and 0 otherwise. Note that this variable is only for ridesharing nodes, $\mathcal{N}^R$. The binary variable x^S_{kim}, called the start arc, is 1 if driver $k \in \mathcal{D}$ drives from its origin $o(k)$ to a candidate pick up node $(i,m) \in \mathcal{N}^P$, and 0 otherwise. The binary variable x^E_{kjn}, called the end arc, is 1 if driver $k \in \mathcal{D}$ travels from candidate delivery node $(j,n) \in \mathcal{N}^D$ to its destination node, $d(k)$, and 0 otherwise. The binary variable x^{OD}_k is 1 if driver $k \in \mathcal{D}$ travels directly from its origin, $o(k)$, to its destination, $d(k)$, and 0 otherwise. In other words, if $x^{OD}_k = 1$, it means that driver $k \in \mathcal{D}$ does not pick up any passengers. The binary variable y_{im} is 1 if passenger $i \in \mathcal{P}^P$ is picked up/delivered at candidate location $(i,m) \in \mathcal{N}^R$, and 0 otherwise. The binary variable z_{ki} is 1 if driver $k \in \mathcal{D}$ picks up passenger $i \in \mathcal{P}^P$, and 0 otherwise. The continuous variable t_{kim} defines the time when driver $k \in \mathcal{D}$ leaves node $(i,m) \in \mathcal{N}$.

Based on the defined variables, particularly the ridesharing arc, start arc, and end arc variables, the SRRPFL is characterized as an arc-flow model. Table 4.3 summarizes all variables in this problem.

Table 4.3: All decision variables defined for the mathematical formulation of the SRRPFL

Notation	Explanation
x^S_{kim}	1 if driver $k \in \mathcal{D}$ travels from its origin location $o(k)$ to a pick up node $(i,m) \in \mathcal{N}^P$, 0 otherwise
x_{kimjn}	1 if driver $k \in \mathcal{D}$ travels directly between ridesharing nodes $(i,m) \in \mathcal{N}^R$ and $(j,n) \in \mathcal{N}^R$, 0 otherwise
x^E_{kjn}	1 if driver $k \in \mathcal{D}$ travels from a delivery node $(j,n) \in \mathcal{N}^D$ to its destination location $d(k)$, 0 otherwise
x^{OD}_k	1 if driver $k \in \mathcal{D}$ travels directly from its origin location $o(k)$ to its destination location $d(k)$, 0 otherwise
y_{kim}	1 if driver $k \in \mathcal{D}$ picks up/delivers passenger $i \in \mathcal{P}^P$ at node $(i,m) \in \mathcal{N}^P \cup \mathcal{N}^D$, 0 otherwise
z_{ki}	1 if driver $k \in \mathcal{D}$ picks up passenger $i \in \mathcal{P}^P$, 0 otherwise
t_{kim}	The time driver/passenger $k \in \mathcal{D} \cup \mathcal{P}^P$ leaves node $(i,m) \in \mathcal{N}$

Figure 4.4 illustrates the variables x^S_{kim}, x_{kimjn}, x^E_{kjn}, and x^{OD}_k. In this figure, the use of commas and parentheses to represent nodes is used for readability purposes. In this example, driver 1 can either choose to pick up passengers and participate in ridesharing ($x^S_{1,(1,1)} = 1$) or drive directly to its destination without picking up any passengers ($x^{OD}_1 = 1$). If the driver decides to pick up passengers, the x_{kimjn} variables are set to 1 ($x_{1,(1,1),(2,3)} = x_{1,(2,3),(3,1)} = x_{1,(3,1),(4,0)} = 1$). After all passengers have been delivered, the driver exits the ridesharing and travels directly to their destination location from the last delivery node ($x^E_{1,(4,0)} = 1$). This representation helps to demonstrate the decision-making process and the different choices available to the driver within the ridesharing model.

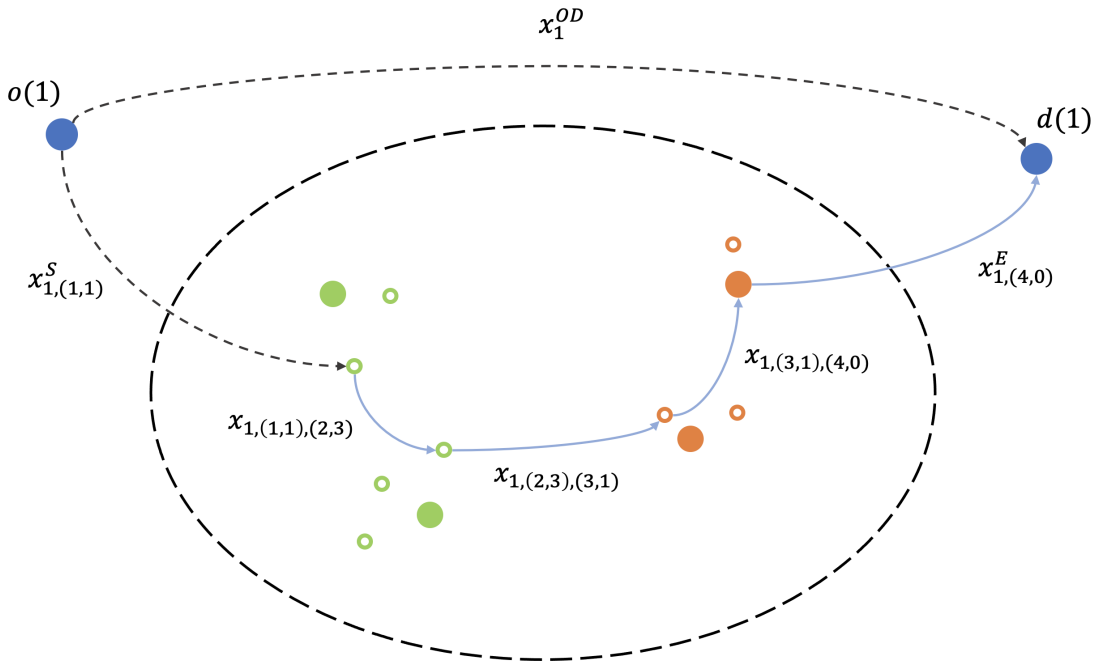


Figure 4.4: Visual representation of decision variables x^S_{kim}, x_{kimjn}, x^E_{kjn}, and x^{OD}_k

4.2 Model Formulation

This section presents the mathematical formulation of the SRRPFL. We start by introducing the objective functions in Subsection 4.2.1. Then, we introduce the constraints in the model in the proceeding subsections.

4.2.1 Objective functions

The model is formulated as a multi-objective optimization problem with two objective functions, which is solved using the lexicographic method. Lexicographic ordering is a method for ranking solutions in multi-objective optimization based on a predefined order of importance of the objectives. The objectives are first sorted based on the values of the objectives in that order, such that the most important objective is considered first, and ties are broken based on the second most important objective, and so on. The first objective aims to maximize the number of passengers that are picked up by drivers, and is formulated in Objective (4.1).

$$\max\ Z_1 = \sum_{k \in \mathcal{D}} \sum_{i \in \mathcal{P}^P} z_{ki} \tag{4.1}$$

The second objective aims to minimize the total travel time for drivers, and is formulated in Objective (4.2). This considers the time drivers leave their origin location and arrive at their destination location.

$$\min\ Z_2 = \sum_{k \in \mathcal{D}} (t_{k,d(k)} - t_{k,o(k)}) \tag{4.2}$$

4.2.2 Routing Constraints

Constraints (4.3) ensure that each driver must leave its origin exactly once.

$$\sum_{(i,m)\in\mathcal{N}^P} x^S_{kim} + x^{OD}_k = 1, \quad k \in \mathcal{D} \tag{4.3}$$

Next, Constraints (4.4) ensure that each driver arrives at its destination node exactly once.

$$\sum_{(j,n)\in\mathcal{N}^D} x^E_{kjn} + x^{OD}_k = 1, \quad k \in \mathcal{D} \tag{4.4}$$

Constraints (4.5) ensure that if a node, (i, m), is visited either as the first ridesharing node or following any arbitrary pick-up node, then driver k must also depart from the node (i, m).

$$x^S_{kim} + \sum_{(j,n)\in\mathcal{N}^P} x_{kjnim} = \sum_{(j,n)\in\mathcal{N}^R} x_{kimjn}, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^P \tag{4.5}$$

Constraints (4.6) ensure that the final delivery node, (j, n), to be visited prior to the driver's destination node, $d(k)$, is indeed traversed following another arbitrary ridesharing node, (i, m). Together, Constraints (4.5) and (4.6) guarantee that a driver who visits a ridesharing node subsequently departs from that node.

$$x^E_{kjn} + \sum_{(i,m)\in\mathcal{N}^D} x_{kjnim} = \sum_{(i,m)\in\mathcal{N}^R} x_{kimjn}, \quad k \in \mathcal{D}, (j,n) \in \mathcal{N}^D \tag{4.6}$$

Constraints (4.7) ensure that each passenger is picked up by a maximum of one driver. If a passenger, i, does not participate in the ridesharing, it will not be picked up by any driver, k.

$$x^S_{kjn} + \sum_{(i,m)\in\mathcal{N}^P} x_{kimjn} - y_{kjn} = 0, \quad k \in \mathcal{D}, (j,n) \in \mathcal{N}^P \tag{4.7}$$

Constraints (4.8) ensure that each passenger is delivered by at most one driver. Consequently, a passenger, i, who does not participate in ridesharing, will not be delivered by a driver, k.

$$x^E_{kjn} + \sum_{(i,m)\in\mathcal{N}^D} x_{kjnim} - y_{kjn} = 0, \quad k \in \mathcal{D}, (j,n) \in \mathcal{N}^D \tag{4.8}$$

Constraints (4.9) ensure that each passenger is picked up or delivered at no more than one candidate location.

$$\sum_{k\in\mathcal{D}} \sum_{m\in\mathcal{M}^P_i\cup\mathcal{M}^D_i} y_{kim} \leq 1, \quad i \in \mathcal{P}^P \tag{4.9}$$

Constraints (4.10) ensure that the variable z_{ki} is equal to 1 if driver k picks up passenger i.

$$z_{ki} = \sum_{m \in \mathcal{M}_i^P} y_{kim}, \quad k \in \mathcal{D}, i \in \mathcal{P}^P \tag{4.10}$$

4.2.3 Coupling and Precedence Constraints

The Coupling Constraints (4.11) ensure that any driver who picks up a passenger also delivers the passenger to one of their candidate delivery nodes.

$$\sum_{m \in \mathcal{M}_i^P} y_{kim} = \sum_{n \in \mathcal{M}_i^D} y_{k,N+i,n}, \quad k \in \mathcal{D}, i \in \mathcal{P}^P \tag{4.11}$$

The Precedence Constraints (4.12) ensure that no deliveries are allowed to occur before pickups.

$$t_{kim} + T^D_{i,m,N+i,n} z_{ki} - t_{k,N+i,n} \leq 0, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^P, n \in \mathcal{M}_i^D \tag{4.12}$$

4.2.4 Time Constraints

Constraints (4.13) ensure that the departure time at node (j,n) is not earlier than the sum of the departure time from the preceding node (i,m) and the travel time between the nodes. The combination of Constraints (4.13) and (4.14) prohibits drivers from waiting at any of these node. M_{kimjn} is the big-M, representing a large value.

$$t_{kim} + T^D_{imjn} - t_{kjn} - M_{kimjn}(1 - x_{kimjn}) \leq 0, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^R, (j,n) \in \mathcal{N}^R \tag{4.13}$$

$$t_{kim} + T^D_{imjn} - t_{kjn} + M_{kimjn}(1 - x_{kimjn}) \geq 0, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^R, (j,n) \in \mathcal{N}^R \tag{4.14}$$

Constraints (4.15) ensure that the departure time at node (i,m) is not earlier than the sum of the departure time from the preceding node $o(k)$ and the travel time between the nodes. The combination of Constraints (4.15) and (4.16) prohibits drivers from waiting at any of these node. $M_{k,o(k),i,m}$ is the big-M, representing a large value.

$$t_{k,o(k)} + T^D_{o(k),i,m} - t_{kim} - M_{k,o(k),i,m}(1 - x^S_{kim}) \leq 0, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^P \tag{4.15}$$

$$t_{k,o(k)} + T^D_{o(k),i,m} - t_{kim} + M_{k,o(k),i,m}(1 - x^S_{kim}) \geq 0, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^P \tag{4.16}$$

Constraints (4.17) ensure that the departure time at node $d(k)$ is not earlier than the sum of the departure time from the preceding node (i,m) and the travel time between the

nodes. The combination of Constraints (4.17) and (4.18) prohibits drivers from waiting at any of these node. $M_{k,i,m,d(k)}$ is the big-M, representing a large value.

$$t_{kim} + T^D_{i,m,d(k)} - t_{k,d(k)} - M_{k,i,m,d(k)}(1 - x^E_{kim}) \leq 0, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^D \tag{4.17}$$

$$t_{kim} + T^D_{i,m,d(k)} - t_{k,d(k)} + M_{k,i,m,d(k)}(1 - x^E_{kim}) \geq 0, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^D \tag{4.18}$$

Constraints (4.19) ensure that the departure time at node $d(k)$ is not earlier than the sum of the departure time from the preceding node $o(k)$ and the travel time between the nodes. The combination of Constraints (4.19) and (4.20) prohibits drivers from waiting at any of these node. $M_{k,o(k),d(k)}$ is the big-M, representing a large value.

$$t_{k,o(k)} + T^D_{o(k),d(k)} - t_{k,d(k)} - M_{k,o(k),d(k)}(1 - x^{OD}_k) \leq 0, \quad k \in \mathcal{D} \tag{4.19}$$

$$t_{k,o(k)} + T^D_{o(k),d(k)} - t_{k,d(k)} + M_{k,o(k),d(k)}(1 - x^{OD}_k) \geq 0, \quad k \in \mathcal{D} \tag{4.20}$$

Constraints (4.21) ensure that all drivers arrive within their predefined time windows, while Constraints (4.22) ensure the same condition for all passengers.

$$\underline{A}_k \leq t_{k,d(k)} \leq \overline{A}_k, \quad k \in \mathcal{D} \tag{4.21}$$

$$\underline{A}_i z_{ki} \leq t_{k,N+i,0} + T^C_{N+i,n} y_{kim} \leq \overline{A}_i z_{ki}, \quad k \in \mathcal{D}, i \in \mathcal{P}^P, n \in \mathcal{M}^D_i \tag{4.22}$$

Constraints (4.23) ensure that the total travel time does not exceed the maximum travel time for drivers, while Constraints (4.24) ensure the same for passengers. $M_{k,i,0}$ is the big-M, representing a large value.

$$t_{k,d(k)} - t_{k,o(k)} \leq T^M_k, \quad k \in \mathcal{D} \tag{4.23}$$

$$t_{k,N+i,0} - t_{k,i,0} \leq T^M_i + M_{k,i,0}(1 - z_{ki}), \quad k \in \mathcal{D}, i \in \mathcal{P}^P \tag{4.24}$$

Constraints (4.25) ensure that if a passenger is picked up at node (i,m), the time of pickup at node (i,m) is equal to or later than the departure time for a passenger from its origin node $(i,0)$, plus the travel time between the nodes. Constraints (4.26) ensure that if a passenger is delivered at node (j,n), the arrival time at its destination node $(j,0)$ is equal to or later than the time the passenger is delivered at node (j,n), plus the time between the nodes.

$$t_{k,i,0} \leq t_{kim} - T^C_{im} y_{kim}, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^P \tag{4.25}$$

$$t_{k,j,0} \geq t_{kjn} + T^C_{jn} y_{kjn}, \quad k \in \mathcal{D}, (j,n) \in \mathcal{N}^D \tag{4.26}$$

4.2.5 Capacity Constraints

Constraints (4.27) ensure that the total number of passengers in a driver's car never exceeds the driver's maximum capacity.

$$\sum_{i \in \mathcal{P}^P} z_{ki} \leq Q_k, \quad k \in \mathcal{D} \tag{4.27}$$

4.2.6 Binary, Continuous and Non-Negativity Constraints

$$x^S_{kim} \in \{0,1\}, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^P \tag{4.28}$$

$$x_{kimjn} \in \{0,1\}, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^R, (j,n) \in \mathcal{N}^R \tag{4.29}$$

$$x^E_{kjn} \in \{0,1\}, \quad k \in \mathcal{D}, (j,n) \in \mathcal{N}^D \tag{4.30}$$

$$x^{OD}_k \in \{0,1\}, \quad k \in \mathcal{D} \tag{4.31}$$

$$y_{kim} \in \{0,1\}, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^P \cup \mathcal{N}^D \tag{4.32}$$

$$z_{ki} \in \{0,1\}, \quad k \in \mathcal{D}, i \in \mathcal{P}^P \tag{4.33}$$

$$t_{kim} \geq 0, \quad k \in \mathcal{D} \cup \mathcal{P}^P, (i,m) \in \mathcal{N} \tag{4.34}$$

Chapter 5

Adaptive Large Neighborhood Search

This section introduces an adaptive large neighborhood search (ALNS) approach to solve the SRRPFL. Section 5.1 provides an overview of the ALNS for the SRRPFL. In Section 5.2, the representation of a solution to the problem is presented. Following that, Section 5.3 presents the construction heuristic used to create an initial solution. Section 5.4 covers the destroy and repair operators, and operator selection. Section 5.5 describes the acceptance criterions, including the simulated annealing acceptance criterion. Section 5.6 details the local neighborhood search (LS) heuristic and its integration with the ALNS. Finally, Section 5.7 describes a set-partitioning formulation of SRRPFL and its integration with the ALNS.

5.1 Overview of ALNS

The choice of using an adaptive large neighborhood search (ALNS) heuristic is motivated by several factors. Firstly, the SRRPFL is a highly constrained combinatorial optimization problem with large solution spaces, rendering exact methods computationally infeasible for real-world instances. Metaheuristics, such as ALNS, offer an effective approach to exploring the solution space, handling complex constraints, and providing high-quality solutions within reasonable computation times. Secondly, ALNS has a proven track record in successfully solving similar problems, such as vehicle routing problems and pick-up and delivery problems. The flexibility of ALNS allows for the development of tailored destroy and repair operators specific to the SRRPFL, enabling the exploration of diverse areas of the solution space and promoting the discovery of better solutions. Lastly, the adaptivity of the ALNS enables the algorithm to autonomously adjust the selection probabilities of the operators throughout the search process, ensuring a more effective exploration of the solution space. Consequently, the ALNS is a well-suited solution approach for the SRRPFL, considering its flexibility, adaptivity, and ability to handle complex constraints while delivering high-quality solutions.

The proposed ALNS heuristic is inspired by the research of Ropke & Pisinger (2006), who developed an ALNS for the pick-up and delivery problem with time windows. Building upon their foundational work, this chapter presents a solution technique that incorporates both destroy and repair operators tailored for the SRRPFL. This method explores the

solution space by iteratively destroying and repairing solutions. An acceptance criterion is used to determine whether to accept a repaired solution or not. In addition, once a solution is accepted, a local search (LS) heuristic is integrated within the ALNS to help refine and improve the solutions generated by the ALNS. The LS allows for exploring a broader solution space while also exploiting local improvements in the vicinity of a solution. The integration of a set-partitioning formulation, hereby referred to as the route combination problem (RCP), within the ALNS serves to enhance its effectiveness in discovering superior solutions. The RCP is later described in Section 5.7. The proposed solution method is illustrated in Figure 5.1.

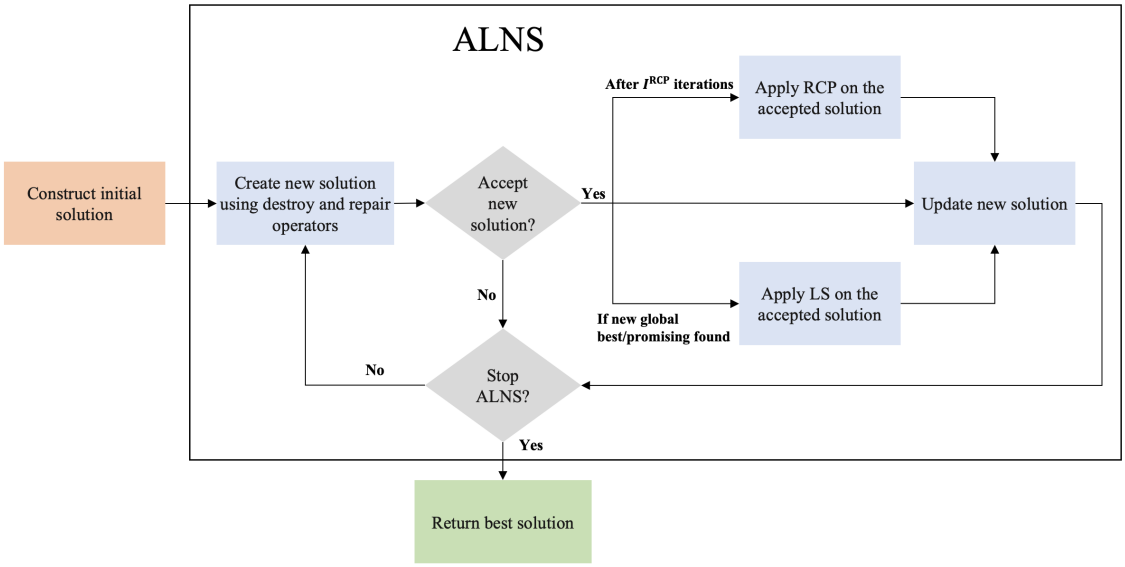


Figure 5.1: Flowchart representing the processes in the ALNS heuristic. LS = Local search. RCP = Route combination problem

The ALNS algorithm is outlined in Algorithm 1. Initially, the algorithm sets the current solution x by first constructing a feasible intial solution, further described in Section 5.3. This is followed by setting the global best solution, x^*, to the current solution x. It also initializes the destroy and repair operators Ω_i^- and Ω_i^+ and their associated operator weights in segment m, $p_{i,m}^-$ and $p_{i,m}^+$, further described in Section 5.4.

In each iteration, the algorithm selects a destroy and a repair operator. It then generates a candidate solution x' by applying the selected destroy and repair operators to the current solution x. The acceptance probability P^{SA} is computed using the simulated annealing criterion, further described in Section 5.5.

If the candidate solution x' is accepted as the new global best solution, a local search is applied to explore locally optimal solutions involving x', further described in Section 5.6. This step investigates the immediate neighborhood of the accepted solution to identify potential improvements and further refine the solution quality. Then, the current solution x and global best solution x^* are updated accordingly, along with their objective values.

If the candidate solution x' is better than the current solution x, but worse than the global best solution x^*, a local search is applied if x' is considered a *promising solution*. A *promising solution* x' is one with an objective value within a predefined threshold, $\delta\%$, of the current best global solution's objective value, later explained in Section 5.6. If the candidate solution x' is accepted as the new global best solution after the local search, both the current solution x and the global best solution x^* are updated accordingly, along

with their objective values. Otherwise, the current solution x is updated to the candidate solution x'.

If the candidate solution x' is worse than the current solution x but is accepted by the acceptance criterion, then the current solution x is updated to the candidate solution x'. Otherwise, the candidate solution x' is rejected.

Based on how the candidate solution x' is accepted or rejected, the accumulated scores of the employed destroy operator s_i^- and repair operator s_i^+ are updated. After completing a set of iterations, the algorithm updates the destroy and repair operator weights for the next segment, $p_{i,m+1}^-$ and $p_{i,m+1}^+$. The destroy and repair operators are chosen based on their previous performances. If a specific operator repeatedly results in new and better solutions, it is used more frequently. After every I^{RCP} iterations, the algorithm solves the RCP on $x^{'}$ to potentially find a new global best solution, further described in Section 5.7. The ALNS algorithm terminates after all iterations are completed, and the global best solution x^* is returned as the output.

Algorithm 1 ALNS

Input: Total number of ALNS iterations (I^{ALNS}), number of segment iterations (I^S), number of set-partition iterations (I^{RCP})

Set current solution x by construction of a feasible solution *(Section 5.3)*
Set global best solution, $x^* \leftarrow x$
Set global best objective, $f(x^*) \leftarrow f(x)$
Set current segment, $m \leftarrow 1$
Initialize destroy operators Ω_i^-, operator weights $p_{i,m}^-$ and operator scores s_i^- *(Subsection 5.4.3)*
Initialize repair operators Ω_i^+, operator weights $p_{i,m}^+$ and operator scores s_i^+ *(Subsection 5.4.3)*
for *iteration* $= 1$ to I^{ALNS} **do**
    Select destroy and repair operators $d \in \Omega_i^-$ and $r \in \Omega_i^+$ using $p_{i,m}^-$ and $p_{i,m}^+$ *(Subsection 5.4.3)*
    Generate a candidate solution $x^{'}$ from the current solution x using d and r
    Generate acceptance probability P^{SA} by simulated annealing (SA) *(Section 5.5)*
    if $x^{'}$ is accepted as the new global best solution **then**
        Apply local search for improving candidate solution $x^{'}$ *(Section 5.6)*
        $x \leftarrow x^{'}$
        $x^* \leftarrow x^{'}$
        $f(x^*) \leftarrow f(x^{'})$
        Reward operators d and r and update s_i^- and s_i^+
    else if $f(x) < f(x^{'}) < f(x^*)$ **then**
        Apply local search for improving candidate solution $x^{'}$ if promising *(Section 5.6)*
        if local search finds a new global best solution **then**
            $x \leftarrow x^{'}$
            $x^* \leftarrow x^{'}$
            $f(x^*) \leftarrow f(x^{'})$
        else
            $x \leftarrow x^{'}$
        end if
        Reward operators d and r and update s_i^- and s_i^+
    else if $f(x^{'}) < f(x)$ and accepted by SA through P^{SA} **then**
        $x \leftarrow x^{'}$
        Reward operators d and r and update s_i^- and s_i^+
    else if $f(x^{'}) < f(x)$ and rejected by SA through P^{SA} **then**
        Penalize operators d and r and update s_i^- and s_i^+
    end if
    if I^S iterations have passed since last weight update **then**
        Update weights $p_{i,m+1}^-$ and $p_{i,m+1}^+$ to be used in segment $m+1$
        Update current segment, $m \leftarrow m+1$
    end if
    if I^{RCP} iterations have passed **then**
        Solve RCP on $x^{'}$ to find a new global best solution *(Section 5.7)*
    end if
end for

Output: x^*

5.2 Solution Representation

A solution to the SRRPFL is depicted using a matrix-like structure. The matrix comprises $|\mathcal{D}|$ rows, one for each driver/vehicle. The columns' indices indicate the sequence in which nodes are visited within a particular driver route. The leftmost node in a row is the driver origin node, which is the first node in a specific route. Further, the rightmost node in a row is the driver destination node.

Figure 5.2 illustrates a solution representation for a case involving two drivers and five passengers. Each cell consists of a tuple, (i, m), representing a node $(i, m) \in \mathcal{N}$. The sequence of nodes for each driver represents the driver's route. All nodes are assigned positions, s, to illustrate the visiting order. The nodes between a driver origin and the midpoint of the route (excluding the driver origin node) are pick-up nodes, marked with the color green. The nodes between the midpoint of the route and the driver destination node (excluding the driver destination node) are delivery nodes, marked with the color orange. For example, for driver *D1*, nodes $(3, 1)$ and $(5, 3)$ are pick-up nodes, as they are positioned between the driver origin node $(1, 0)$ and the midpoint of the route. Nodes $(8, 0)$ and $(10, 4)$ are the delivery nodes, as they are situated between the midpoint of the route and the driver destination node $(13, 0)$.

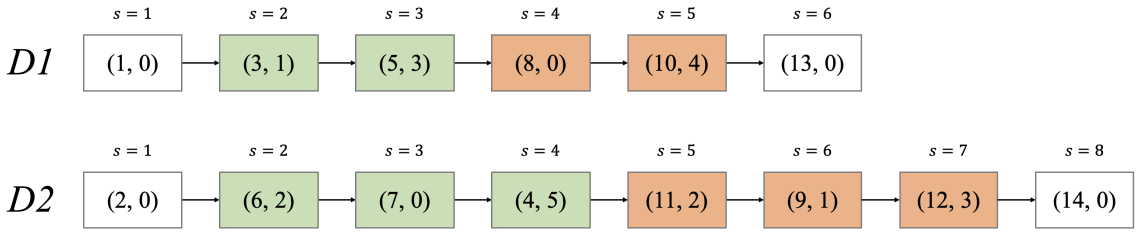


Figure 5.2: Solution representation of an instance with two drivers and five passengers. The leftmost numbers represent the drivers. Driver *D1* picks up two passenger and driver *D2* picks up three passengers.

5.3 Construction of Initial Solution

This section presents the insertion heuristic used for the construction of the initial solution. The pseudocode of the heuristic is illustrated in Algorithm 2. This construction heuristic provides a structured approach to generating initial feasible solutions by systematically exploring potential passenger assignments to drivers.

The algorithm takes the set of all drivers and the set of all passengers as input and aims to construct routes for the drivers by iteratively assigning passengers to them. Thus, the algorithm starts with an initial solution, which consists of empty driver routes where no passengers are picked up, and modifies it to create the initial solution.

The algorithm processes each passenger one by one and tries to find the best driver route in which the passenger can be inserted at the lowest additional cost. It does this by examining all possible positions for the pick-up and delivery nodes of the passenger into each driver's route, while ensuring the feasibility constraints, i.e., driver capacity, maximum travel times, and time windows, are met. Then, the algorithm calculates the cost increase between the new route with the inserted pick-up and delivery nodes, and the current route of the driver. This cost increase is used to compare different passenger assignments and

to determine the most cost-effective way of accommodating the passenger in the driver's route while maintaining feasibility.

If a feasible insertion is found for a passenger, the relevant driver's route is updated with the new pick-up and delivery nodes. If no feasible insertion is found, the passenger is added to the set of unassigned passengers. The algorithm repeats this process for all passengers, and once completed, it outputs the final routes for each driver and the set of unassigned passengers.

Algorithm 2 Insertion Construction Heuristic

Input: Set of all drivers ($\mathcal{D}$), set of all passengers ($\mathcal{P}^P$)

Initialize set of unassigned passengers, L
Initialize all drivers routes, R
for $i \in \mathcal{P}^P$ **do**
    Set lowest cost increase, $c^* \leftarrow \infty$
    Set best driver, $k^* \leftarrow None$
    Set best route, $r^* \leftarrow None$
    for $k \in \mathcal{D}$ **do**
        for all candidate pick-up locations for i **do**
            Initialize all pick-up positions in $R(k)$
            for all pick-up positions in $R(k)$ **do**
                if feasible pick-up node insertion **then**
                    Create a temporary route with inserted pick-up node
                    Initialize all delivery positions in the temporary route
                    for all candidate delivery locations for i **do**
                        for all delivery positions in the temporary route **do**
                            if feasible delivery node insertion **then**
                                Create a final route $r^{'}$ with the pick-up and delivery node
                                Calculate cost increase $c^{'}$ between $r^{'}$ and $R(k)$
                                if $c^{'} < c^*$ **then**
                                    $c^* \leftarrow c^{'}$
                                    $k^* \leftarrow k$
                                    $r^* \leftarrow r^{'}$
    if k^* is $None$ **then**
        Add i to L
    else
        Update $R(k^*)$ with r^*

Output: R and L

5.4 Large Neighborhood Search

In an adaptive large neighborhood search (ALNS) approach, a collection of both destroy and repair opertors are utilized to explore the solution space efficiently. During each iteration of the algorithm, one destroy operator and one repair operator are selected and applied to the current solution. The selection process is guided by a roulette wheel mechanism, which is determined by adaptive weights that are dynamically updated throughout the search process. Further details on the implementation of this approach, along with the selection and updating of heuristic weights, are provided in the subsequent subsections.

Subsection 5.4.1 details the destroy operators that destroy portions of the current solution to explore alternative paths. Next, Subsection 5.4.2 details the repair operators that reconstruct partial solutions. Subsection 5.4.3 describes the probabilistic approach for selecting destroy and repair operators based on adaptively updated weights.

5.4.1 Destroy Operators

In this subsection, we introduce a variety of operators in the set Ω_i^- designed to destroy portions of the current solution, thereby creating opportunities for the exploration of alternative solution paths. These operators contribute to breaking free from local optima and facilitate the discovery of potentially better solutions. Each destroy operator employs a distinct strategy for removing elements from the solution, offering a diverse range of approaches for perturbing the current state.

The number of passengers removed during each destruction phase is determined by a random factor, calculated using a specific function. This function computes the removal quantity based on a random value within a predefined range (γ), which is then multiplied by the total number of passengers. The selection of this range is further explained in Subsection 7.2.1. The result is rounded up to the nearest integer to ensure that at least one passenger is removed. This approach allows for a variable number of passengers to be removed, providing flexibility and adaptability in the search process. Furthermore, each destroy operator returns updated driver routes and a list of unassigned passengers.

Random Removal

The random removal operator operates by removing a specified number of passengers to remove from the driver routes. This iterates through the desired number of removals, randomly selecting a driver and a passenger within that driver's route. The chosen passenger, along with its pick-up and delivery nodes, is removed from the driver's route.

Worst Deviation Removal

The worst deviation removal operator aims to identify and remove passengers that cause the highest additional cost in the solution by evaluating the impact they have on their respective driver's route. The operator calculates the deviation for each passenger by comparing the objective values of the driver's route with and without the passenger. The passengers causing the greatest deviation are considered to have the most impact on the solution quality. The deviation values for each passenger are sorted, and the passengers with the highest impact on the solution quality are identified. These passengers are then removed from their respective driver's route.

Relatedness Removal

The relatedness removal operator aims to identify and remove passengers that have a high degree of relatedness in terms of their impact on a driver's route. The general idea behind this operator is that by removing passengers with a higher degree of relatedness, it becomes easier to rearrange them when reintegrating them into the solution, potentially leading to an improved solution. On the contrary, passengers that are very different from each other in terms of their impact on a driver's route might be difficult to reorganize, which may result in them being reinserted into their initial positions through a repair operator.

The process begins by selecting a random driver from the list of available drivers. For the selected driver, a random seed passenger is chosen in the driver's route, and the relatedness between this seed passenger and all other passengers in the route is calculated. The relatedness between a seed passenger, i, and another passenger, j, is defined using a relatedness measure $R(i, j)$. The relatedness measure consists of finding passengers that have close pick-up and delivery nodes to the seed passenger's pick-up and delivery nodes. Thus, calculating the total travel time difference between the seed passenger's pick-up and delivery nodes and the other passenger's pick-up and delivery nodes. The relatedness measure is given by Equation (5.1).

$$R(i, j) = T^D_{imjn} + T^D_{i+N,m,j+N,n} \tag{5.1}$$

T^D_{imjn} is the direct travel time between pick up nodes (i, m) and (j, n) (*Subsection 4.1.2*). $T^D_{i+N,m,j+N,n}$ is the direct travel time between delivery nodes $(i + N, m)$ and $(j + N, n)$. Once the relatedness values for all passengers have been calculated, the passengers are sorted based on their relatedness to the seed passenger, from least to most related. The operator then removes a specified number of passengers with the lowest relatedness values.

Spread Removal

The spread removal operator identifies and removes passengers whose pick-up locations have the greatest minimum distance to other passengers within a driver's route. To achieve this, the operator starts by selecting a random driver and then calculates the minimum distance between the pick-up location of each passenger in the selected driver's route and the pick-up locations of all other passengers in the same route. The passenger with the greatest minimum distance is removed from the driver's route.

The motivation behind the spread removal operator lies in its ability to diversify the search process by targeting passengers with spatially distant pick-up locations. By removing passengers with the greatest minimum distance to other passengers within a driver's route, the operator encourages the exploration of alternative route configurations that may lead to more efficient solutions. Furthermore, by focusing on spatially distant passengers, the spread removal operator may indirectly contribute to the reduction of total travel time.

Cluster Removal

The cluster removal operator finds and removes clusters of passengers in a driver's route who are in close geographical proximity. This is similar to relatedness removal which also focuses on close geographical proximity. The algorithm of the cluster removal is described in Algorithm 3. The cluster removal operator employs the k-means clustering algorithm, with k set to 2, to divide the passengers on a driver's route into two distinct groups based on their geographical locations. This process is executed for each driver individually. At the outset, the algorithm initializes the positions of two centroids (the centers of each cluster).

Each passenger is then assigned to the nearest centroid. After the initial assignment, an iterative process begins. During each iteration, every passenger is reassigned to the closest centroid, and then the positions of the centroids are updated based on the newly assigned passengers. This cycle repeats until the assignments of passengers to centroids remain constant between iterations, indicating that the optimal clustering (with the least total distance from passengers to their respective centroids) has been achieved.

Upon establishing the passenger clusters, the operator randomly selects one cluster and removes all passengers within that cluster from the driver's route, including their associated delivery nodes. This operation is performed for each driver, ensuring that the total number of passengers removed does not exceed the predefined number of removals. By removing entire clusters, the cluster removal operator reduces the probability of passengers being reinserted into their initial positions during the repair phase.

Algorithm 3 Cluster Removal

Input: Set of routes, number of centroids, k

for $d \in \mathcal{D}$ **do**
    Initialize the position of each of the k centroids (c_1, c_2 ... c_k)
    Assign each passenger to the centroid closest to them
    repeat
        for passenger i in d's route **do**
            Assign i to the centroid closest to them
        end for
        Update the position of each centroid based on the assigned passengers
    until assignments of passengers to centroids do not change
    Select one of the k clusters randomly and remove it
end for

Output: Destroyed routes and list of unassigned passengers

5.4.2 Repair Operators

In this subsection, we present repair operators in the set Ω_i^+ which serve as an essential component in the process of reconstructing and enhancing the partial solutions generated by the destroy operators. The primary function of a repair operator is to systematically reintroduce the unassigned passengers back into the solution. Each repair operator uses a unique approach for rebuilding the solution, allowing for the exploration of various solution paths and the potential identification of more improved solutions.

Insertion Repair

The insertion repair operator is an adaptation of the construction heuristic, incorporating a degree of randomization to facilitate a more diverse exploration of the solution space.

Like the construction heuristic, this repair operator is responsible for reintroducing removed passengers into the driver's routes by iteratively assigning passengers to them. The removed passengers are first shuffled randomly to introduce variation in the order of insertion. For each passenger, the operator then iterates over all drivers and possible pick-up locations, but with a randomized order of evaluation. For each feasible pick-up insertion, a temporary route is created, and the operator proceeds to examine all possible delivery locations, once again in a randomized order.

For each feasible delivery insertion, the operator calculates the cost increase associated

with the insertion of the passenger at the given pick-up and delivery positions. It then selects the insertion with the lowest cost increase, updating the driver's route accordingly. If no feasible insertion is found for a passenger, they are added to the list of unassigned passengers.

By incorporating randomization into the passenger order, pick-up positions, and delivery positions, the insertion repair operator might create a search process that explores a wider range of potential solutions, increasing the likelihood of finding better-quality routes.

Regret-k Repair

The regret-k repair operator is a repair method that takes into account the regret value associated with different insertion options when reintroducing removed passengers into the driver's routes. The regret value represents the "lost opportunity" or "regret" of not choosing those alternative positions.

In the regret-k repair process, for each unassigned passenger, the algorithm first identifies the top k insertion positions that result in the lowest cost increase for the route. It then calculates the regret value for these positions by summing the cost differences between the best insertion position and the subsequent $k-1$ alternatives. The goal is to find the passenger with the maximum regret value and reinsert them into the route. The higher the regret value, the more important it is to choose the best insertion position for that passenger, as it implies that alternative positions would result in higher costs. This process continues until all unassigned passengers are considered for reinsertion. By calculating the regret value, this method prioritizes the reinsertion of passengers that have the greatest impact on overall solution quality.

In mathematical terms, the regret value of the insertion is given by Equation (5.2).

$$max\{\sum_{j=1}^{k-1}(c_i - c_{i+j})\} \tag{5.2}$$

Here, k denotes the number of insertion positions considered, c_i signifies the cost increase for the best insertion position (ith position) for the passenger, and c_{i+j} refers to the cost increase for the subsequent jth alternative insertion position.

Maximum Capacity Insertion Repair

The maximum capacity insertion repair operator aims to insert passengers into vehicles with the highest remaining capacity. In this process, drivers are organized in descending order based on their remaining capacity. Starting with the driver possessing the largest remaining capacity, the operator attempts to insert passengers previously removed by destroy operators into that current vehicle. If none of the removed passengers can be accommodated, the procedure moves on to the driver with the next largest remaining capacity and repeats the process. Once a passenger is successfully inserted into a route, the route is updated, and the process starts again with the updated capacities. This approach continues until either all removed passengers have been inserted or no driver can accommodate the remaining passengers.

5.4.3 Choosing Destroy and Repair Operators

During the adaptive large neighborhood search, one destroy operator and one repair operator need to be chosen for each iteration. Ropke & Pisinger (2006) employed a probabilistic approach to select the destroy and repair operators, in which this thesis uses the same procedure. This selection process is based on adaptively updating the weights of these operators throughout the search.

Initially, all destroy operators Ω_i^- and repair operators Ω_i^+ are assigned equal and non-zero weights $p_{i,m}^-$ and $p_{i,m}^+$ for the initial segment m. Then, the ith operators are selected by the roulette wheel selection mechanism. This mechanism calculates the probability ϕ_i of choosing the ith operator. Equation (5.3) illustrates the probability ϕ_i^- of choosing the ith destroy operator, and Equation (5.4) illustrates the probability ϕ_i^+ of choosing the ith repair operator. For both equations, $p_{i,m}^-$ and $p_{i,m}^+$ are the weights for operator i in segment m.

$$\phi_i^- = \frac{p_{i,m}^-}{\sum_{k \in \Omega^-} p_{k,m}^-} \tag{5.3}$$

$$\phi_i^+ = \frac{p_{i,m}^+}{\sum_{k \in \Omega^+} p_{k,m}^+} \tag{5.4}$$

After the selection, the destroy and repair operators are applied to the current solution. The weights of these operators are updated after each iteration based on their performance. Inspired by Ropke & Pisinger (2006), a scoring mechanism is employed to either reward or penalize the operators, with the score calculated based on the improvement in solution quality achieved by each operator. To do this, four different parameters (σ_i) are defined to reward or penalize the destroy and repair operators based on their performance. These parameters signify varying levels of rewards or penalties, contingent on the outcome of the operators' application in the current iteration:

σ_1 : Reward given to both destroy and repair operators when the new candidate solution is the best global solution found so far.

σ_2 : Reward given to both destroy and repair operators when the new candidate solution has not been accepted before and is better than the current solution, but not better than the best global solution found so far.

σ_3 : Reward given to both destroy and repair operators when the new candidate solution has not been accepted before and is worse than the current solution, but is accepted by the simulated annealing acceptance criterion.

σ_4 : Penalty given to both destroy and repair operators when the new candidate solution is worse than the current solution and is rejected based on the simulated annealing acceptance criterion.

The rationale behind these parameters is to promote the selection of operators that contribute to finding improved solutions. Operators that consistently result in solution quality improvements receive higher rewards and are selected more often, while those that do not contribute to improvements are penalized and selected less frequently.

The weights of the destroy and repair operators are then updated after each segment, taking into account both past performance and the most recent score. As mentioned, let $p^-_{i,m}$ and $p^+_{i,m}$ represent the weights of the ith destroy and repair operators in segment m, respectively. After each iteration, the updated weights for the next segment, $p^-_{i,m+1}$ and $p^+_{i,m+1}$, can be calculated using the following equations:

$$p^-_{i,m+1} = (1-r)p^-_{i,m} + r\frac{s^-_i}{a^-_i}, \tag{5.5}$$

$$p^+_{i,m+1} = (1-r)p^+_{i,m} + r\frac{s^+_i}{a^+_i}, \tag{5.6}$$

where r is the reaction factor (ranging from 0 to 1) that determines the influence of the scores from the current segment on the weight update. It controls the balance between past performance and the most recent score: a larger value of r places greater importance on the recent score, while a smaller value of r emphasizes past performance. s^-_i and s^+_i are the accumulated scores over the last segment of the ith destroy and repair operators, and a^-_i and a^+_i are the number of times the ith destroy and repair operators have been selected over the last segment, respectively. The updated weights are then used to calculate the probabilities for selecting destroy and repair operators in the next iteration in the next segment, as shown in Equations (5.3) and Equation (5.4).

The adaptive method for selecting destroy and repair operators in the ALNS heuristic enables it to explore various search trajectories and converge on high-quality solutions. By updating the operator weights, the algorithm adapts to the performance of the operators, focusing on those that have shown better results in recent iterations. This adaptive weight adjustment mechanism ensures that well-performing operators are selected more frequently, guiding the search towards promising regions of the solution space and focusing on operators more likely to yield better solutions.

5.5 Acceptance Criterion

In this section, we discuss the acceptance criterion used for evaluating and accepting solutions in the ALNS heuristic. Subsection 5.5.1 explains the general acceptance criterion based on the lexicographic objective functions of the SRRPFL. Subsection 5.5.2 explains the acceptance criterion based on simulated annealing, which is employed when a candidate solution is not accepted as the new global best solution.

5.5.1 General Acceptance Criterion

As introduced in Chapter 4, the lexicographic objective functions of the SRRPFL prioritize maximizing the number of passengers picked up by a driver and subsequently minimizing the total travel time for all drivers. In the ALNS heuristic, we accept a candidate solution x' as the new global best solution if the number of passengers picked up is higher. If the number of picked-up passengers is equal to the current global best's, we accept a candidate solution x' as the new global best solution only if its total travel time is lower than the current global best solution's total travel time.

Therefore, a candidate solution x' is considered better than another solution x if x' has picked up more passengers than x, or if they have an equal number of picked-up passengers and x' has a lower total travel time for drivers than x.

5.5.2 Simulated Annealing

An acceptance criterion based on simulated annealing (SA) is employed for the ALNS heuristic, similar to the acceptance criterion Ropke & Pisinger (2006) employed, when a candidate solution x' is not accepted as the new global best solution. The SA algorithm is outlined in Algorithm 4.

Algorithm 4 Simulated Annealing

Input: Candidate solution x', current solution x, cooling factor ξ, initial temperature T_{start}

Set temperature, $T \leftarrow T_{start}$
Compute the objective difference for both objectives, $\Delta Objective_1 \leftarrow f(x')_1 - f(x)_1$, $\Delta Objective_2 \leftarrow f(x')_2 - f(x)_2$
Compute the objective difference between both objectives, $\Delta \leftarrow \Delta Objective_1 - \Delta Objective_2$
if $\Delta Objective_1 > 0$ **then**
    Set acceptance probability, $P^{SA} \leftarrow 1$
else if $\Delta Objective_1 = 0$ and $\Delta Objective_2 < 0$ **then**
    Set acceptance probability, $P^{SA} \leftarrow 1$
else
    Calculate acceptance probability, $P^{SA} \leftarrow e^{\frac{\Delta}{T}}$
end if
Update temperature: $T \leftarrow T \cdot \xi$

Output: P^{SA} and T

The simulated annealing acceptance criterion involves evaluating a candidate solution x' obtained after applying the selected removal and repair operators. The objective values of the candidate solution, $f(x')$, and the current solution, $f(x)$, are compared. The acceptance criterion is similar to the one described in Subsection 5.5.1. We accept the candidate solution x' if there is an improvement in the first objective of maximizing the number of passengers picked up. If the number of passengers picked up remains unchanged, we accept the candidate solution x' only if there is an improvement in the second objective of minimizing the total travel time for drivers. Otherwise, the candidate solution x' may still be accepted as the current solution x with a probability calculated using the following formula:

$$P^{SA} = e^{\frac{-(f(x')-f(x))}{T}} \tag{5.7}$$

where $T > 0$ is referred to as the temperature. The temperature is initially set to a value T_{start} and is updated at each iteration by scaling the current temperature with a cooling rate, $\xi \in (0, 1)$, such that $T = T \times \xi$. This allows for more exploration in the early stages of the search when the temperature is higher and more exploitation towards the final stages as the temperature decreases. The selection of the initial temperature T_{start} and the cooling rate ξ is explained in Subsection 7.2.1.

5.6 Local Neighborhood Search

In this section, we introduce a local neighborhood search (LS) to enhance the performance of the ALNS heuristic in finding high-quality solutions more efficiently. The local neighborhood search employs local search operators (LSOs) that work in conjunction with the ALNS heuristic to refine the solutions obtained. The main idea behind the local neighborhood search is to explore the neighboring solutions of a candidate solution found by applying minor modifications through the use of LSOs.

Integrating a local neighborhood search within the ALNS framework is beneficial and appropriate for several reasons. Firstly, it allows for a more fine-grained exploration of the solution space by examining the vicinity of the current candidate solution. This complementary approach combines the strengths of both ALNS, which explores the broader solution landscape, and local search, which focuses on exploiting local improvements. The result is an enhanced ability to discover high-quality solutions that might have been overlooked by the ALNS alone. Secondly, incorporating a local neighborhood search can help accelerate convergence towards optimal or near-optimal solutions by continuously refining the candidate solutions generated by the ALNS heuristic. This can lead to a more efficient search process, as local search operators can quickly identify and exploit improvements in the solution, thus contributing to faster convergence and better solution quality. Lastly, the integration of a local neighborhood search within the ALNS framework provides a more versatile solution method. The combination of global exploration and local exploitation enables the heuristic to adapt more effectively to various problem instances and search scenarios, increasing the overall performance and adaptability of the proposed solution method.

Subsection 5.6.1 describes the integration of the LS with the ALNS heuristic, the process of applying LSOs to optimize candidate solutions, and the conditions for initiating the local search. Subsection 5.6.2 presents the local search operators (LSOs) used for the local search.

5.6.1 Local Neighborhood Search Strategy

In our approach, we integrate local neighborhood search with the ALNS heuristic to enhance the discovery of improved solutions. As outlined in Algorithm 1, the local neighborhood search is initiated if the candidate solution x' becomes the new global best solution, in order to potentially explore even better solutions in its vicinity. If the candidate solution x' is not as good as the current global best solution x^*, but is better than the current solution x, the local search is initiated under specific conditions. The candidate solution x' must pick up the same number of passengers as the current global best solution, and its total travel time should be within a predefined threshold, $\delta\%$, of the current best global solution's total travel time. If these conditions are met, the candidate solution x' is considered a *promising solution*. Consequently, the local search is conducted, which may result in finding a new global best solution, an improved solution to x', or no change at all. If x' is not considered a *promising solution*, the local search is not initiated, and the current solution x is set as the candidate solution x'.

When the local search is initiated, the LSOs are applied one after another in a predefined order, with the output of one LSO serving as the input to the next. Each LSO explores different route configurations according to the rules of first-improvement. If a change leads to an improvement of the input solution, that change is made. If the input solution is not

improved by a given LSO, it is passed unaltered to the subsequent LSO. If one or more LSOs find an improved solution, a new iteration of applying all the LSOs is initiated once the current iteration is finished. If no LSO succeeds in finding an improved solution, the output from the local search remains the same as the candidate solution provided by the ALNS iteration.

5.6.2 Local Neighborhood Search Operators

This section presents various local neighborhood search operators used to explore the solution space and improve the quality of the solution. Each operator considers assigning unassigned passengers to drivers whose routes have been modified, in order to explore potential improvements for each solution.

Intra-Passenger Swap

This operator modifies the order of passenger pick-ups and deliveries for all driver routes. It consists of two separate sequential procedures: Firstly, the passenger pick-up swap. Then, the passenger delivery swap.

Passenger pick-up swap modifies the order of passenger pick-ups within a driver's route. For each driver route, it iterates through all possible pairs of pick-up nodes and swaps their positions. The new route is then evaluated in terms of objectives, and if the new route shows improvement, it is accepted as the best route. Figure 5.3 shows an example of swapping node $(7, 0)$ with node $(6, 2)$. To further explore the solution space, the operator also considers all candidate pick-up locations for the swapped nodes, based on the information available in the $\mathcal{M}_i^P$ set.

Passenger delivery swap modifies the order of passenger deliveries within a driver's route. Similar to the pick-up swap, it iterates through all possible pairs of delivery nodes and swaps their positions. The operator also considers all candidate delivery locations for the swapped nodes, based on the information available in the $\mathcal{M}_i^D$ set. The new route is then evaluated in terms of objectives, and if the new route shows improvement, it is accepted as the best route.

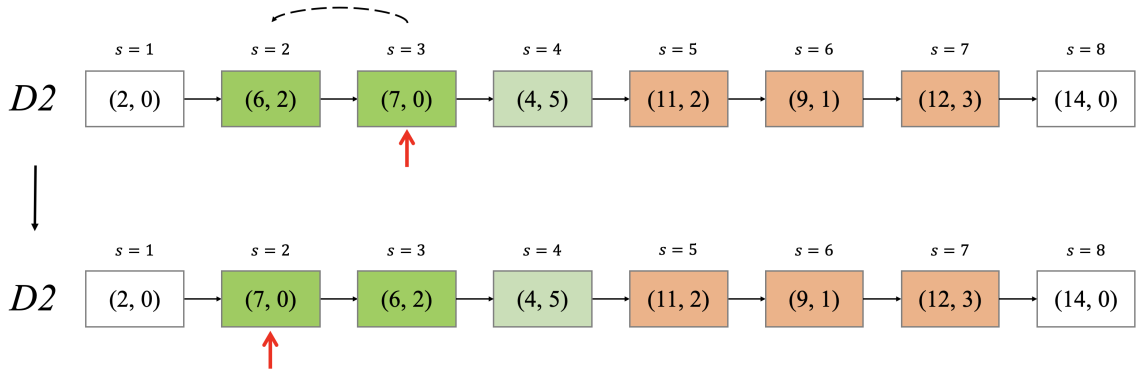


Figure 5.3: Example of Intra-Passenger Swap where Passenger 1 and Passenger 2 swaps the order of when they are picked up

Inter-Passenger Relocate

This operator performs a relocate of pick-up and delivery nodes between two different drivers' routes. It starts by filtering out the available drivers' routes based on their capa-

city. Then, for each driver, it iterates through their pick-up nodes and their corresponding delivery nodes. Next, it selects another driver and iterates through their pick-up nodes and corresponding delivery nodes as well. The pick-up and delivery nodes from the first driver's route are swapped with those from the second driver's route. During this process, for each candidate location, if no constraints are violated, it calculates the new objective values for both objectives. If the new solution is accepted as the the new global best solution, the new routes are accepted as the best routes, and the process continues iterating. Otherwise, the operator moves on to the next passenger. The inter-passenger swap operator continues iterating until no further improvements can be found. Figure 5.4 illustrates an example of this LSO.

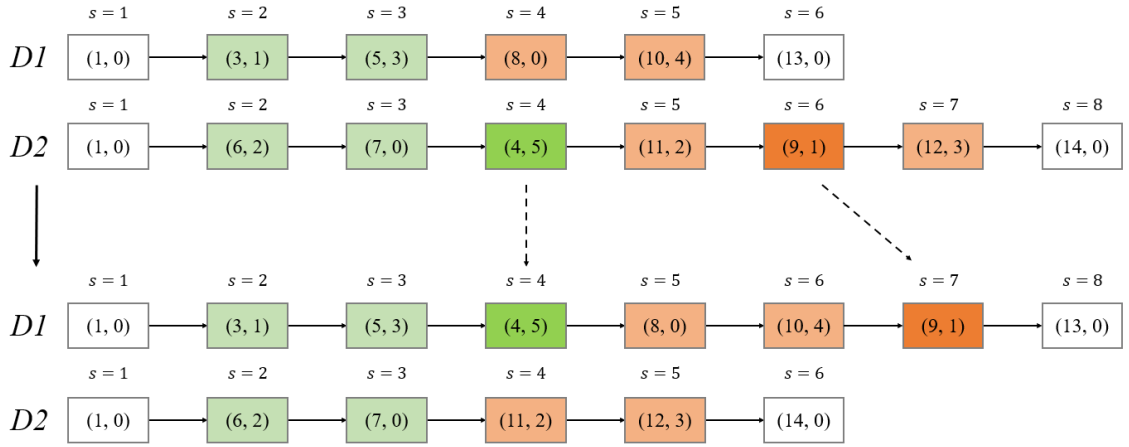


Figure 5.4: Example of Inter-Passenger Relocate where Passenger 3 in Driver 2's route is transferred to Driver 1's route

Candidate Location Shift

This operator shifts the candidate location for a passenger's pick-up and delivery location within a driver's route. For each driver route, the function iterates through all pick-up nodes and modifies the candidate location of the chosen pick-up node with all other candidate locations for the same passenger from the set $\mathcal{M}_i^P$. The function then constructs a new driver route by inserting the new candidate location and updating the original driver route. Once all passenger pick-up nodes have been considered in a route, the operator does the same with the delivery nodes, using the $\mathcal{M}_i^D$ set. Figure 5.5 illustrates this by shifting the candidate location of node $(7, 0)$ to $(7, 2)$.

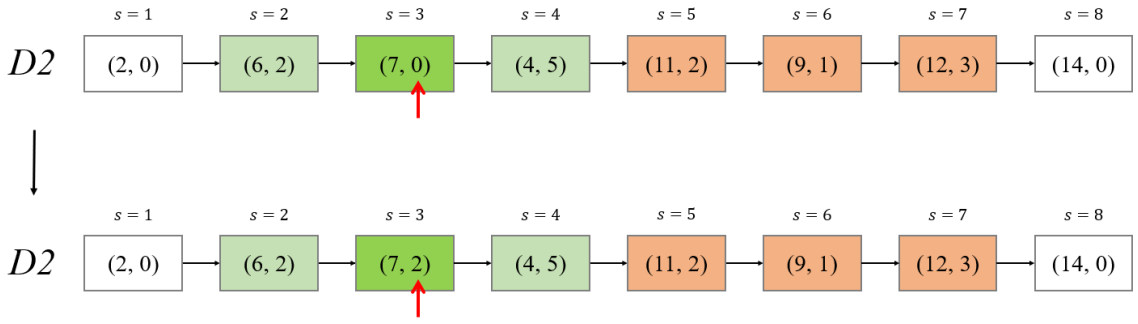


Figure 5.5: Illustration of the candidate location shift operator, showcasing the replacement of a passenger's pick-up location within a driver's route with a new candidate pick-up location from the set $\mathcal{M}_i^P$

5.7 The Route Combination Problem

As explained in Section 5.2, each driver has a route in the complete solution generated by the ALNS heuristic. However, while some routes may perform well for certain drivers, the overall solution may be suboptimal. To address this issue and enhance the solutions provided by the ALNS, we introduce the route combination problem (RCP). This is a set-partitioning problem, with the same objectives as the SRRPFL.

Transitioning from the arc-flow model to the route combination problem (RCP) formulation, we now utilize a path-flow model. This shift represents the flow of drivers and passengers along entire paths (routes) instead of individual arcs, providing an alternative perspective for solving the SRRPFL and enhancing the overall solution quality.

The set-partitioning formulation of the problem is incorporated within the ALNS to enhance its effectiveness in finding better solutions. By transforming the problem into a set-partitioning representation, the ALNS is able to exploit the inherent structure and characteristics of the SRRPFL, thereby enabling more efficient exploration and identification of high-quality solutions. Additionally, the set-partitioning formulation provides a systematic and compact representation of feasible solutions, which contributes to improving the overall performance and solution quality of the ALNS.

We solve the RCP every I^{RCP} iterations of the ALNS, and accept the solution based on the same criteria as finding a new global best solution. To formulate the RCP, we use the notation presented in Section 4.1 along with additional notation relevant to the RCP, which we introduce below.

Sets

$\mathcal{R}_k$ - set of all routes previously identified as available for driver $k \in \mathcal{D}$ during the ALNS search

Parameters

N_{kr} - number of passengers picked up by driver $k \in \mathcal{D}$ on route $r \in \mathcal{R}_k$

T_{kr} - travel time for driver $k \in \mathcal{D}$ on route $r \in \mathcal{R}_k$

A_{ikr} - 1 if passenger $i \in \mathcal{P}^P$ is picked up by driver $k \in \mathcal{D}$ on route $r \in \mathcal{R}_k$, 0 otherwise

Decision Variable

x_{kr} - 1 if driver k travels route r, 0 otherwise

Objective Functions

Similar to the objective functions described in Subsection 4.2.1, the RCP is also formulated as a multi-objective optimization problem with two objective functions. These two

objective functions are solved in a lexicographic method, similar to the objective functions presented in Subsection 4.2.1. Objective (5.8) aims to maximize the number of passengers that is picked up, while Objective (5.9) minimizes the total travel time for drivers.

$$\max Z_1 = \sum_{k \in \mathcal{D}} \sum_{r \in \mathcal{R}_k} N_{kr} x_{kr} \tag{5.8}$$

$$\min Z_2 = \sum_{k \in \mathcal{D}} \sum_{r \in \mathcal{R}_k} T_{kr} x_{kr} \tag{5.9}$$

Constraints

Constraints (5.10) ensure that each passenger is a part of maximum one route that is chosen by the RCP. Further, Constraints (5.11) ensure that the RCP selects exactly one route for every driver. Lastly, Constraints (5.12) put a binary requirement on the use of routes.

$$\sum_{k \in \mathcal{D}} \sum_{r \in \mathcal{R}_k} A_{ikr} x_{kr} \leq 1, \quad i \in \mathcal{P}^P \tag{5.10}$$

$$\sum_{r \in \mathcal{R}_k} x_{kr} = 1, \quad k \in \mathcal{D} \tag{5.11}$$

$$x_{kr} \in \{0, 1\}, \quad k \in \mathcal{D}, r \in \mathcal{R}_k \tag{5.12}$$

Chapter 6

Case Study and Test Instances

This chapter provides a description of the case study used in this thesis, along with the test instances generated to evaluate the proposed algorithms. Section 6.1 presents the Sotra case with an overview of the region, its geographical location, and a brief introduction to the data used in this thesis. Section 6.2 explains the methods employed to determine the candidate locations (sets $\mathcal{M}_i^P$ and $\mathcal{M}_i^D$). Section 6.3 presents an overview of the test instance generation process. Section 6.4 presents the test instances generated.

6.1 The Sotra Case

This section presents the Sotra case. Subsection 6.1.1 presents an overview of the Sotra region and its geographical location. Subsection 6.1.2 presents an introduction to the data used in this thesis, and detail of the various zones in the Sotra region and their corresponding origin and destination locations.

6.1.1 Overview

Sotra is an island archipelago located off the coast of Bergen, Norway's second-largest city. One of the distinguishing features of Sotra is the single bridge, known as Sotrabroen, that connects the island to the greater Bergen area. This connection is vital for the daily lives of Sotra's 40,000 inhabitants (Øygarden kommune, 2023), as many of them commute to Bergen for work, education, and other essential services. Due to the limited public transportation options available on Sotra, a substantial number of residents rely on private vehicles for their daily commute. This heavy dependence on cars has led to a considerable increase in traffic, particularly on Sotrabroen, which connects Sotra to the greater Bergen area. The resulting congestion has become a major concern for both local authorities and residents, as it affects the efficiency of the transportation system, the environment, and the overall quality of life. A map of Sotra and the greater Bergen area is displayed in Figure 6.1. Sotra is located on the west coast, while Bergen is situated on the mainland. The green marker indicates the location of Sotrabroen.

Figure 6.1: Map of Sotra and Bergen, highlighting the location of Sotrabroen (green marker), the bridge connecting the island to the mainland

6.1.2 Input Data

The input data received for this thesis is accredited to Telia, a Nordic telecom company, which provided trip data sets for the Sotra region (Telia, 2022). These data sets contain information on the travel patterns of residents in the area. The trip data sets encompass origin-destination matrices at various levels of detail and different time periods to cater to a broad range of use cases. Trips are defined using a dynamic break parameter that represents the minimum duration of stationary dwelling, which is an activity that breaks an ongoing trip and initiates a new trip when the individual starts moving again. The break parameter, in essence, determines whether a trip is split into two parts based on the distance traveled, the duration of the stop, and the directional change. The trip data is aggregated into hourly intervals, providing hourly trip counts for each calendar day within the data collection period, which spans from 3rd October 2022 to 30th October 2022. The data set contains information on various aspects of the trips, such as the starting and ending locations, postal codes, Grid IDs, and the total number of trips. In addition, the data is classified into weekdays and weekends, as well as trips originating from the island and trips heading to the island.

The data provided by Telia enables us to pinpoint the origin and destination locations for each trip when people travel from Sotra to the greater Bergen area. Figure 6.2 illustrates the various zones of origin and destination locations. The red markers represent origin locations, while the blue markers indicate destination locations. The origin locations are divided into three distinct zones, whereas all the destination locations are contained within a single zone. In total, there are 43 origin locations and 20 destination locations. This represents an extension of the work presented in Nitter & Yang (2022), where the authors considered 24 origin locations and only four destination locations. Zone 1 consists of thirteen locations, Zone 2 consists of 20 locations, and Zone 3 consists of nine locations. Zone 4, the destination zone, consists of 20 locations. A more detailed description of the zones with location names and labels is presented in Appendix B, and the exact coordinates for all origin and destination locations can be found in Appendix C.

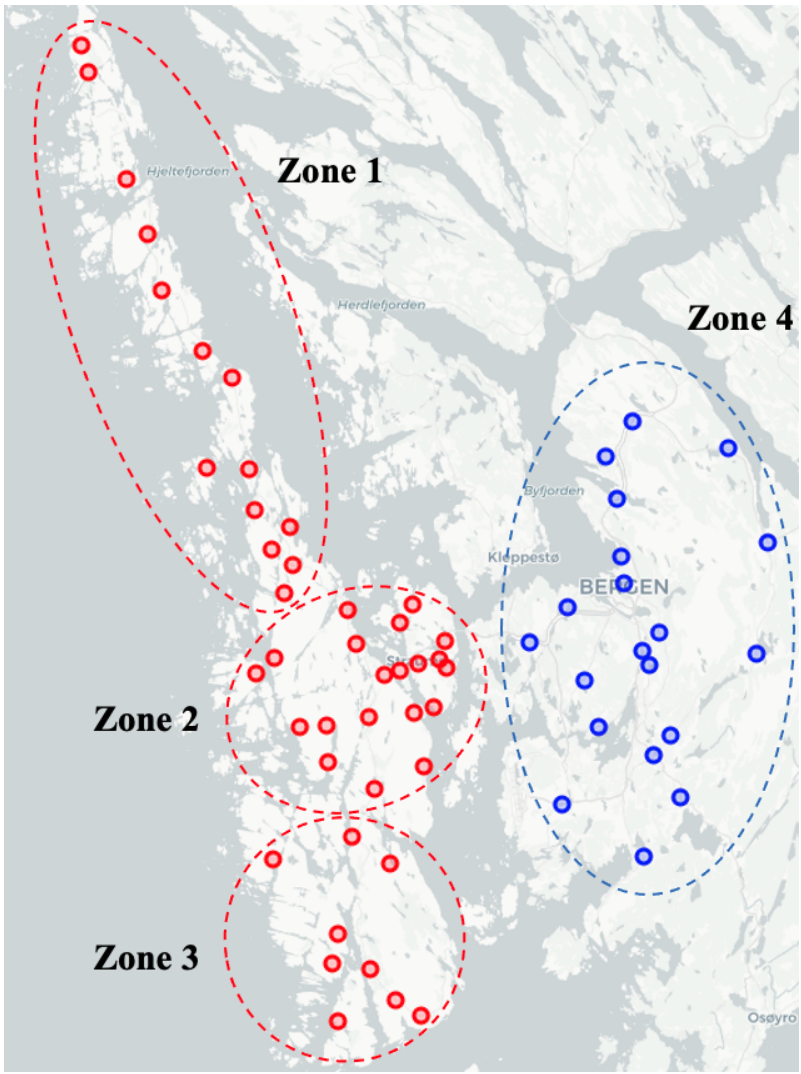


Figure 6.2: Map of the Sotra region, showing the origin (red markers) and destination (blue markers) locations in their respective zones. The origin locations are divided into three zones, while all destination locations are within a single zone. This figure builds upon the work of Nitter & Yang (2022) by considering 43 origin locations and 20 destination locations.

6.2 Candidate Locations

A candidate location is defined as a location where a passenger can travel to and be picked up by a driver, or a location where a driver can deliver a passenger before the passenger travels to its destination, as described in Chapter 3. Candidate locations are divided into candidate pick-up ($\mathcal{M}_i^P$) and candidate delivery locations ($\mathcal{M}_i^D$), as described in Chapter 4.

Various methods can determine the sets $\mathcal{M}_i^P$ and $\mathcal{M}_i^D$, depending on the input data. In this thesis, the set $\mathcal{M}_i^P$ includes the origin location of passenger i as well as other nearby possible origin locations. These additional origin locations are determined by a parameter ρ, which represents the maximum distance from the origin location of passenger i to other origin locations. If other origin locations are within the distance defined by ρ, we select the θ closest origin locations and include them in the $\mathcal{M}_i^P$ set for passenger i. In this chapter, θ will be set to 3, as it represents a feasible number of locations that a passenger might reasonably travel to for pick-up.

Apart from the distance-based approach, other methods to determine the sets $\mathcal{M}_i^P$ and $\mathcal{M}_i^D$ could include using density-based clustering algorithms, such as DBSCAN, which identifies regions with a high density of points as clusters. These clusters could represent areas with high demand for pick-up or delivery locations. Another approach might involve incorporating historical data of frequently used pick-up and delivery locations, which could help identify areas with high demand for shared rides. Additionally, leveraging real-time data, such as traffic conditions and public transit schedules, could enable dynamic determination of candidate locations that optimize travel time and efficiency for both passengers and drivers. Finally, considering passengers' preferences and individual constraints, such as accessibility requirements or preferred walking distances, could also contribute to the determination of the sets $\mathcal{M}_i^P$ and $\mathcal{M}_i^D$, allowing for a more personalized and inclusive shared ride experience.

For the set $\mathcal{M}_i^D$, the determination can also be based on a similar ρ parameter describing how far away other destination locations are. However, in this thesis, we intentionally choose not to include other destination locations in the $\mathcal{M}_i^D$ set. Consequently, the only delivery locations for passengers is their respective destination locations, with no other candidate delivery locations considered. This decision is based on the rationale that people may be willing to travel to another location for pick-up, thus justifying the inclusion of multiple locations in the $\mathcal{M}_i^P$ set. On the other hand, when being delivered from their homes to their workplaces, people generally prefer to arrive at their exact destination location rather than a different one.

People might prefer arriving at their exact destination for several reasons, including saving time and effort, familiarity with the area, time sensitivity, and reducing inconvenience. However, certain scenarios may benefit from multiple candidate delivery locations, such as urban settings with high population density, areas with well-developed public transportation networks, and when addressing environmental concerns. In this thesis, we focus on cases where the set of delivery locations is limited to passengers' exact destinations. Expanding this set could be advantageous in various real-world situations, such as faster shared rides in urban settings, better integration with public transit networks, and supporting sustainable urban transportation. Additionally, carpooling scenarios could benefit from multiple delivery points, enhancing efficiency and attractiveness of shared transportation modes. For instance, in large workplace campuses, having multiple delivery locations allows for more convenient and efficient carpooling systems.

6.3 Test Instance Generation

This section provides an overview of the test instance generation process, which aims to create representative instances of travel demands and preferences for Sotra's residents during specific hours of the day. The procedure for test instance generation is outlined in Subsection 6.3.1, and the assumptions made is discussed in Subsection 6.3.2.

6.3.1 Procedure

The test instance generation is a three-step process that starts with the raw data, as described in Section 6.1. First, in order to preprocess this data, we counted the number of trips taken from each origin to each destination, specifically focusing on cases where people traveled from Sotra to the greater Bergen area. It is important to note that the data also includes movement of people between locations within Sotra and locations within Bergen. Second, after counting the trips from origin to destination, we obtained a matrix representing the total number of travel occurrences during specific hours for each route in a month. In other words, matrices of total travel occurrences were created for the following time intervals during a day: 6-7, 7-8, 8-9, and 9-10 am. These matrices are referred to as "occurrences" matrices. These occurrence matrices are subsequently aggregated to create a single occurrence matrix. Third, from the aggregated occurrences matrix, we generate test instances by calculating the probability of selecting each route from an origin to a destination. This is accomplished by dividing the number of occurrences of a specific route by the total number of routes. We can then compute a distribution, called F, based on these routes.

During the creation of the occurrence matrix, there were routes with very negligible numbers of trips from certain origins to destinations in Bergen. In certain instances, there were no recorded travels from a specific origin to a particular destination within a specific hour throughout a given month. To refine our data, these origins were filtered out. Additionally, the data contained origins and destination where some people traveled from Sotra to locations other than locations in the greater Bergen area, such as Oslo. These occurrences were also filtered out.

Algorithm 5 is used to create test instances. This algorithm takes as input the filtered travel data for Sotra's residents from 06:00 to 10:00 between October 3rd, 2022, and October 30th, 2022, obtained from Telia, along with the number of drivers $|\mathcal{D}|$ and passengers $|\mathcal{P}^P|$ to be included in an instance. The algorithm starts by computing the number of occurrences between each origin and destination, N. Based on these occurrences, the probability of choosing an origin and destination to create a particular route, P^{route}, is computed by dividing N by the total number of trips recorded in the input data. Then, a distribution F is created. This distribution is obtained by using the calculated probabilities P^{route} to determine the likelihood of each route being chosen. Further, driver origin and destination locations are generated. The algorithm iterates over the number of drivers, choosing an origin and destination for each driver based on the distribution F. A similar purpose is served for passengers. The algorithm iterates over the number of passengers, selecting an origin and destination for each passenger according to the distribution F. Finally, the instance I is returned as output. Each driver and passenger in the instance I has a maximum travel time and a time window. The time window is randomly chosen from four possible windows within the hour the instances are generated. These are [0X:00, 0X:30], [0X:15, 0X:45], [0X:30, 0X+1:00], and [0X:45, 0X+1:15], where

X represents a given hour between 6 and 9. The maximum travel time determination is further explained in Subsection 6.3.2.

Algorithm 5 Test Instance Generation

Input: Filtered travel data for Sotra's residents from 06:00 to 10:00 between October 3rd 2022, and October 30th 2022, obtained from Telia. Number of drivers, $|\mathcal{D}|$. Number of passengers, $|\mathcal{P}^P|$

1: Compute the number of occurrences between origins and destinations, N
2: Compute the probability of assigning origins and destinations to create a route, P^{route}
3: Compute the distribution of assigning origins and destinations, F
4: **for** k in range $|\mathcal{D}|$ **do**
5: Choose an origin and destination for a driver based on distribution F
6: **end for**
7: **for** i in range $|\mathcal{P}^P|$ **do**
8: Choose an origin and destination for a passenger based on distribution F
9: **end for**

Output: Instance I

6.3.2 Ridesharing and Instance Generation Assumptions

The input data revealed that few people reside on the fringes of Sotra, specifically to the north of Zone 1 and south of Zone 3 (Figure 6.2). As a result, only a handful of drivers would likely be assigned to these areas based on the distribution. However, positioning drivers in these sparsely populated areas could potentially lead to more people participating in ridesharing in Sotra, aligning with our objective to promote such a practice. This is because drivers starting from the edge of Sotra would likely pass numerous other origin locations before reaching Sotrabroen. In reality, according to the occurrence matrix, a substantial number of people are traveling from Zone 2 (Figure 6.2).

This data pattern suggests a higher likelihood of assigning drivers and passengers with origins in Zone 2 when employing Algorithm 5 for instance generation. In this context, the generated drivers might not be willing to travel to Zones 1 and 3 to pick up passengers. This problem becomes particularly pronounced when generating smaller instances, where the origins of the generated drivers predominantly lie in Zone 2, leading to test instances that might not accurately reflect potential ridesharing scenarios. However, this issue is somewhat mitigated in larger instances due to the increased availability of drivers, thus enhancing the chances of drivers originating from or passing through Zones 1 and 3. Consequently, the coverage of all zones is more likely to be comprehensive in larger instances compared to smaller ones, yielding a distribution of drivers and passengers that is more realistic and representative of potential ridesharing scenarios in Sotra.

To address the issue highlighted for smaller and some medium-sized instances, we adopted a specific approach for instance generation. After creating the instances using Algorithm 5, we run the instances in the ALNS heuristic to assess the early ride-sharing effectiveness. Specifically, we focus on whether over 75% of passengers can be picked up during the initial iterations of the ALNS, starting from the result provided by the construction heuristic. This check will be referred to as the *75% ridesharing check*. If an instance achieves this threshold, we accept it as a valid test instance; if not, a new test instance using Algorithm 5 is generated. This validation process is applied exclusively for the small instance groups and the smallest medium instance group, as later presented in Subsection 6.4.2. For the

two largest medium instances and all large instances, we abstain from performing the *75% ridesharing check*. This decision is based on the reasoning outlined in the previous paragraph, which highlights how the increased availability of drivers in larger instances reduces the potential for spatial bias, rendering the distribution of drivers and passengers more representative of potential ridesharing scenarios in Sotra.

The maximum time a passenger is willing to travel to reach a candidate pick-up location is set to 10 minutes ($\rho = 10$ min). This value was chosen based on the assumption that passengers would generally be willing to travel a short distance to their pick-up locations to ensure a more efficient and cost-effective shared transportation service. Furthermore, inspired by the work of Nitter & Yang (2022), we define the maximum travel time parameter T_k^M for each driver and passenger as a multiplier of the direct travel time from their origin to their destination, $(T^D_{o(k),d(k)}, k \in \mathcal{D})$ and $(T^D_{imjn}, (i,m) \in \mathcal{N}^P, (j,n) \in \mathcal{N}^D)$, respectively. This multiplier is set at 1.7x, implying that both drivers and passengers are willing to tolerate longer car journeys in exchange for the benefits of ridesharing. Furthermore, the maximum capacity for drivers, Q_k, is set to four for all drivers. This implies that each driver can pick up a maximum of four passengers.

6.4 Test Instances

This section presents the test instances used for evaluating the performance of the ALNS algorithm and its extensions, which will be discussed in detail in Chapter 7. Subsection 6.4.1 describes the parameter tuning instances, which are specifically created for the purpose of tuning the parameters in the ALNS heuristic. Subsection 6.4.2 describes the performance instances, designed to assess the overall performance of the ALNS algorithm and its extensions under various conditions.

6.4.1 Parameter Tuning Instances

Parameter tuning instances are created for the purpose of tuning the parameters in the ALNS. These instances are solely used for tuning purposes and are generated using Algorithm 5, without doing the *75% ridesharing check* presented in Subsection 6.3.2. Chapter 7 presents which parameters are tuned using the parameter tuning instances. These instances are displayed in Table 6.1. The Instance ID indicates the instance size, number of drivers, and number of passengers. For example, M1-5D-18P refers to medium instance 1, with 5 drivers and 18 passengers.

6.4.2 Performance Instances

The performance instances are generated using Algorithm 5. All instances are divided into instance groups. There are nine instance groups in total, with each instance group consisting of five instances, resulting in 45 instances overall. The instances within each instance group share the same number of drivers and passengers. Dividing the instances into instance groups and generating unique instances within each group is beneficial when testing the ALNS algorithm for several reasons. First, this approach provides a diverse range of scenarios, allowing the algorithm to be tested on varying conditions. Second, by having different generated instances within each group, we can assess the algorithm's adaptability and flexibility in handling changes in origin and destination locations. This

Table 6.1: Parameter tuning instances with Instance ID format: InstanceNumber-Drivers-Passengers (e.g., 1-5D-18P represents instance 1 with 5 drivers and 18 passengers)

Instance ID	Drivers	Passengers
1-5D-18P	5	18
2-6D-20P	6	20
3-8D-25P	8	25
4-9D-28P	9	28
5-10D-30P	10	30
6-12D-36P	12	36
7-13D-41P	13	41
8-14D-46P	14	46

is crucial, as real-world scenarios often involve unpredictable changes. Lastly, this division and variation in instances contribute to a more comprehensive evaluation of the ALNS algorithm's performance, increasing the reliability of the results and their potential applicability in practical settings.

The instance groups are summarized in Table 6.2. Here, the letters S, M and L stands for small, medium and large. Instance IDs column displays how the different instances in each instance group are identified, where X indicates unique instances. As mentioned in Subsection 6.3.2, the *75% ridesharing check* has been employed for instance groups S1, S2, S3, and M1. All instances are summarized in Appendix D. These instance groups are used in Chapter 7 to test the performance of ALNS heuristic and its extensions.

Table 6.2: Summary of instance groups and corresponding Instance IDs

Instance Group	Drivers	Passengers	Instance ID
S1	1	4	S1-1D-4P-X
S2	2	6	S2-2D-6P-X
S3	4	10	S3-4D-10P-X
M1	8	20	M1-8D-20P-X
M2	12	30	M2-12D-30P-X
M3	16	42	M3-16D-42P-X
L1	20	60	L1-20D-60P-X
L2	25	75	L2-25D-75P-X
L3	35	100	L3-35D-100P-X

Chapter 7

Computational Study

This chapter presents the computational study of the adaptive large neighborhood search's (ALNS) performance when solving the Static Ridesharing Routing Problem with Flexible Locations (SRRPFL).

Section 7.1 outlines the test environment used for the computational study. Section 7.2 details the process of parameter tuning and presents comparisons between large neighborhood search (LNS), ALNS, and its respective extensions. Finally, Section 7.3 compares the performance of the heuristic with that of the commercial solver.

7.1 Test Environment and Stopping Criterion

In this section, we describe the test environments used for the computational study. Two different test environments were employed for parameter tuning and performance testing of the ALNS heuristic.

The parameter tuning of the ALNS heuristic, later detailed in Section 7.2.1, was conducted on a Dell OptiPlex 7780 AIO, equipped with an Intel Core i7-10700 processor running at 2.90 GHz and 16 GB of RAM. The operating system installed on the machine is Windows 10. The ALNS, instance generation, and local search extension were run in Python v3.8.8. The arc-flow model and the model of the route combination problem (RCP) were run using Gurobi v10.0. Table 7.1 provides a summary of the hardware and software components utilized in the parameter tuning process

Table 7.1: Hardware and software used for parameter tuning

Computer type	Dell OptiPlex 7780 AIO
Processor	Intel Core i7-10700, 2.90 GHz (8 cores)
RAM	16 GB
Operating system	Windows 10
Commercial solver	Gurobi v10.0
Programming language	Python v3.8.8

For the performance testing of the ALNS heuristic, the tests were run on a computing node

in the *Solstorm* computing cluster at the Norwegian University of Science and Technology. The computing node is a HP bl685c G7 computer, running on Linux CentOS version 7. It has four 2.2GHz AMD Opteron 6274 processors with 16 cores each and 128GB of RAM. The ALNS, instance generation, and local search extension were run in Python v3.9.6. The arc-flow model and the model of the route combination problem (RCP) were run using Gurobi v9.5. Table 7.2 provides a summary of the hardware and software components utilized for performance testing of the ALNS heuristic.

Table 7.2: Hardware and software used for performance testing

Computing node	HP bl685c G7
Processors	Four 2.2GHz AMD Opteron 6274 (16 cores each)
RAM	128 GB
Operating system	Linux CentOS, version 7
Commercial solver	Gurobi v9.5
Programming language	Python v3.9.6

In our case study, the SRRPFL is supposed to be solved daily, generating routes in a static setting. As a result, the ALNS heuristic has two stopping criteria: (1) a predetermined number of I^{ALNS} iterations have been performed, or (2) the runtime reaches 3600 seconds. The second stopping criterion where runtime reaches 3600 seconds is considered reasonable for our case study. Since the ridesharing routes need to be generated and communicated to the drivers and passengers before their daily commute - by setting the maximum runtime to 3600 seconds, we ensure that the ALNS heuristic provides a timely solution that can be effectively implemented in the real-world context. Similarly, the maximum runtime for the commercial solver is set to 3600 seconds.

Owing to the inherent randomness of the ALNS heuristic, each test instance is run multiple times to obtain a reliable estimate of the algorithm's average performance. When presenting the results from running the ALNS heuristic on a test instance, we report the average results from five independent runs. In the process of tuning parameters, five runs are conducted per test instance and parameter setting, which will be discussed in detail in Section 7.2.1. The commercial solver's results are deterministic; therefore, a single run per instance is sufficient.

7.2 Configurations of the ALNS heuristic

This section describes the parameter tuning process and performance testing for various configurations of the ALNS heuristic. Subsection 7.2.1 describes the tuning process for the parameters and provides details on the un-tuned parameters. Subsection 7.2.2 discusses the impact of using adaptive weights in the ALNS heuristic. Subsection 7.2.3 investigates the extensions of the ALNS and evaluates their influence on the heuristic's performance.

7.2.1 Parameter Tuning

Parameter tuning is an important aspect of designing and implementing an ALNS. The performance of an ALNS-based algorithm is affected by the choice of parameters, which control the balance between exploration and exploitation within the search process. Selecting suitable parameter values ensures that the algorithm can efficiently navigate the solution space and converge to high-quality solutions.

In this thesis, we refer to *un-tuned parameters* as those that have not been systematically adjusted throughout the development of the ALNS heuristic. These parameters were mainly determined by a trial-and-error process during the creation of the ALNS heuristic. These parameters were given less focus because they were assumed to have less impact on the solutions compared to other factors. An illustration of the un-tuned parameters, along with their associated values and descriptions, is presented in Table 7.3.

Table 7.3: Summary of un-tuned parameters with their values and descriptions used in the ALNS heuristic

Parameter	Value	Description
I^{ALNS}	5000	Number of ALNS iterations
I^{S}	100	Number of segments between each operator weight update
I^{RCP}	300	Number of iterations between each time the route combination problem (RCP) *(Section 5.7)* is solved
T_{start}	-	Simulated annealing temperature, set so that the probability of accepting a candidate solution is 50% if the candidate solution is less than 5% worse than the current solution
ξ	0.97	The cooling rate for the simulated annealing acceptance criterion *(Section 5.5)*
k	3	Regret-k parameter *(Section 5.4.1)*

We chose 5000 iterations for the ALNS (I^{ALNS}) to balance the trade-off between solution quality and computational time. The number of segments between each operator weight update (I^{S}) was set to 100. This value was chosen to allow the algorithm to respond to changes in the search landscape while avoiding excessive oscillations in the operator weights, which could negatively affect the search process. We scheduled the RCP to be solved every 300 iterations (I^{RCP}) (Section 5.7). This frequency ensured that the problem was solved often enough to take advantage of new information from the ALNS heuristic. The initial simulated annealing temperature (T_{start}) was set such that the probability of accepting a candidate solution is 50% if the candidate solution is less than 5% worse than

the current solution, inspired by Liu et al. (2019). The cooling rate for the simulated annealing acceptance criterion (ξ) was set to 0.97 (Section 5.4.1). This value was chosen to gradually decrease the temperature and, consequently, the likelihood of accepting worse solutions as the search progresses. For the regret-k destroy operator (Section 5.4.1), we set the parameter k to 3.

For tuning purposes, eight tuning instances are generated, as described in Section 6.4.1. The systematically tuned parameters, listed in Table 7.4, play crucial roles in the ALNS heuristic. The parameter γ represents the percentage range of passenger removals for destroy operators. The parameters σ_1, σ_2, and σ_3 are rewards given to operators for adaptive weight adjustment, while σ_4 is the penalty applied to operators. The parameter δ is a percentage factor that determines whether a candidate solution's second objective value is within δ% of the global best's second objective value (a *promising solution*). Lastly, r is the reaction factor used in adaptive weight adjustment.

In the tuning process, we systematically adjust each parameter in the order presented in Table 7.4. The configuration of the ALNS during parameter tuning is ALNS with local search and solves the path-flow model RCP (ALNS + LS + RCP). We begin by tuning γ. For each value of γ, we conduct tests and identify the best result that provides the best performance. Once we have determined the best value for γ, we fix it and move on to the next parameter in the table. We follow the same procedure for each subsequent parameter, ensuring that all previously tuned parameters remain fixed while tuning the current one. This sequential approach allows us to fine-tune the ALNS heuristic systematically and efficiently, ultimately resulting in a well-tuned algorithm. This tuning process is performed using the un-tuned parameters displayed in Table 7.3. For each tuning instance, we run five tests for each parameter setting and calculate the average objective values and computational times. We evaluate five different settings for each parameter to find the best setting. The best parameter setting is determined by comparing the average objective values and computational times.

Table 7.4 presents the tuned parameters along with their initial and final values, as well as their descriptions. Upon inspecting Table 7.4, it is evident that many parameters have retained their initial values. The only parameter that has undergone a change is the percentage factor δ, which has increased from 85% to 90%. As a result, we observe a slight improvement, highlighting the benefit of tuning parameters to enhance the performance of the ALNS heuristic. The details of each parameter's tuning process are provided in Appendix E.

Table 7.4: Summary of tuned parameters, their initial and final values, and descriptions. Param. = Parameter

Param.	Initial Value	Final Value	Description
γ	$[5\%, 15\%]$	$[5\%, 15\%]$	Percentage range of number of removals of passengers for destroy operators
σ_1	100	100	ALNS reward for finding a new global best solution *(Subsection 5.4.3)*
σ_2	30	30	ALNS reward for finding a candidate solution that is better than the current solution, but not better than the global best solution *(Subsection 5.4.3)*
σ_3	10	10	ALNS reward for finding a candidate solution that is worse than the current solution, but accepted by the simulated annealing criterion *(Subsection 5.4.3)*
σ_4	-5	-5	ALNS penalty for finding a candidate solution that is worse than the current solution, and rejected by the simulated annealing criterion *(Subsection 5.4.3)*
δ	85%	90%	Percentage factor that determines if a candidate solution is a promising solution, where its second objective value is within $\delta\%$ of the global best's second objective value *(Subsection 5.6.1)*
r	0.10	0.10	ALNS reaction factor for operator weights update *(Subsection 5.4.3)*

7.2.2 Comparing ALNS and LNS

In this section, we compare the performance of the adaptive large neighborhood search (ALNS) heuristic to the large neighborhood search (LNS) heuristic. Both ALNS and LNS are metaheuristics that iteratively explore the solution space by applying a combination of destroy and repair heuristics. However, the main difference between them lies in the way they select the destroy and repair operators.

In ALNS, the selection of operators is guided by the use of adaptive weights, as described in Subsection 5.4.3. The adaptive weights allow the algorithm to dynamically adjust the probability of choosing a specific destroy and repair operator based on its past performance. On the other hand, LNS does not employ adaptive weights and selects destroy and repair operators randomly in each iteration. By comparing the performance of ALNS and LNS, we aim to investigate the effects of using adaptive weights in the search process and assess the contribution of this feature in finding high-quality solutions. In the proceeding sections, we refer the primary objective of maximizing number of passengers picked up as Objective 1, and the secondary objective of minimizing total travel time for drivers as Objective 2.

To evaluate the performance of the LNS, and ALNS heuristic and its extensions, we introduce a "Gap" metric. This serves as an indicator of the solution quality relative to the best-known solution for both objectives. For performance testing, we run the LNS, and ALNS heuristic and its extensions five times for each instance. To compute Gap^{1gx} for Objective 1 and Gap^{2gx} for Objective 2 for each instance x in instance group g, we calculate the average performance and compare this to the best performance across the heuristic configurations that are compared. Gap^{1gx} for each instance x in instance group g is defined as the difference between the best performance and the average performance, expressed as a percentage, because Objective 1 is maximized. On the other hand, Gap^{2gx} for each instance in instance group g is defined as the difference between the average performance and the best performance, expressed as a percentage, because Objective 2 is minimized. The gaps for each instance group, Gap^{Obj1} and Gap^{Obj2}, are calculated as the average gaps for all instances in the instance group. These metrics provide a meaningful measure for assessing the effectiveness of the ALNS heuristic and its extensions across different instances, allowing for a comprehensive comparison and analysis. This definition of gap is used throughout the rest of this chapter.

In addition to the "Gap" metric, we also make use of the average coefficient of variation (CV) for both objectives, termed as CV^{Obj1} and CV^{Obj2}. The CV is a useful statistic that describes the level of variability in relation to the mean of the population. To compute CV^{1gx} for Objective 1 and CV^{2gx} for Objective 2 for each instance x in instance group g, we calculate the ratio of the standard deviation to the mean of the objective value. The coefficient of variation for each instance group, CV^{Obj1} and CV^{Obj2}, is calculated as the average CV for all instances in the instance group. We compute CV^{Obj1} and CV^{Obj2} for each objective to capture the relative variability in our data. For instance, a low coefficient of variation indicates that the data points are very close to the mean, and hence to each other, suggesting a more consistent and reliable heuristic. On the other hand, a high coefficient of variation indicates that the data points are spread out over a large range of values. In the context of the LNS, and ALNS heuristic and its extensions, the CV is important for measuring the consistency of the performance. It helps evaluate not just the quality of the solutions (which is indicated by the "Gap" metric), but also the reliability of the heuristic. This definition of CV is used throughout the rest of this chapter.

Table 7.5: Comparison of results for the ALNS and LNS heuristics. $\overline{\textbf{Obj. 1}}$ and $\overline{\textbf{Obj. 2}}$ represent the average objective values for Objectives 1 and 2 for each instance group, respectively. CV^{Obj1} and CV^{Obj2} represent the average coefficient of variation for Objectives 1 and 2 for each instance group, respectively. Gap^{Obj1} and Gap^{Obj2} represent the average gap for Objectives 1 and 2 for each instance group across the ALNS and LNS heuristics, respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds

Instance	**LNS**							**ALNS**						
Group	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	CV^{Obj1}	CV^{Obj2}	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	CV^{Obj1}	CV^{Obj2}	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$
S1	3.6	42.50	0.00%	0.00%	0.00%	0.00%	55.8	3.6	42.50	0.00%	0.00%	0.00%	0.00%	51.3
S2	5.6	67.88	0.00%	5.23%	0.00%	2.20%	59.2	5.6	68.60	0.00%	5.32%	0.00%	2.30%	75.3
S3	9.0	131.63	4.10%	3.19%	3.95%	3.54%	188.8	9.0	131.01	4.10%	3.20%	3.95%	3.40%	193.6
M1	18.7	266.28	1.10%	1.20%	0.00%	0.14%	1224.1	18.8	265.45	1.21%	1.12%	0.00%	0.13%	1176.0
M2	27.0	375.38	0.00%	0.71%	0.00%	0.33%	2553.3	27.0	375.22	0.00%	0.68%	0.00%	0.45%	2457.4
M3	41.4	520.57	0.35%	1.93%	0.10%	0.33%	3125.9	41.4	517.06	0.32%	1.86%	0.00%	0.42%	3344.9
L1	58.1	635.93	0.45%	1.88%	1.19%	1.72%	3600.0	58.3	630.89	0.30%	1.67%	0.98%	1.54%	3600.0
L2	71.1	746.12	1.20%	3.02%	0.45%	2.37%	3600.0	71.1	741.69	1.21%	3.21%	0.49%	2.21%	3600.0
L3	98.4	1079.44	0.68%	3.21%	0.41%	2.50%	3600.0	98.5	1074.31	0.65%	2.99%	0.45%	2.67%	3600.0
Average	37.0	429.53	0.88%	1.92%	0.68%	1.46%	2000.8	37.0	427.41	0.87%	2.23%	0.65%	1.46%	2011.0

Upon close examination of the data displayed in Table 7.5, it is evident that the ALNS and LNS heuristics exhibit comparable performance. Minor differences, however, are observable in some specific areas. For instance, the average value of Objective 1, denoted as $\overline{\textbf{Obj. 1}}$, reveals that the ALNS heuristic surpasses its LNS counterpart marginally for the M1, L1, and L3 instance groups. Nonetheless, the average $\overline{\textbf{Obj. 1}}$ across all instance groups for both the ALNS and LNS remains the same. In relation to the average value of Objective 2, $\overline{\textbf{Obj. 2}}$, across all instance groups, the data exhibits a small difference between the ALNS and LNS heuristics, with the ALNS trailing by a marginal value of 2.12. Furthermore, the average coefficient of variation for Objective 1, CV^{Obj1}, indicates a comparative performance between the ALNS and LNS. The average coefficient of variation for Objective 2, represented as CV^{Obj2}, shows that the LNS heuristic scores lower than the ALNS. Lastly, a notable observation lies in the average gaps for both objectives. The ALNS heuristic exhibits lower average value of Gap^{Obj1} compared to the LNS, by 0.03%. In terms of the average value of Gap^{Obj2}, the ALNS and LNS have equal values. These observations suggests that the ALNS, on average, is capable of identifying solutions of marginally superior quality.

The observation that the ALNS is capable of finding only marginally superior solutions compared to the LNS is consistent with the findings of Turkeš et al. (2021). In their meta-analysis on the importance of the adaptive layer in adaptive large neighborhood search (ALNS), Turkeš et al. (2021) found that on average, the addition of an adaptive layer to an ALNS algorithm improves the objective function value by only 0.14% (with a 95% confidence interval of 0.06–0.21%). While the adaptive layer can add value (and indeed does so in a limited number of studies), it also adds complexity.

Upon examining the performance of the destroy and repair operators, a subtle pattern emerges. For the destroy operators, all perform almost equally. However, the worst deviation destroy operator consistently shows a preference or outperforms the others. While the rest of the destroy operators also perform at a similar level, there is no discernible pattern suggesting that any one of these operators is consistently preferred or outperforms the others. This holds true across all instance groups. In contrast, for the repair operators, one operator consistently underperforms or is less preferred, namely the regret-k Repair operator. The insertion and maximum capacity insertion repairs perform on par with each other, whereas the regret-k repair fails to match their performance. This pattern remains consistent across all instance groups, indicating that the insertion and maximum capacity insertion repairs are consistently preferred over the regret-k repair operator. The weight evolution graphs illustrating these findings can be found in Appendix F.

7.2.3 Performance Testing of the ALNS and Its Extensions

This section presents the performance testing of the ALNS heuristic and its extensions, namely local search (LS) and the route combination problem (RCP). The performance instances used to produce the results are the ones presented in Subsection 6.4.2. The first extension to the ALNS heuristic introduced in this thesis is the integration of local search (ALNS + LS). As described in Section 5.6, the local search explore neighboring solutions to improve the candidate solution found in an ALNS iteration. The second extension is solving the RCP in the ALNS heuristic (ALNS + RCP), which is detailed in Section 5.7. The RCP utilizes routes from solutions found in previous iterations of the ALNS heuristic, combining them to discover new and improved solutions for the SRRPFL.

The data in Tables 7.6 and 7.7 provide a detailed comparison of the performance of the

ALNS heuristic and its extensions across the instance groups (S1 to L3). Each extension's effectiveness is evaluated based on these parameters: the average coefficient of variation for Objectives 1 and 2 (CV^{Obj1} and CV^{Obj2}), the average gap for Objectives 1 and 2 (Gap^{Obj1} and Gap^{Obj2}), and the average computational time.

Table 7.6 provides a comparative performance analysis of the basic ALNS heuristic and ALNS + LS. Local search is employed whenever a new global best solution is discovered or when a candidate solution is identified as a *promising solution*, as outlined in Subsection 5.6.1. A review of the data in Table 7.6 reveals that the ALNS + LS outperforms the basic ALNS in terms of average gap for both objectives. More specifically, the ALNS + LS reports an average gap of 0.54% for Objective 1 and 1.78% for Objective 2, which is lower than the basic ALNS. Moreover, the coefficient of variation, which is indicative of the heuristic's reliability, also displays improvement for both objectives with the incorporation of local search. For Objective 1, it falls from 0.87% to 0.64%, and for Objective 2, it falls from 2.23% to 1.71%. However, the integration of local search does come with an increase in average computational time. The average solution time witnesses a rise from 2011.0 seconds for the basic ALNS to 2097.4 seconds for ALNS + LS. Despite this slight increase, the superior performance metrics substantiate the value added by the local search, making it a worthwhile trade-off.

The inclusion of local search in the ALNS heuristic yields improvements in average objective values, most notably in the M1-M3 and L1-L3 instance groups. This observation is confirmed by the lower average gaps detailed in Table 7.6 for ALNS + LS. For the small instance groups (S1, S2, S3), the addition of local search leads to a great reduction in the CV for Objective 2, especially for S2. This suggests that local search can enhance the stability of the results, especially in situations with fewer drivers and passengers, where small differences can have a larger impact. For the large instance groups (L1, L2, L3), the CV for Objective 1 and Objective 2 is generally lower with ALNS + LS than with ALNS alone across all instances.

We observe that the ALNS + LS tends to discover new global best solutions more rapidly compared to the standalone ALNS. This rapid convergence enhances the chances of generating high-quality solutions, even under computational restrictions. Furthermore, this suggests that in this problem domain, new global solutions are often located close to a current solution. It has been noticed that once a local search is triggered, the local search operators that frequently lead to the discovery of a new global best solution are primarily the Intra-Passenger Swap and Candidate Location Shift, as presented in Subsection 5.6.2.

As for the computational time, larger instances (L1, L2, L3) seem to reach the ALNS stopping criteria of 3600 seconds, indicating a considerable increase in computational time with the enlargement of the instance group size. This observation suggests that the problem complexity influences the computational time required by the heuristic.

The application of the local search generally improves the performance of the ALNS heuristic. This improvement can be seen through the decrease in both the CV and the average gaps for Objectives 1 and 2. Notably, the average gaps in all instance groups are lower in the ALNS + LS method than in the standard ALNS, indicating that adding local search helps the ALNS heuristic to find higher quality solutions. Thus, we can conclude that local search indeed enhances the performance of the ALNS heuristic.

Table 7.6: 1. Comparison of results for the ALNS heuristic and its extensions. CV^{Obj1} and CV^{Obj2} represent the average coefficient of variation for Objectives 1 and 2 for each instance group, respectively. Gap^{Obj1} and Gap^{Obj2} represent the average gap for Objectives 1 and 2 for each instance group across all ALNS configurations (ALNS, ALNS + LS, ALNS + RCP, ALNS + LS + RCP), respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds

Instance	**ALNS**					**ALNS + LS**				
Group	CV^{Obj1}	CV^{Obj2}	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$	CV^{Obj1}	CV^{Obj2}	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$
S1	0.00%	0.00%	0.00%	0.00%	51.5	0.00%	0.00%	0.00%	0.00%	47.7
S2	0.00%	5.32%	0.00%	3.19%	75.3	0.00%	2.86%	0.00%	2.09%	98.8
S3	4.10%	3.20%	3.95%	3.23%	193.6	3.59%	3.04%	3.10%	2.53%	256.4
M1	1.21%	1.21%	0.68%	1.25%	1176.0	0.59%	1.00%	0.43%	1.26%	1272.4
M2	0.00%	0.68%	0.00%	0.32%	2457.4	0.00%	0.41%	0.00%	0.36%	2949.8
M3	0.32%	1.86%	0.15%	2.45%	3344.9	0.21%	1.34%	0.10%	2.01%	3451.8
L1	0.30%	1.67%	0.20%	1.82%	3600.0	0.18%	1.54%	0.13%	1.76%	3600.0
L2	1.21%	3.21%	0.89%	3.67%	3600.0	0.84%	2.49%	0.70%	3.13%	3600.0
L3	0.65%	2.99%	0.46%	2.95%	3600.0	0.36%	2.75%	0.37%	2.84%	3600.0
Average	0.87%	2.23%	0.70%	2.10%	2011.0	0.64%	1.71%	0.54%	1.78%	2097.4

As described in Subsection 7.2.1, the route combination problem (RCP) is solved every 300 iterations (I^{RCP}) of the ALNS heuristic. Table 7.7 presents the results for the ALNS heuristic with the RCP. Comparing the performance of ALNS + RCP against ALNS + LS, as shown in Table 7.6, we observe that ALNS + RCP demonstrates superior performance. The average coefficient of variation for Objective 1, under ALNS + RCP, stands at 0.17%, substantially lower than the 0.64% observed with ALNS + LS. Similarly, the average coefficient of variation for Objective 2 is also reduced to 0.82% in ALNS + RCP, a notable improvement from the 1.71% recorded under ALNS + LS. In addition to these metrics, the average gap observed for both objectives under ALNS + RCP also sees notable improvement, with Objective 1 and Objective 2 recording gaps of 0.10% and 0.86%, respectively. This performance edge is reinforced when we examine the average computational time, where ALNS + RCP outperforms ALNS + LS. Based on this preliminary data analysis, it can be inferred that the ALNS algorithm, when paired with the RCP, offers more consistent results, higher accuracy in meeting objectives, and superior computational efficiency compared to when it is combined with local search.

Upon examining Table 7.7, it becomes clear that ALNS + RCP substantially improves the coefficient of variation for both objectives across all instance groups. In certain instance groups, the coefficient of variation (CV) for Objective 1 has been entirely eliminated, dropping to 0.00%, which constitutes a notable enhancement compared to ALNS + LS. We also observe a similar reduction in the gaps; instance groups have experienced gaps reduced to 0.00% for Objective 1. The same degree of improvement is observed across all instances for the gap in Objective 2. Every instance group has seen reductions in the gap, thereby emphasizing the effectiveness of ALNS + RCP in overall performance. Upon examination of the generated solutions, it becomes apparent that ALNS + RCP is highly effective in identifying new global best solutions. Based on these findings, we conclude that the introduction of RCP improves the ALNS heuristic performance, performing better

than the ALNS and ALNS + LS.

The reason for the superior performance of a set-partitioning formulation (Section 5.7), over the local search formulation (Section 5.6) likely stems from the inherent characteristics of the routing problem itself. The SRRPFL is combinatorial in nature, meaning that it involve the arrangement of elements within a set in some specific order. Local search formulations typically perform well on such problems by iteratively searching the nearby solution space for improvements. However, this can also limit their performance, as they might be prone to local optima and may not explore the broader solution space effectively. This is a point of divergence for the RCP, enabling it to exploit the problem's inherent combinatorial structure more efficiently. The set-partitioning approach taken by the RCP in the ALNS allows for the aggregation and simultaneous examination of various routes discovered over the course of the algorithm's execution. By doing so, it allows the identification of combinations of routes that yield an overall better solution, even if those routes individually might not have been the best ones. The ALNS collects a vast pool of discovered routes for each iteration, and after every 300 iterations, the RCP is solved. This process lets the ALNS + RCP identify and exploit synergies between routes that might not be evident when considering routes independently, as is the case in local search. Furthermore, the RCP provides a systematic and compact representation of feasible solutions, taking into consideration the entire path or route of drivers and passengers, rather than individual arcs. This path-flow model provides an alternative perspective, enabling a broader and more holistic view of the problem, enhancing the overall solution quality. In conclusion, the reasons for the superior performance of ALNS + RCP likely lie in its capacity to exploit the combinatorial nature of the routing problem and its systematic consideration of route combinations.

Table 7.7: 2. Comparison of results for the ALNS heuristic and its extensions. CV^{Obj1} and CV^{Obj2} represent the average coefficient of variation for Objectives 1 and 2 for each instance group, respectively. Gap^{Obj1} and Gap^{Obj2} represent the average gap for Objectives 1 and 2 for each instance group across all ALNS configurations (ALNS, ALNS + LS, ALNS + RCP, ALNS + LS + RCP), respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds

Instance	**ALNS + RCP**					**ALNS + LS + RCP**				
Group	CV^{Obj1}	CV^{Obj2}	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$	CV^{Obj1}	CV^{Obj2}	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$
S1	0.00%	0.00%	0.00%	0.00%	58.0	0.00%	0.00%	0.00%	0.00%	53.5
S2	0.00%	2.79%	0.00%	2.04%	94.3	0.00%	0.00%	0.00%	0.00%	109.7
S3	1.02%	1.41%	0.45%	2.43%	197.1	0.00%	0.08%	0.00%	0.04%	274.6
M1	0.00%	0.08%	0.00%	0.06%	1158.4	0.00%	0.07%	0.00%	0.03%	1408.9
M2	0.00%	0.06%	0.00%	0.06%	2451.8	0.00%	0.05%	0.00%	0.10%	2990.2
M3	0.21%	1.00%	0.10%	1.35%	3289.0	0.21%	0.78%	0.10%	1.23%	3400.2
L1	0.00%	0.40%	0.00%	0.38%	3600.0	0.00%	0.18%	0.00%	0.19%	3600.0
L2	0.16%	0.68%	0.17%	0.57%	3600.0	0.00%	0.49%	0.00%	0.38%	3600.0
L3	0.18%	0.99%	0.21%	0.83%	3600.0	0.18%	0.75%	0.21%	0.99%	3600.0
Average	0.17%	0.82%	0.10%	0.86%	2005.4	0.04%	0.27%	0.03%	0.33%	2115.2

Finally, in Table 7.7, we present the outcomes of our exploration into the performance of the ALNS configuration that includes both local search and the RCP (ALNS + LS + RCP). Our findings suggest that this configuration yields the lowest average coefficient of variation across all examined scenarios: 0.04% for Objective 1 and 0.27% for Objective 2. The average gaps for this configuration, standing at 0.03% for Objective 1 and 0.33% for Objective 2, also outperform those observed in the other configurations (ALNS, ALNS + LS, ALNS + RCP). Although the average computational time of 2115.2 seconds mirrors that of ALNS + LS, the superior outcomes demonstrate the value of integrating both local search and RCP in the ALNS framework.

As compared to the ALNS + RCP configuration, the ALNS + LS + RCP configuration either matches or improves the coefficient of variation across all instance groups. Specifically, for Objective 1, the enhancement is particularly pronounced for the L2 instance group, where the CV^{Obj1} reduces from 0.16% to 0.00%. As for Objective 2, the ALNS + LS + RCP configuration produces several reductions in the CV^{Obj2}, indicating decreased variation in solution outcomes. The most notable decrease is observed in the S2 instance group, with the CV^{Obj2} plummeting from 2.79% to 0.00%. Examining the gaps reveals similar improvements, with the ALNS + LS + RCP configuration either matches or improves the gap for most of the instance group. The instance group that does not follow this pattern is L3, where the gap increasea from 0.83% to 0.99% compared to ALNS + RCP. Overall, these observations highlight the added value that the simultaneous inclusion of local search and RCP brings to the ALNS heuristic, supporting its effectiveness in generating high-quality solutions for the routing problem at hand.

Based on the findings presented in this subsection, we observe that the ALNS configuration that includes both local search and the RCP produces the best average performance, and conclude that this is the best ALNS configuration. Thus, this configuration will be used in the proceeding analysis and chapters, and is referred to as the ALNS heuristic.

A comparative analysis between the ALNS and the construction heuristic, examining their respective performances and convergence tendencies, has been carried out for a more comprehensive understanding of their efficiency. This analysis can be found in Appendix G. The summarized comparison between the ALNS extensions is displayed in Table 7.8.

Table 7.8: Comparison of results between the ALNS extensions. Gap^{Obj1} and Gap^{Obj2} represent the average gap for Objectives 1 and 2 for each instance group across the ALNS extension, respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds

Instance	**ALNS**			**ALNS + LS**			**ALNS + RCP**			**ALNS + LS + RCP**		
Group	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$
S1	0.00%	0.00%	51.5	0.00%	0.00%	47.7	0.00%	0.00%	58.0	0.00%	0.00%	53.5
S2	0.00%	3.19%	75.3	0.00%	2.09%	98.8	0.00%	2.04%	94.3	0.00%	0.00%	109.7
S3	3.95%	3.23%	193.6	3.10%	2.53%	256.4	0.45%	2.43%	197.1	0.00%	0.04%	274.6
M1	0.68%	1.25%	1176.0	0.43%	1.26%	1272.4	0.00%	0.06%	1158.4	0.00%	0.03%	1408.9
M2	0.00%	0.32%	2457.4	0.00%	0.36%	2949.8	0.00%	0.06%	2451.8	0.00%	0.10%	2990.2
M3	0.15%	2.45%	3344.9	0.10%	2.01%	3451.8	0.10%	1.35%	3289.0	0.10%	1.23%	3400.2
L1	0.20%	1.82%	3600.0	0.13%	1.76%	3600.0	0.00%	0.38%	3600.0	0.00%	0.19%	3600.0
L2	0.89%	3.67%	3600.0	0.70%	3.13%	3600.0	0.17%	0.57%	3600.0	0.00%	0.38%	3600.0
L3	0.46%	2.95%	3600.0	0.37%	2.84%	3600.0	0.21%	0.83%	3600.0	0.21%	0.99%	3600.0
Average	0.70%	2.10%	2011.0	0.54%	1.78%	2097.4	0.10%	0.86%	2005.4	0.03%	0.33%	2115.2

7.3 Comparing ALNS to the Commercial Solver

This section provides a comparison of the solutions produced by the adaptive large neighborhood search (ALNS) heuristic and the commercial solver, Gurobi. As outlined in Section 7.1, the commercial solver has a predefined stopping criterion. If Gurobi does not find a solution within the allocated time frame of 3600 seconds for any given instance, the solver will cease its operation. By comparing the results of the ALNS heuristic with those of Gurobi, we can evaluate the heuristic's quality. The ultimate goal is not merely to match the performance of established commercial solvers but to investigate potential scenarios where the ALNS heuristic might offer unique advantages, such as superior performance on specific types of problems or quicker solution times under certain conditions.

Since the SRRPFL is a multi-objective optimization problem, the current version of Gurobi cannot produce the gap, upper, and lower bounds for both objectives. Additionally, Gurobi is deterministic, and therefore only one run of each instance is required. The gaps, referred to as Gap^{GObj1} and Gap^{GObj2}, represent the discrepancy between the average objective value of the instance group found by the ALNS heuristic and that found by the commercial solver. This is calculated by taking the difference between the average objective values for each instance group as determined by Gurobi and the average objective values for each instance group as found by the ALNS heuristic, then dividing by the average objective values for each instance group as determined by Gurobi.

Table 7.9 presents the results from Gurobi and the ALNS heuristic. As can be observed, the commercial solver achieves optimality for instance groups S1, S2, and S3. Comparatively, the ALNS heuristic finds the same objective values for instance groups S1 and S2, but not for S3. In S3, the ALNS heuristic finds a lower Objective 1 than Gurobi in only two runs of a single instance. For the instance group M1, the commercial solver reaches its time limit of 3600 seconds without achieving optimality. In this instance group, the ALNS heuristic outperforms the commercial solver, resulting in negative values for Gap^{GObj1} and Gap^{GObj2}. For Gap^{GObj1}, we can observe that it reaches a negative value of -28.76%, which is a noteworthy result. It demonstrates that the ALNS heuristic produces solutions of substantially higher quality than the commercial solver within the 3600 seconds time limit. We can also observe an increase in the average time used by the commercial solver between instance groups S1 to M1. For the subsequent instance groups, the commercial solver is unable to find solutions within the 3600 seconds time limit as the problem size increases. For these instance groups, the commercial solver ran for over 48 hours without showing any indication of initialization of solving the instances.

By comparing the ALNS heuristic with the commercial solver, it becomes evident that the commercial solver struggles with real-sized instances of the SRRPFL. In contrast, the ALNS heuristic demonstrates its capability to find potential high-quality solutions for these more demanding, real-sized instances. Based on our comparative analysis with Gurobi, the ALNS heuristic demonstrates promising capabilities, potentially offering high-quality solutions even for complex and large-scale instances. However, we acknowledge that the actual quality of these solutions can only be definitively assessed in the context of real-world applications or through comparison with other established solvers. As a conclusion, the ALNS heuristic may be a valuable alternative to the existing commercial solvers for tackling the SRRPFL.

Table 7.9: Comparison of results for Gurobi and the ALNS heuristic with LS and RCP. The column **Instance Group** shows the instance groups with the number of passengers in each instance group. $\overline{\textbf{Obj. 1}}$ and $\overline{\textbf{Obj. 2}}$ represent the average objective values for Objectives 1 and 2 for each instance group, respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds. Gap^{GObj1} and Gap^{GObj2} represent the average gap for Objectives 1 and 2 for each instance group across the commercial solver and the ALNS heuristic

Instance	**Gurobi**			**ALNS + LS + RCP**				
Group	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	Gap^{GObj1}	Gap^{GObj2}	$\overline{\textbf{Time [s]}}$
S1 (4)	3.6	42.50	3.8	3.6	42.50	0.00%	0.00%	53.5
S2 (6)	5.6	66.42	101.2	5.6	66.42	0.00%	0.00%	109.7
S3 (10)	9.5	133.26	638.1	9.4	132.42	1.05%	-0.63%	274.6
M1 (20)	14.6	290.88	3600.0	18.8	265.96	-28.76%	-8.57%	1408.9
M2 (30)	-	-	-	27.0	374.20	-	-	2990.2
M3 (42)	-	-	-	41.4	507.91	-	-	3400.2
L1 (60)	-	-	-	59.0	616.53	-	-	3600.0
L2 (75)	-	-	-	71.8	721.01	-	-	3600.0
L3 (100)	-	-	-	99.0	1039.19	-	-	3600.0
Average	-	-	-	37.3	418.46	-	-	2115.2

Chapter 8

Managerial Insights

Chapter 7 outlines the results derived from applying the ALNS heuristic (ALNS + LS + RCP) to a variety of test instances. A close examination of these results allows us to highlight the potential impacts of ridesharing. In Section 8.1, we demonstrate the benefits of ridesharing, particularly in terms of potential reductions in traffic congestion and CO_2 emissions. Section 8.2 discusses the effects of changing maximum travel times for drivers and passengers. Following this, Section 8.3 discusses the benefits of incorporating candidate locations.

8.1 The Value of Ridesharing

Reducing Traffic Congestion

One of the main challenges faced by people traveling from Sotra to the greater Bergen area during morning hours is the traffic congestion caused by Sotrabroen. The results presented in Table 7.9 clearly show that the ALNS heuristic with local search (LS) and the route combination problem (RCP) produces solutions where a majority of potential passengers are picked up. Upon analyzing the three larger instances (L1, L2, and L3), it is observed that, on average, over 95.0% of potential passengers are picked up and delivered. Although the largest instance group only represents approximately 15.0% of the actual number of drivers and passengers traveling from Sotra to the greater Bergen area every hour between 06:00 and 10:00 (Telia, 2022), the results indicate that ridesharing can reduce the number of cars on the road, given that the participants are willing to participate.

Before investigating the implications of reduced traffic congestion on total travel time, it is important to compare the variations in travel time between scenarios with and without ridesharing, while making the initial assumption that the number of cars on the road does not influence the waiting time in potential traffic. The scenario without ridesharing refers to a situation where each driver and passenger drives directly from their origin location to their destination locations, without any additional stops to pick up or deliver other passengers. Figure 8.1 illustrates the average additional travel time incurred through the scenario with ridesharing, under the aforementioned assumption, as compared to the scenario without ridesharing. The results show that the average extra travel time remains relatively stable, hovering between 30.0% and 40.0%, across all the small and medium instance groups. Upon examining the three large instance groups, it becomes evident that the average extra travel time incurred through ridesharing decreases. The difference

in travel time with and without ridesharing is approximately 10.0% for instance group L3. This marked decrease in extra travel time in larger instance groups suggests that ridesharing becomes more efficient as the system scales up. As the number of drivers and passengers increases, the opportunity for convenient pairing also increases, thereby reducing the extra travel time for each individual.

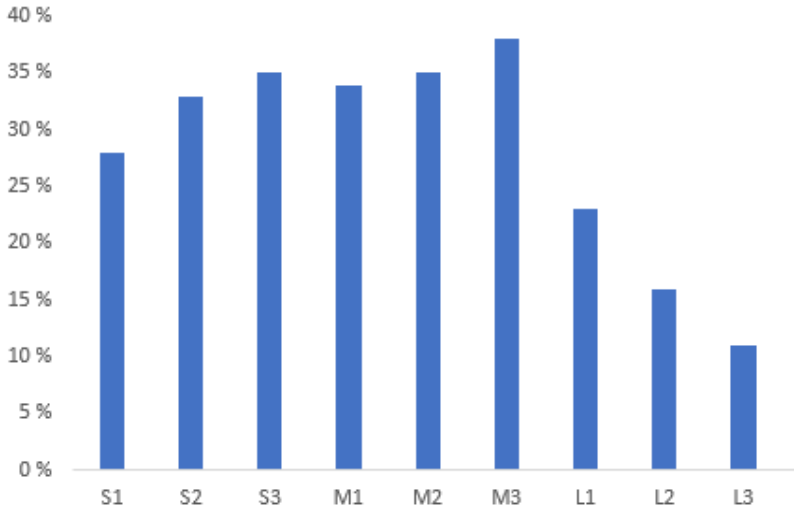


Figure 8.1: The average extra travel time for drivers in each instance group with ridesharing

Although Figure 8.1 indicates that ridesharing can cause an approximate increase in total travel time of 10.0% for instance group L3, it is crucial to note that this is in comparison to a scenario without any ridesharing. To get a broader understanding, one should consider the potential additional waiting times due to queuing on the roads (or Sotrabroen) that would be incurred by all drivers and passengers if they each had to drive their own cars. Considering this, the results could look different. According to a study by the Norwegian Automobile Federation (NAF), reducing the number of cars in congested areas in Norway by 15.0% could largely alleviate traffic congestion (NAF, 2015). Table 8.1 illustrates the impact on the number of cars reduced through ridesharing. The **Cars Red.** column describes the average percentage of cars reduced with ridesharing. For example, in the L3 instance group consisting of 35 drivers and 100 passengers, the ALNS heuristic produced a solution where an average of 99.0 passengers are picked up in that instance group. Comparing this to the situation without any ridesharing, the number of cars on the road is reduced by 73.3%. This reduction should not be underestimated. As highlighted by the NAF study, even a 15.0% reduction in the number of cars in congested areas could largely alleviate traffic congestion. Therefore, a reduction of over 70.0%, as seen in most instance groups in Table 8.1, points towards potential improvements in traffic flow and considerable reductions in travel times, benefiting both individuals and society as a whole.

Table 8.1: Overview of the number of cars reduced implementing the solutions from the ALNS heuristic. $\overline{\textbf{Obj. 1}}$ represents the average objective values for Objectives 1. $\overline{\textbf{Cars Red.}}$ is the average percentage of cars that will be reduced in ridesharing for an instance group, produced by the ALNS heuristic

Instance Group	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Cars Red.}}$
S1	3.6	72.0%
S2	5.6	70.0%
S3	9.4	67.1%
M1	18.8	67.1%
M2	27.0	64.3%
M3	41.4	71.4%
L1	59.0	73.7%
L2	71.8	71.8%
L3	99.0	73.3%

Reducing CO_2 Emissions

Besides the possibility of reducing traffic congestion, the implementation of ridesharing also offers an opportunity for substantial environmental impact. By pooling drivers and passengers into shared vehicles, the total distance driven can be reduced, addressing not only congestion issues, but also reduction of CO_2 emissions. Thus, ridesharing can make a contribution to preserving the environment.

By conducting a comparison between scenarios with and without ridesharing, it becomes possible to quantify the extent of savings that can be achieved through the implementation of ridesharing. Table 8.2 presents an overview of the average distance traveled in all test instances, considering both scenarios with and without ridesharing, along with the percentage reduction in kilometers between the two variations. After examining the $\overline{\textbf{Reduction}}$ column, it becomes clear that the largest test instances exhibit the most substantial reduction in kilometers driven. Note that a reduction in total kilometers driven directly translates to a decrease in CO_2 emissions. As mentioned, the largest test instance only covers approximately 15.0% of the total number of travelers passing Sotrabroen every weekday in the morning hours. As a result, the potential reduction in CO_2 emissions greatly surpasses the values presented in this thesis.

Table 8.2: Overview of the average number of kilometers traveled for each driver and passenger in each test instance with and without ridesharing. The column **Instance Group** shows the instance groups with the number of passengers in each instance group. **w/o RS** is the average number of kilometers driven in each test instance without ridesharing. **w RS** states the average number of kilometers driven in each test instance with ridesharing. **Reduction** is the percentage reduction in kilometers that comes from implementing ridesharing solutions

Instance	Total Kilometers Traveled		
Group	w/o RS	w RS	Reduction
S1 (4)	96.22	65.87	31.5%
S2 (6)	151.11	68.50	54.7%
S3 (10)	268.37	150.84	43.8%
M1 (20)	679.47	322.09	52.6%
M2 (30)	1029.45	412.42	59.9%
M3 (42)	1160.65	530.61	54.3%
L1 (60)	1598.22	554.22	65.3%
L2 (75)	2102.06	690.36	67.2%
L3 (100)	2934.00	926.53	68.3%

8.2 The Value of Maximum Travel Time

In the context of the SRRPFL, the parameter for maximum travel time (T_k^M) represents a critical factor in determining the attractiveness and efficiency of the system. As described in Subsection 6.3.2, this parameter is determined for each driver and passenger as a multiplier of their direct travel time from origin to destination location. The ALNS heuristic was configured with a multiplier set at 1.7x for the computational study conducted in Chapter 7, implying that both drivers and passengers are willing to tolerate travel times up to 70% longer than their direct route.

However, the value of this maximum travel time is subject to variation, depending on the tolerances and preferences of individual drivers and passengers. To account for this variation and to understand its impact on our model, we undertake an investigation into the effects of altering the maximum travel time multiplier. We adjust the multiplier from its base case of 1.7x to other values such as 1.1x, 1.3x, 1.5x, and 1.9x. The goal of this exploration is to observe and analyze the effects these changes have on the overall utility of the ridesharing, total travel time, and the computation time of the ALNS heuristic.

Table 8.3 illustrates the impact of varying maximum travel time multipliers on the average number of passengers picked up, the average total travel time for all drivers, and the average computation time for the ALNS heuristic. As we can observe, changes in maximum travel times notably affect the number of picked up passengers (**Obj. 1**). For instance, when comparing the 1.7x and 1.9x multipliers, all instance groups exhibit a higher number of passengers picked up with the 1.9x multiplier than with the 1.7x. Additionally, the lowest multiplier, 1.1x, results in a considerable decrease in the number of passengers picked up compared to the larger multipliers. Despite these changes, the computational time of the ALNS heuristic for each multiplier configuration shows minor variation.

Furthermore, the investigation into varying this parameter illuminates its notable impact on the number of passengers picked up. As the multiplier increases, we see a corresponding increase in the number of passengers served, indicating a more efficient system. The increase in the maximum travel time essentially broadens the pool of feasible rideshare matches by allowing for longer detours. It can accommodate a larger number of passengers that might otherwise be discarded due to exceeding the initial maximum travel time constraint. However, the willingness to accept longer travel times (higher multipliers) is heavily dependent on individual user preferences, balancing convenience, and the benefits offered by ridesharing. Conversely, a lower multiplier could potentially limit the system's capacity to serve more passengers, limiting the system's overall efficiency and effectiveness.

Table 8.3: Comparison of results for different values of maximum travel time multiplier. $\overline{\textbf{Obj. 1}}$ and $\overline{\textbf{Obj. 2}}$ represent the average objective values for Objectives 1 and 2 for each instance group, respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds. The ALNS heuristic was configured with a multiplier set at 1.7x for the computational study conducted in Chapter 7

Instance	**1.1x**			**1.3x**			**1.5x**			**1.7x**			**1.9x**		
Group	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$
S1	1.4	32.17	21.3	2.8	37.29	36.2	3.4	39.36	60.1	3.6	42.50	53.5	3.8	45.70	62.4
S2	2.0	53.55	41.2	4.8	59.91	72.5	5.4	62.36	99.3	5.6	66.42	109.7	5.8	73.20	110.7
S3	4.1	103.54	162.5	7.4	114.05	262.9	8.6	122.12	219.2	9.4	132.42	274.6	9.8	142.21	271.7
M1	8.8	203.10	1157.7	12.4	203.10	1242.0	16.0	236.19	1274.9	18.8	265.96	1408.9	19.5	274.43	1203.7
M2	17.6	313.50	2976.7	21.8	313.50	2268.4	25.6	365.66	2294.7	27.0	374.20	2990.2	28.6	415.35	2527.7
M3	25.2	435.30	3600.0	35.7	435.30	3350.1	39.4	475.62	3513.7	41.4	507.91	3400.2	41.6	501.54	3546.2
L1	43.2	523.79	3600.0	53.0	556.75	3600.0	57.9	601.44	3600.0	59.0	616.53	3600.0	59.2	621.99	3600.0
L2	50.4	594.84	3600.0	61.4	627.15	3600.0	68.1	690.27	3600.0	71.8	721.01	3600.0	73.8	755.31	3600.0
L3	75.9	895.39	3600.0	90.2	938.22	3600.0	96.3	1003.37	3600.0	99.0	1039.19	3600.0	99.8	1060.72	3600.0
Average	25.4	350.58	2085.4	32.2	365.04	1999.6	35.6	399.42	2030.3	37.3	418.46	2115.2	38.0	432.72	2059.2

8.3 The Value of Candidate Locations

Comparing Ridesharing With and Without Candidate Locations

Incorporating candidate locations into a ridesharing system enhances the flexibility for both drivers and passengers. This additional element allows the SRRPFL to discover solutions that can accommodate more passengers while maintaining or even reducing total travel time for drivers. Table 8.4 provides a comparison of results for the ALNS heuristic with LS and RCP, with ($\rho = 10$ min) and without ($\rho = 0$ min) candidate locations. As recalled from Subsection 6.3.2, ρ is denoted as the maximum time a passenger is willing to travel to reach a candidate pick-up location. Here, $\rho = 0$ min implies that the only pick-up location for passengers is their origin location.

The results in Table 8.4 clearly illustrate the impact of introducing candidate locations. There is an average increase in the number of passengers picked up, with the average value of $\overline{\textbf{Obj. 1}}$ across all instance groups rising from 35.1 without candidate locations to 37.3 with them. Furthermore, candidate locations contribute to shorter average travel times for drivers, with the average value of $\overline{\textbf{Obj. 2}}$ across all instance groups decreasing from 477.25 without candidate locations to 418.46 with them.

The inclusion of candidate locations does not substantially compromise passenger convenience. This conclusion is based on the observation that candidate locations, when available, are frequently used, as indicated by the high average **CP** value at 82.82%. In addition, the average travel time for a passenger to reach a candidate location is not excessively long at 4.15 minutes (and in any case no longer than 10 minutes for any passenger ($\rho = 10$ min), indicating that candidate locations do not impose undue travel burdens on the passengers. On the other hand, the introduction of candidate locations does impact the average computational time of the ALNS heuristic. With candidate locations, the average computational time, $\overline{\textbf{Time [s]}}$, nearly doubles from 1154.6 seconds to 2115.2 seconds. This increase can be attributed to the additional possibilities that the ALNS heuristic needs to consider, leading to a generally higher computational time.

Despite the increased computational time, this analysis highlights the utility of candidate locations in ridesharing scenarios. Their use presents an effective strategy for balancing the dual objectives of maximizing passenger pick-ups and minimizing driver travel time. Hence, the incorporation of candidate locations into a ridesharing model could greatly enhance its practical applicability.

Table 8.4: Comparison of results for the ALNS heuristic with LS and RCP, with and without candidate locations. $\overline{\textbf{Obj. 1}}$ and $\overline{\textbf{Obj. 2}}$ represent the average objective values for Objectives 1 and 2 for each instance group, respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds. $\overline{\textbf{CP}}$ denotes the average percentage of picked up passengers who use a candidate location other than its origin. $\overline{\textbf{TT [min]}}$ refers to the average travel time it takes for passengers who are picked up at a candidate location, to travel to that specific candidate location, measured in minutes

Instance	Without Candidate Locations			With Candidate Locations				
Group	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{CP}}$	$\overline{\textbf{TT [min]}}$	$\overline{\textbf{Time [s]}}$
S1	2.6	50.35	15.9	3.6	42.50	100.00%	4.01	53.5
S2	4.0	89.30	31.2	5.6	66.42	69.67%	4.22	109.7
S3	8.0	149.89	87.2	9.4	132.42	77.85%	4.10	274.6
M1	16.2	299.40	121.1	18.8	265.96	82.30%	4.87	1408.9
M2	25.0	436.89	262.4	27.0	374.20	86.57%	4.63	2990.2
M3	39.0	553.60	462.0	41.4	507.91	81.04%	3.96	3400.2
L1	57.7	717.68	2221.5	59.0	616.53	80.82%	3.90	3600.0
L2	67.3	806.10	3588.3	71.8	721.01	84.15%	4.03	3600.0
L3	97.9	1173.89	3600.0	99.0	1039.19	83.00%	3.66	3600.0
Average	35.1	477.25	1154.6	37.3	418.46	82.82%	4.15	2115.2

Figure 8.2 extends the information presented in Figure 8.1. It shows how much additional travel time is needed for each instance group, comparing the cases with and without candidate locations. The findings presented in Table 8.4 are mirrored in these results. Essentially, it is clear that by employing candidate locations, the overall travel time for all drivers can be decreased.

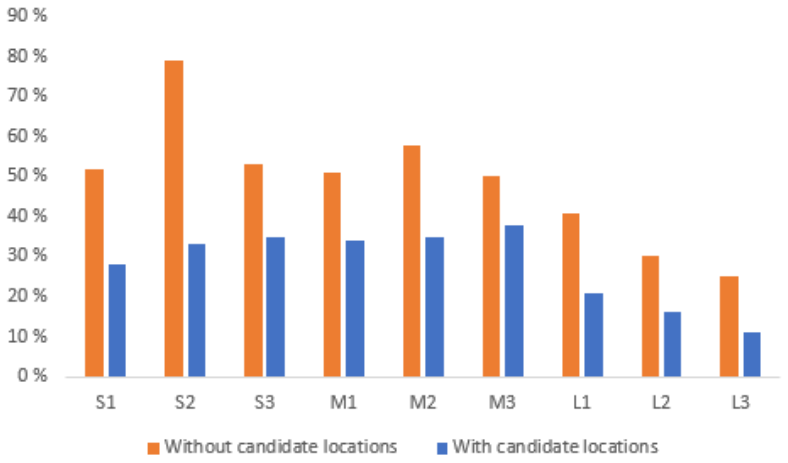


Figure 8.2: The average extra travel time for drivers in each instance group with ridesharing

Figure 8.3 and Figure 8.4 provide an illustration of the utilization of candidate locations. These figures depict the results obtained from running instance S1-1D-4P-1 from the instance group S1, both with and without candidate locations. This particular instance consists of a single driver and four passengers.

Figure 8.3 presents the route that the driver takes without the consideration of candidate locations. The driver embarks on their journey from the origin location at Ågotnes and proceeds to pick up three passengers from their respective origin locations in Straume, Brattholmen, and Arefjord. It is apparent that the driver has to traverse each origin location to collect the passengers. Upon delivering the passengers, two are destined for Solheim, while the remaining passenger, as well as the driver, head towards Bergen sentrum.

Figure 8.4 presents the driver's route when candidate locations are taken into account. In this scenario, the driver is capable of accommodating all four passengers. To achieve this, the passenger from Foldnes navigates to a candidate location in Bildøy, and a similar approach is taken by the passenger from Straume. Consequently, the driver picks up both passengers at Bildøy. Subsequently, the passengers from Arefjord and Brattholmen move to their respective candidate location at Knarrevik, where they are picked up by the driver. The passenger from Foldnes, whose destination is Loddefjord, is the first to be delivered. The remaining passengers are delivered in a similar fashion. It is likely that the inability of the driver to pick up the passenger from Foldnes in the scenario without candidate locations is due to exceeding the driver's maximum travel time. Moreover, as depicted in Figure 8.4, the driver's route with candidate locations is notably shorter than the one without candidate locations.

In conclusion, the utilization of candidate locations clearly illustrates an increase in the overall efficiency of the ridesharing system. Not only does it enable the driver to accommodate more passengers, thereby maximizing ride-sharing potential and reducing individual vehicle usage, it also notably reduces the driver's total travel distance, thereby minimizing fuel consumption. Additionally, by offering passengers the option to navigate to nearby candidate locations, ridesharing systems can better distribute pickup points, leading to better route planning and lesser chances of overburdening certain routes or locations.

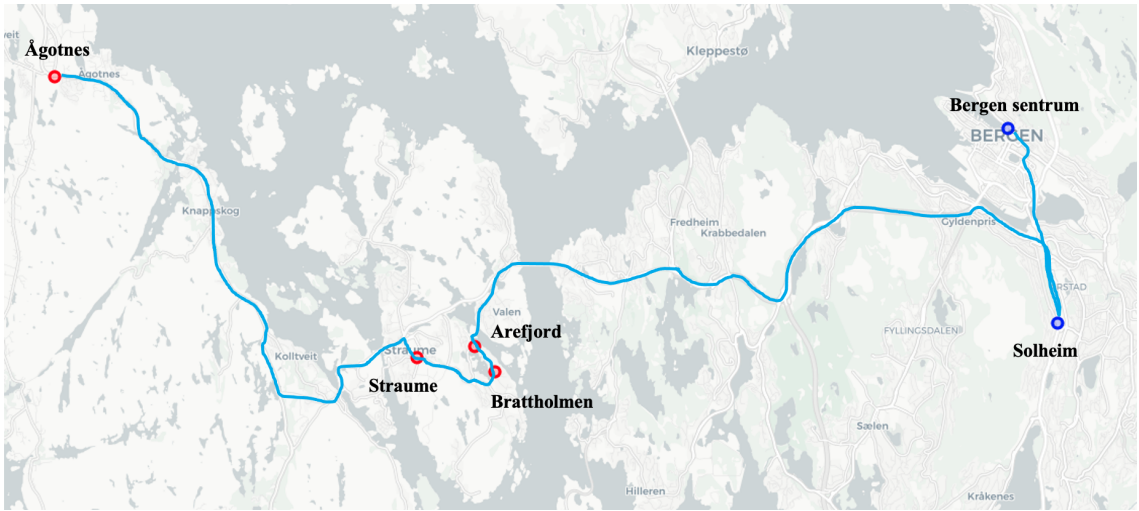


Figure 8.3: Driver's route without candidate locations for InstanceID S1-1D-4P-1, depicting the driver starting from Ågotnes and picking up passengers from Straume, Brattholmen, and Arefjord

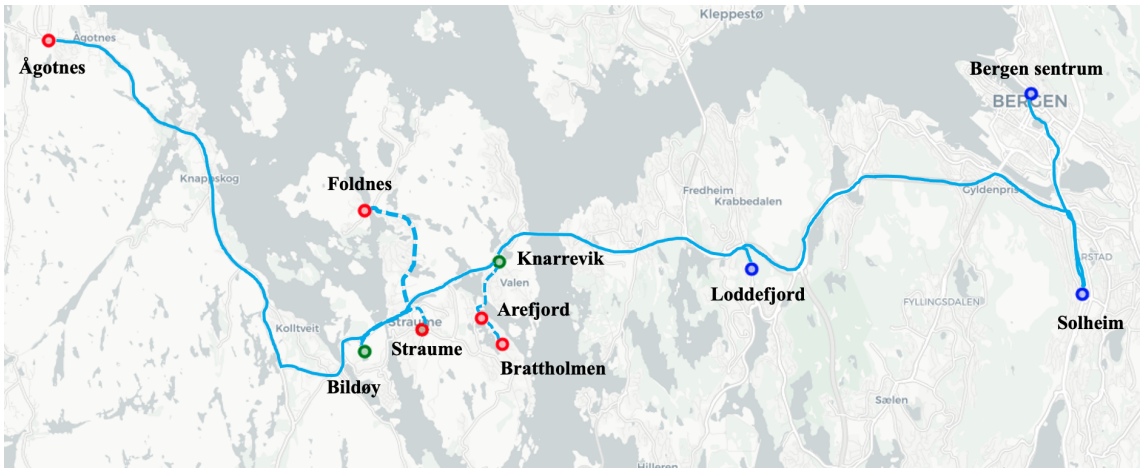


Figure 8.4: Driver's route with candidate locations for InstanceID S1-1D-4P-1, illustrating the driver's ability to pick up all passengers by utilizing candidate locations in Bildøy and Knarrevik

Comparing Varying Numbers of Candidate Locations for Passengers

The evaluation of the value of candidate locations may also take into account the quantity of candidate locations per passenger. As outlined in Section 6.2, the θ parameter describes the θ closest origin locations to be included as candidate locations for each passenger. The ALNS heuristic was configured with $\theta = 3$ for the computational study conducted in Chapter 7. The tested values of θ in this subsection include: $\theta = 1$, $\theta = 3$, $\theta = 6$, and $\theta = 9$, where $\theta = 1$ includes only the nearest candidate location, $\theta = 3$ includes the three nearest candidate locations, and so on.

Table 8.5 shows how changing θ values affects the average number of passengers picked up, the average total travel time for all drivers, and the average computation time for the ALNS heuristic. The $\overline{\textbf{Obj. 1}}$ column shows that as θ increases, so does the average number of passengers picked up across all instance groups. For example, when $\theta = 1$, the average value of $\overline{\textbf{Obj. 1}}$ across all instance groups is 36.3. This average value increases as we increase θ. For instance, when $\theta = 9$, the average value of $\overline{\textbf{Obj. 1}}$ across all instance groups becomes 37.8. This observation suggests that when passengers have more candidate locations to choose from, more of them get picked up.

Furthermore, it appears that as θ increases, the average total travel time for drivers, shown in the $\overline{\textbf{Obj. 2}}$ column, tends to decrease. For the same values of $\overline{\textbf{Obj. 1}}$ across the instance groups, a larger θ value generally results in either the same or reduced total travel time for all drivers compared to smaller θ values. However, this is not always the case. For example, in the instance group L3, the $\overline{\textbf{Obj. 2}}$ is 1030.94 for $\theta = 6$, which is lower than the value of 1046.66 at $\theta = 9$. This observation aligns with the lexicographic objective functions described in Subsection 4.2.1, because the ALNS heuristic finds solutions where more passengers are picked up when $\theta = 9$. Lastly, the computational time of the ALNS heuristic, displayed in the $\overline{\textbf{Time [s]}}$ column, increases with the values of θ. For example, the average computational time for instance group M1 with $\theta = 3$ is 1408.9 seconds, and with $\theta = 9$ it is 2628.2 seconds. This notable increase of computational time with increasing values of θ is likely due to the greater number of candidate locations that the ALNS heuristic needs to process.

In conclusion, the value of θ can considerably influence the effectiveness of the ridesharing model. By enabling a greater number of candidate locations for each passenger to choose from, the system can accommodate more passengers while potentially reducing the total

travel time for drivers. However, varying values of θ have less notable impact on $\overline{\textbf{Obj. 1}}$ in comparison to varying maximum travel times from Section 8.2. Furthermore, this does come with the trade-off of increased computation time due to the additional candidate locations the ALNS heuristic must process. Therefore, in deploying this model, a balance must be sought between optimizing rideshare success and travel efficiency, and managing computational resources. This emphasizes the need for ridesharing systems to be adaptable, enabling them to adjust parameters such as θ to suit specific operational contexts or objectives, whether it's maximizing passenger service, minimizing driver travel time, or optimizing computational efficiency.

Table 8.5: Comparison of results for different values of θ, where θ represent the maximum number of candidate locations a passenger can have. $\overline{\textbf{Obj. 1}}$ and $\overline{\textbf{Obj. 2}}$ represent the average objective values for Objectives 1 and 2 for each instance group, respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds. The ALNS heuristic was configured with $\theta = 3$ for the computational study conducted in Chapter 7

Instance	θ **= 1**			θ **= 3**			θ **= 6**			θ **= 9**		
Group	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$
S1	3.2	43.61	31.8	3.6	42.50	53.5	3.6	42.50	94.5	3.6	42.50	153.8
S2	5.2	70.07	50.0	5.6	66.42	109.7	5.6	66.42	118.9	5.6	66.42	148.8
S3	8.8	142.03	134.1	9.4	132.42	274.6	9.4	132.42	314.9	9.4	132.42	377.6
M1	17.4	293.82	622.6	18.8	265.96	1408.9	19.2	255.60	1700.3	19.2	252.08	2628.2
M2	26.6	416.52	1239.4	27.0	374.20	2990.2	28.2	384.77	3059.6	28.2	381.15	3472.7
M3	40.4	529.91	2019.1	41.4	507.91	3400.2	41.6	489.96	3562.7	41.6	486.93	3600.0
L1	58.2	665.35	3453.7	59.0	616.53	3600.0	59.0	599.23	3600.0	59.2	604.74	3600.0
L2	69.8	771.97	3600.0	71.8	721.01	3600.0	73.1	724.31	3600.0	73.5	724.90	3600.0
L3	97.6	1077.25	3600.0	99.0	1039.19	3600.0	99.4	1030.94	3600.0	99.5	1046.66	3600.0
Average	36.3	445.62	1639.3	37.3	418.46	2115.2	37.7	414.08	2184.6	37.8	415.25	2355.3

Chapter 9

Concluding Remarks

This thesis presents a potential solution to the unique transportation challenges faced by regions like Sotra and the greater Bergen area in Norway, namely, ridesharing. Recognizing the need for sustainable and efficient commuting alternatives, the thesis aimed to develop a method of coordinating passengers and drivers within a ridesharing system. With this focus, the Static Ridesharing Routing Problem with Flexible Locations (SRRPFL) was formulated, aiming to design optimized routes for a set of drivers to pick-up and deliver passengers at flexible locations.

Emphasizing the importance of individual travel needs, preferences, and constraints, the SRRPFL sought to maximize passenger participation and minimize total travel time for all drivers, creating an efficient transportation network that could alleviate traffic congestion and contribute to sustainability goals. The introduction of flexible locations in the SRRPFL model elevated the concept of ridesharing by adding a layer of convenience and customization to the system. This feature allowed passengers to choose from a set of candidate pick-up locations (and potentially also candidate delivery locations), increasing their participation by accommodating individual travel plans and constraints.

An arc-flow mixed integer programming model was developed to solve the SRRPFL using a commercial solver. However, since the commercial solver was only able to solve tiny instances of the problem, an adaptive large neighborhood search (ALNS) heuristic was proposed. This heuristic, grounded in the work of Ropke & Pisinger (2006), was adapted to fit the unique requirements of the SRRPFL. The ALNS heuristic presented in this thesis includes both the destroy and repair operators introduced by Ropke & Pisinger (2006), alongside new operators specifically devised for the SRRPFL. Furthermore, a local search (LS) component was integrated to enhance the solutions generated by the ALNS, resulting in frequent improvements of the global best solution. The ALNS heuristic also incorporates a set-partitioning problem, known as the route combination problem (RCP), which optimizes the utilization and combination of routes previously encountered in the search.

A real-world dataset, provided by Telia, was employed to accurately represent the transportation landscape of Sotra and the greater Bergen area. This dataset detailed travel patterns of residents in the region, including detailed information on trip origin and destination and the frequency of trips. Utilizing the data provided by Telia allowed us to incorporate specific geographic and demographic details. In total, 43 origin locations and 20 destination locations were considered, an extension of previous work by Nitter & Yang (2022). These locations were divided into four distinct zones, providing a rich test-

ing ground for the SRRPFL. This dataset formed the basis for generating the 45 unique performance test instances used to evaluate the ALNS heuristic. These test instances, representative of the diversity and complexity inherent in the transportation network, were methodically categorized based on their size into instance groups. A thorough analysis of the heuristic's performance on these instances revealed that the best results were achieved when both the local search and the RCP components were included in the ALNS configuration.

The results generated show that the ALNS heuristic finds solutions that demonstrates benefits of ridesharing. Using the ALNS heuristic with LS and the RCP, over 95% of potential passengers were picked up and delivered in larger instances, leading to a substantial reduction in the number of individual cars on the road. The possibility of reducing traffic congestion with ridesharing could have an important impact on the overall travel time and commuting experience. In addition, ridesharing has environmental benefits by reducing CO_2 emissions. By pooling passengers, the total distance driven is reduced, thus reducing the overall carbon footprint. This can contribute to environmental conservation efforts.

The study on the value of maximum travel time (T_k^M) revealed its crucial role in the ridesharing system. It shows that an increase in the maximum travel time multiplier essentially expands the potential ridesharing matches, enabling the accommodation of a larger number of passenger requests. The versatility of this parameter allows ridesharing systems to better cater to user preferences and tolerances, balancing convenience with the benefits of shared commuting. Despite the associated increase in overall travel time, the potential for enhancing overall system efficiency by serving a greater number of passengers underscores its importance.

Furthermore, candidate locations were found to be a valuable addition to the ridesharing model. They introduced greater flexibility and efficiency to the system. Incorporating candidate locations led to an increase in the average number of passengers picked up and a decrease in the average total travel time for drivers. While a higher θ value (representing more candidate locations) can lead to more passengers being picked up and potentially lower total travel time for drivers, it also leads to a notable increase in computational time. The insights highlight the crucial role that candidate locations play in enhancing the performance of such systems, indicating that ridesharing models that consider candidate locations could see more success in implementation.

The ALNS heuristic, along with the proposed extensions, demonstrates its potential by providing high-quality solutions for all generated test instances within the 3600 seconds time limit. Consequently, the heuristic shows its capability to address the SRRPFL for the actual case considered in this study. Therefore, it exhibits strong potential as a practical and efficient tool for the ridesharing challenges faced by the Sotra region and the greater Bergen area.

Chapter 10

Future Research

An evaluation of the work conducted on the Static Ridesharing Routing Problem with Flexible Locations (SRRPFL) reveals several promising areas for future investigation. This chapter is organized around three key areas of potential advancement. First, we explore potential extensions to the mathematical model. Second, we consider how the adaptive large neighborhood search (ALNS) heuristic might be improved. Finally, we explore alternative approaches to analyze and understand the behavior of actors within the ridesharing systems.

First, there are several interesting extensions to the SRRPFL. One potential area of extension is incorporating traffic and travel time uncertainties to reflect fluctuating traffic conditions better. In such scenarios, the optimization problem becomes stochastic, providing a more realistic modeling of real-world ridesharing operations. In addition to these, the concept of "flexible drivers" could be integrated into the model. In the SRRPFL, roles are defined a priori - users are either drivers or passengers. With flexible drivers, a new subset of users is introduced who own a car but are also open to being passengers in another car. This flexibility potentially introduces additional layers of solution spaces and opportunities for increased ridesharing efficiency. Furthermore, another compelling extension could address the issue of reverse commuting in ridesharing - from the workplace to the homes of the participants. Currently, the SRRPFL mainly focuses on ridesharing from home to the workplace. The addition of a reverse commute perspective would not only increase the scope of the model but also the realism, as it takes into account the complete daily commuting cycle.

The second opportunity for further research involves enhancing the efficiency of the adaptive large neighborhood search (ALNS) heuristic, particularly through preprocessing and the creation of "compatibility subsets". One identified improvement relates to the preliminary assignment of passengers to drivers. In the current approach, the heuristic does not preclude any passenger-driver pairings, even those that may be logistically infeasible due to spatial constraints or other factors. This lack of preprocessing means that the ALNS heuristic may spend considerable computational resources exploring unlikely or impossible assignments. In light of this, one proposed enhancement to the ALNS heuristic is to identify and eliminate these infeasible pairings. Passengers that are evidently out of a driver's reach could then be excluded from that driver's potential assignments. This "compatibility preprocessing" could result in the creation of compatibility subsets, where each subset contains only those passengers that a given driver could feasibly pick up. By excluding infeasible options in the initial phase, the ALNS heuristic would be streamlined and better able to focus on the exploration of viable solutions.

The third direction for future research lies in the exploration of alternative approaches to analyzing ridesharing systems, especially focusing on understanding and modeling the behavior of actors within the system. This research could involve choice models, a set of methods frequently used in transportation and logistics research to predict how individuals react to various incentives or policy measures. Choice models, including discrete choice models and stochastic choice models, capture individual decision-making processes under conditions of uncertainty. These models could be utilized to estimate how potential passengers and drivers might respond to different aspects of a ridesharing scheme. For instance, one could study how fare prices, waiting times, detour lengths, or environmental considerations might influence the choice of a passenger to participate in ridesharing. Similarly, drivers' willingness to share rides might be modeled considering factors such as compensation, detour lengths, or passenger compatibility. One of the main advantages of choice models is that they can take into account the heterogeneity of actors, acknowledging that different individuals may have different preferences and react differently to the same conditions. These models, thus, have a high potential to enrich our understanding of ridesharing systems and improve their design and operation. This kind of behavioral modeling could complement the analysis based on the SRRPFL, providing insights on how to make ridesharing schemes more attractive to potential users and increase their participation rate. It might also help in assessing the potential impacts of policy measures aimed at promoting ridesharing.

Bibliography

Agatz, N., Erera, A., Savelsbergh, M., & Wang, X. (2012). Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, *223*, 295–303. URL: https://www.sciencedirect.com/science/article/pii/S0377221712003864. doi:https://doi.org/10.1016/j.ejor.2012.05.028.

Auad-Perez, R., & Hentenryck, P. V. (2022). Ridesharing and fleet sizing for on-demand multimodal transit systems. *Transportation Research Part C: Emerging Technologies*, *138*, 103594. URL: https://www.sciencedirect.com/science/article/pii/S0968090X22000407. doi:https://doi.org/10.1016/j.trc.2022.103594.

Baldacci, R., Maniezzo, V., & Mingozzi, A. (2004). An exact method for the car pooling problem based on lagrangean column generation. *Operations Research*, *52*, 422–439. URL: http://www.jstor.org/stable/30036593.

Bruck, B. P., Incerti, V., Iori, M., & Vignoli, M. (2017). Minimizing co2 emissions in a practical daily carpooling problem. *Computers & Operations Research*, *81*, 40–50. URL: https://www.sciencedirect.com/science/article/pii/S030505481630301X. doi:https://doi.org/10.1016/j.cor.2016.12.003.

Cordeau, J.-F., & Laporte, G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, *37*, 579–594. URL: https://www.sciencedirect.com/science/article/pii/S0191261502000450. doi:https://doi.org/10.1016/S0191-2615(02)00045-0.

Dahle, L., Andersson, H., Christiansen, M., & Speranza, M. G. (2019). The pickup and delivery problem with time windows and occasional drivers. *Computers & Operations Research*, *109*, 122–133. URL: https://www.sciencedirect.com/science/article/pii/S0305054819301066. doi:https://doi.org/10.1016/j.cor.2019.04.023.

Fielbaum, A., Bai, X., & Alonso-Mora, J. (2021). On-demand ridesharing with optimized pick-up and drop-off walking locations. *Transportation Research Part C: Emerging Technologies*, *126*, 103061. URL: https://www.sciencedirect.com/science/article/pii/S0968090X21000887. doi:https://doi.org/10.1016/j.trc.2021.103061.

Ghandeharioun, Z., & Kouvelas, A. (2023). Real-time ridesharing operations for on-demand capacitated systems considering dynamic travel time information. *Transportation Research Part C: Emerging Technologies*, *151*, 104115. URL: https://www.sciencedirect.com/science/article/pii/S0968090X23001043. doi:https://doi.org/10.1016/j.trc.2023.104115.

He, P., Jin, J. G., Schulte, F., & Trépanier, M. (2023). Optimizing first-mile ridesharing services to intercity transit hubs. *Transportation Research Part C: Emerging Technologies*, *150*, 104082. URL: https://www.sciencedirect.com/science/article/pii/S0968090X23000712. doi:https://doi.org/10.1016/j.trc.2023.104082.

Ho, S. C., Szeto, W., Kuo, Y.-H., Leung, J. M., Petering, M., & Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, *111*, 395–421. URL: https://www.sciencedirect.com/science/article/pii/S0191261517304484. doi:https://doi.org/10.1016/j.trb.2018.02.001.

Hou, L., Li, D., & Zhang, D. (2018). Ride-matching and routing optimisation: Models and a large neighbourhood search heuristic. *Transportation Research Part E: Logistics and Transportation Review*, *118*, 143–162. URL: https://www.sciencedirect.com/science/article/pii/S1366554517310050. doi:https://doi.org/10.1016/j.tre.2018.07.003.

Hsieh, F.-S. (2020). A comparative study of several metaheuristic algorithms to optimize monetary incentive in ridesharing systems. *ISPRS International Journal of Geo-Information*, *9*. URL: https://www.mdpi.com/2220-9964/9/10/590.

Kaan, L., & Olinick, E. V. (2013). The vanpool assignment problem: Optimization models and solution algorithms. *Computers & Industrial Engineering*, *66*, 24–40. URL: https://www.sciencedirect.com/science/article/pii/S0360835213001824. doi:https://doi.org/10.1016/j.cie.2013.05.020.

Øygarden kommune (2023). Om oss. URL: https://www.oygarden.kommune.no/politikk-og-organisasjon/organisasjon/om-oss/.

Li, T., Xu, M., Sun, H., Xiong, J., & Dou, X. (2023). Stochastic ridesharing equilibrium problem with compensation optimization. *Transportation Research Part E: Logistics and Transportation Review*, *170*, 102999. URL: https://www.sciencedirect.com/science/article/pii/S1366554522003763. doi:https://doi.org/10.1016/j.tre.2022.102999.

Lin, Q., Xu, W., Chen, M., & Lin, X. (2019). A probabilistic approach for demand-aware ride-sharing optimization, . (p. 141–150). URL: https://doi.org/10.1145/3323679.3326512. doi:10.1145/3323679.3326512.

Liu, R., Tao, Y., & Xie, X. (2019). An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits. *Computers & Operations Research*, *101*, 250–262. URL: https://www.sciencedirect.com/science/article/pii/S030505481830220X. doi:https://doi.org/10.1016/j.cor.2018.08.002.

Lu, Q., & Dessouky, M. (2004). An exact algorithm for the multiple vehicle pickup and delivery problem. *Transportation Science*, *38*, 503–514. URL: http://www.jstor.org/stable/25769222.

Lu, Q., & Dessouky, M. M. (2006). A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, *175*, 672–687. URL: https://www.sciencedirect.com/science/article/pii/S0377221705004698. doi:https://doi.org/10.1016/j.ejor.2005.05.012.

Madsen, O. B. G., Ravn, H. F., & Rygaard, J. M. (1995). A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. URL: https://link.springer.com/article/10.1007/BF02031946#citeas. doi:https://doi.org/10.1007/BF02031946.

NAF (2015). Kun 600 biler ekstra skaper køen på E18.

Nitter, J., & Yang, S. (2022). Reducing traffic congestion through ridesharing: Case of sotra. project thesis, norwegian university of science and technology (ntnu).

Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2008). A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, *58*, 81–117. URL: https://doi.org/10.1007%2Fs11301-008-0036-4. doi:10.1007/s11301-008-0036-4.

Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2010). Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, *37*, 1129–1138. URL: https://www.sciencedirect.com/science/article/pii/S030505480900241X. doi:https://doi.org/10.1016/j.cor.2009.10.003.

Pelzer, D., Xiao, J., Zehe, D., Lees, M. H., Knoll, A. C., & Aydt, H. (2015). A partition-based match making algorithm for dynamic ridesharing. *IEEE Transactions on Intelligent Transportation Systems*, *16*, 2587–2598. doi:10.1109/TITS.2015.2413453.

Psaraftis, H. N. (1980). A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, *14*, 130–154. URL: http://www.jstor.org/stable/25767975.

Ropke, S., Cordeau, J. F., & Laporte, G. (2006). Models and branch-and-cut algorithms for pickup and delivery problems with time windows.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, *40*, 455–472. URL: http://www.jstor.org/stable/25769321.

Smet, P. (2021). Ride sharing with flexible participants: a metaheuristic approach for large-scale problems. *Int. Trans. Oper. Res.*, *28*, 91–118.

Statens Vegvesen (2023). Sotrabrua. URL: https://www.vegvesen.no/trafikkdata/start/utforsk?datatype=volume&display=chart&fbclid=IwAR1Cpdd0T71mkRUlm8zvsu1vtiYyuPGi8HWmDr0FYabGej6jdeakWqxHScY&from=2023-05-30#trpids=29614V805708.

Stiglic, M., Agatz, N., Savelsbergh, M., & Gradisar, M. (2015). The benefits of meeting points in ride-sharing systems. *Transportation Research Part B: Methodological*, *82*, 36–53. URL: https://www.sciencedirect.com/science/article/pii/S0191261515002088. doi:https://doi.org/10.1016/j.trb.2015.07.025.

Sun, Y., Chen, Z.-L., & Zhang, L. (2020). Nonprofit peer-to-peer ridesharing optimization. *Transportation Research Part E: Logistics and Transportation Review*, *142*, 102053. URL: https://www.sciencedirect.com/science/article/pii/S1366554520307043. doi:https://doi.org/10.1016/j.tre.2020.102053.

Telia (2022). Trafikkdata Øygården bergen uke 39-43 2022, crowd insights (tom henriksen), . URL: https://www.sciencedirect.com/science/article/pii/S0968090X21000887.

Turkeš, R., Sörensen, K., & Hvattum, L. M. (2021). Meta-analysis of metaheuristics: Quantifying the effect of adaptiveness in adaptive large neighborhood search. *European Journal of Operational Research*, *292*, 423–442. URL: https://www.sciencedirect.com/science/article/pii/S037722172030936X. doi:https://doi.org/10.1016/j.ejor.2020.10.045.

Wang, J., Sun, Y., Zhang, Z., & Gao, S. (2020). Solving multitrip pickup and delivery problem with time windows and manpower planning using multiobjective algorithms. *IEEE/CAA Journal of Automatica Sinica*, *7*, 1134–1153. doi:10.1109/JAS.2020.1003204.

Zheng, M., & Pantuso, G. (2023). Trading off costs and service rates in a first-mile ride-sharing service. *Transportation Research Part C: Emerging Technologies*, *150*, 104099. URL: https://www.sciencedirect.com/science/article/pii/S0968090X23000888. doi:https://doi.org/10.1016/j.trc.2023.104099.

Appendix A

Mathematical Model

Sets

$\mathcal{D}$	-	set of drivers k
$\mathcal{P}^P$	-	set of passenger origin locations i
$\mathcal{P}^D$	-	set of passenger destination locations j
$\mathcal{M}_i^P$	-	set of candidate pick-up locations for passenger i
$\mathcal{M}_i^D$	-	set of candidate delivery locations for passenger i
$\mathcal{N}^P$	-	set of pick-up nodes
$\mathcal{N}^D$	-	set of delivery nodes
$\mathcal{N}^R$	-	set of ridesharing nodes
$\mathcal{N}$	-	set of all nodes
$\mathcal{A}_k$	-	set of arcs driver k can traverse

Parameters

$o(k)$	-	origin node for driver k
$d(k)$	-	destination node for driver k
T^D_{imjn}	-	direct travel time from node $(i,m) \in \mathcal{N}^R \cup \{o(k)\}$ to node $(j,n) \in \mathcal{N}^R \cup \{d(k)\}$
T^C_{im}	-	direct travel time between origin/destination location for passenger $i \in \mathcal{P}^P$ and its candidate pick up/delivery location $(i,m) \in \mathcal{N}^P \cup \mathcal{N}^D$
T^M_k	-	maximum travel time for driver/passenger $k \in \mathcal{D} \cup \mathcal{P}^P$
$\underline{A}_k$	-	earliest arrival time at destination for driver/passenger $k \in \mathcal{D} \cup \mathcal{P}^P$
$\overline{A}_k$	-	latest arrival time at destination for driver/passenger $k \in \mathcal{D} \cup \mathcal{P}^P$
Q_k	-	maximum capacity for driver k

Decision Variables

x^S_{kim}	-	1 if driver $k \in \mathcal{D}$ travels from its origin location $o(k)$ to a pick up node $(i,m) \in \mathcal{N}^P$, 0 otherwise
x_{kimjn}	-	1 if driver $k \in \mathcal{D}$ travels directly between ridesharing nodes $(i,m) \in \mathcal{N}^R$ and $(j,n) \in \mathcal{N}^R$, 0 otherwise
x^E_{kjn}	-	1 if driver $k \in \mathcal{D}$ travels from a delivery node $(j,n) \in \mathcal{N}^D$ to its destination location $d(k)$, 0 otherwise
x^{OD}_k	-	1 if driver $k \in \mathcal{D}$ travels directly from its origin location $o(k)$ to its destination location $d(k)$, 0 otherwise
y_{kim}	-	1 if driver $k \in \mathcal{D}$ picks up/delivers passenger $i \in \mathcal{P}^P$ at node $(i,m) \in \mathcal{N}^P \cup \mathcal{N}^D$, 0 otherwise
z_{ki}	-	1 if driver $k \in \mathcal{D}$ picks up passenger $i \in \mathcal{P}^P$, 0 otherwise
t_{kim}	-	The time driver/passenger $k \in \mathcal{D} \cup \mathcal{P}^P$ leaves node $(i,m) \in \mathcal{N}$

Objective Functions

$$\max\ z_1 = \sum_{k \in \mathcal{D}} \sum_{i \in \mathcal{P}^P} z_{ki} \tag{A.1}$$

$$\min\ z_2 = \sum_{k \in \mathcal{D}} (t_{k,d(k)} - t_{k,o(k)}) \tag{A.2}$$

Constraints

$$\sum_{(i,m) \in \mathcal{N}^P} x^S_{kim} + x^{OD}_k = 1, \quad k \in \mathcal{D} \tag{A.3}$$

$$\sum_{(j,n) \in \mathcal{N}^D} x^E_{kjn} + x^{OD}_k = 1, \quad k \in \mathcal{D} \tag{A.4}$$

$$x^S_{kim} + \sum_{(j,n) \in \mathcal{N}^P} x_{kjnim} = \sum_{(j,n) \in \mathcal{N}^R} x_{kimjn}, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^P \tag{A.5}$$

$$x^E_{kjn} + \sum_{(i,m) \in \mathcal{N}^D} x_{kjnim} = \sum_{(i,m) \in \mathcal{N}^R} x_{kimjn}, \quad k \in \mathcal{D}, (j,n) \in \mathcal{N}^D \tag{A.6}$$

$$x^S_{kjn} + \sum_{(i,m) \in \mathcal{N}^P} x_{kimjn} - y_{kjn} = 0, \quad k \in \mathcal{D}, (j,n) \in \mathcal{N}^P \tag{A.7}$$

$$x^E_{kjn} + \sum_{(i,m) \in \mathcal{N}^D} x_{kjnim} - y_{kjn} = 0, \quad k \in \mathcal{D}, (j,n) \in \mathcal{N}^D \tag{A.8}$$

$$\sum_{k\in\mathcal{D}}\sum_{m\in\mathcal{M}_i^P\cup\mathcal{M}_i^D} y_{kim} \le 1, \quad i\in\mathcal{P}^P \tag{A.9}$$

$$z_{ki} = \sum_{m\in\mathcal{M}_i^P} y_{kim}, \quad k\in\mathcal{D}, i\in\mathcal{P}^P \tag{A.10}$$

$$\sum_{m\in\mathcal{M}_i^P} y_{kim} = \sum_{n\in\mathcal{M}_i^D} y_{k,N+i,n}, \quad k\in\mathcal{D}, i\in\mathcal{P}^P \tag{A.11}$$

$$t_{kim} + T^D_{i,m,N+i,n} z_{ki} - t_{k,N+i,n} \le 0, \quad k\in\mathcal{D}, (i,m)\in\mathcal{N}^P, n\in\mathcal{M}_i^D \tag{A.12}$$

$$t_{kim} + T^D_{imjn} - t_{kjn} - M_{kimjn}(1-x_{kimjn}) \le 0, \quad k\in\mathcal{D}, (i,m)\in\mathcal{N}^R, (j,n)\in\mathcal{N}^R \tag{A.13}$$

$$t_{kim} + T^D_{imjn} - t_{kjn} + M_{kimjn}(1-x_{kimjn}) \ge 0, \quad k\in\mathcal{D}, (i,m)\in\mathcal{N}^R, (j,n)\in\mathcal{N}^R \tag{A.14}$$

$$t_{k,o(k)} + T^D_{o(k),i,m} - t_{kim} - M_{k,o(k),i,m}(1-x^S_{kim}) \le 0, \quad k\in\mathcal{D}, (i,m)\in\mathcal{N}^P \tag{A.15}$$

$$t_{k,o(k)} + T^D_{o(k),i,m} - t_{kim} + M_{k,o(k),i,m}(1-x^S_{kim}) \ge 0, \quad k\in\mathcal{D}, (i,m)\in\mathcal{N}^P \tag{A.16}$$

$$t_{kim} + T^D_{i,m,d(k)} - t_{k,d(k)} - M_{k,i,m,d(k)}(1-x^E_{kim}) \le 0, \quad k\in\mathcal{D}, (i,m)\in\mathcal{N}^D \tag{A.17}$$

$$t_{kim} + T^D_{i,m,d(k)} - t_{k,d(k)} + M_{k,i,m,d(k)}(1-x^E_{kim}) \ge 0, \quad k\in\mathcal{D}, (i,m)\in\mathcal{N}^D \tag{A.18}$$

$$t_{k,o(k)} + T^D_{o(k),d(k)} - t_{k,d(k)} - M_{k,o(k),d(k)}(1-x^{OD}_k) \le 0, \quad k\in\mathcal{D} \tag{A.19}$$

$$t_{k,o(k)} + T^D_{o(k),d(k)} - t_{k,d(k)} + M_{k,o(k),d(k)}(1-x^{OD}_k) \ge 0, \quad k\in\mathcal{D} \tag{A.20}$$

$$\underline{A}_k \le t_{k,d(k)} \le \overline{A}_k, \quad k\in\mathcal{D} \tag{A.21}$$

$$\underline{A}_i z_{ki} \le t_{k,N+i,0} + T^C_{N+i,n} y_{kim} \le \overline{A}_i z_{ki}, \quad k\in\mathcal{D}, i\in\mathcal{P}^P, n\in\mathcal{M}_i^D \tag{A.22}$$

$$t_{k,d(k)} - t_{k,o(k)} \le T^M_k, \quad k\in\mathcal{D} \tag{A.23}$$

$$t_{k,N+i,0} - t_{k,i,0} \le T^M_i + M_{k,i,0}(1-z_{ki}), \quad k\in\mathcal{D}, i\in\mathcal{P}^P \tag{A.24}$$

$$t_{k,i,0} \leq t_{kim} - T^C_{im} y_{kim}, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^P \tag{A.25}$$

$$t_{k,j,0} \geq t_{kjn} + T^C_{jn} y_{kjn}, \quad k \in \mathcal{D}, (j,n) \in \mathcal{N}^D \tag{A.26}$$

$$\sum_{i \in \mathcal{P}^P} z_{ki} \leq Q_k, \quad k \in \mathcal{D} \tag{A.27}$$

$$x^S_{kim} \in \{0,1\}, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^P \tag{A.28}$$

$$x_{kimjn} \in \{0,1\}, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^R, (j,n) \in \mathcal{N}^R \tag{A.29}$$

$$x^E_{kjn} \in \{0,1\}, \quad k \in \mathcal{D}, (j,n) \in \mathcal{N}^D \tag{A.30}$$

$$x^{OD}_k \in \{0,1\}, \quad k \in \mathcal{D} \tag{A.31}$$

$$y_{kim} \in \{0,1\}, \quad k \in \mathcal{D}, (i,m) \in \mathcal{N}^P \cup \mathcal{N}^D \tag{A.32}$$

$$z_{ki} \in \{0,1\}, \quad k \in \mathcal{D}, i \in \mathcal{P}^P \tag{A.33}$$

$$t_{kim} \geq 0, \quad k \in \mathcal{D} \cup \mathcal{P}^P, (i,m) \in \mathcal{N} \tag{A.34}$$

Appendix B

Zones and Location Names

B.1 Zone 1

Figure B.1 illustrates the origin locations in Zone 1, where each location is represented by a label such as *"O1"* for origin location 1. The corresponding place or location names for these origin locations are listed in Table B.1.

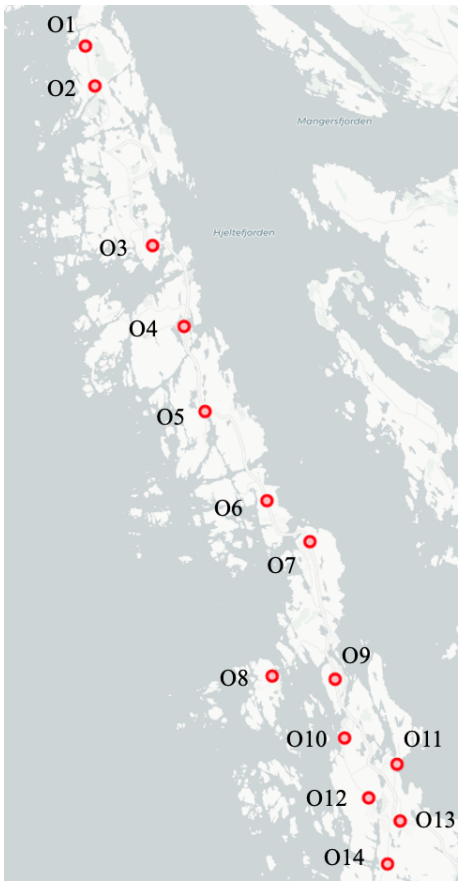


Figure B.1: Map of Zone 1 in the Sotra region, with origin locations marked by red markers and labeled as O1, O2, etc.

Table B.1: Corresponding location names for origin locations in Zone 1 (In Norwegian)

Origin location	Location name	Origin location	Location name
O1	Hellesøy	O11	Vindesnes
O2	Seløyna	O12	Ongeltveit
O3	Tjeldstø	O13	Ågotnes
O4	Oøy	O14	Kårtveit
O5	Blomvåg		
O6	Rongøyna		
O7	Torsteinsvik		
O8	Turøyna		
O9	Misje		
O10	Sollsvika		

B.2 Zone 2

Figure B.2 illustrates the origin locations in Zone 2. The corresponding place or location names for these origin locations are listed in Table B.2.

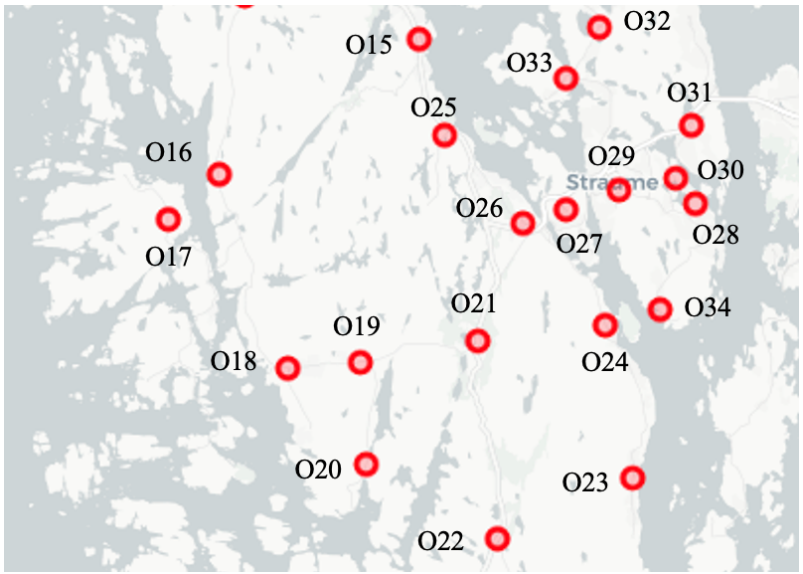


Figure B.2: Map of Zone 2 in the Sotra region, with origin locations marked by red markers and labeled as O15, O16, etc.

Table B.2: Corresponding location names for origin locations in Zone 2 (In Norwegian)

Origin location	Location name	Origin location	Location name
O15	Knappskog	O25	Morland
O16	Sekkingstad	O26	Kolltveit
O17	Algerøyna	O27	Bildøy
O18	Møvik	O28	Brattholmen
O19	Ulveset	O29	Straume
O20	Nessjøen	O30	Arefjord
O21	Fjell	O31	Knarrevik
O22	Tellnes	O32	Vågen
O23	Liaskjeret	O33	Foldnes
O24	Hovden	O34	Ebbesvik

B.3 Zone 3

Figure B.3 illustrates the origin locations in Zone 3. The corresponding place or location names for these origin locations are listed in Table B.3.

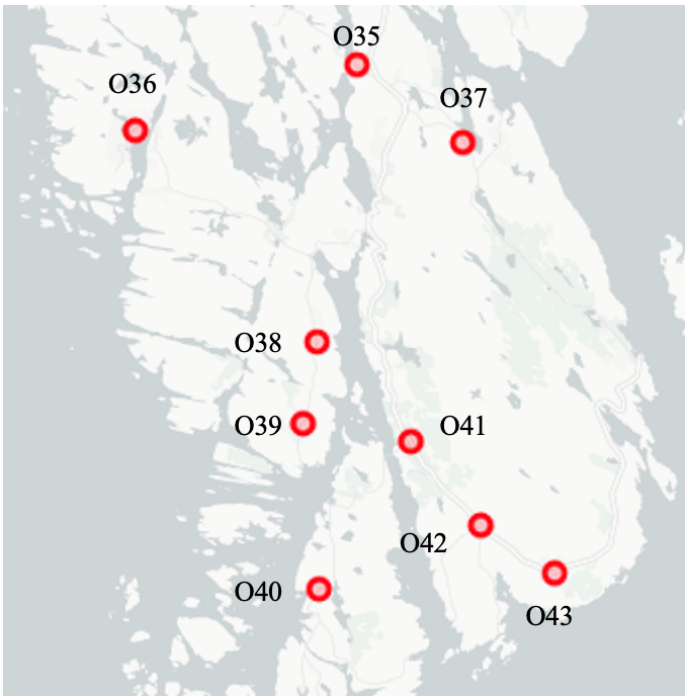


Figure B.3: Map of Zone 3 in the Sotra region, with origin locations marked by red markers and labeled as O35, O36, etc.

Table B.3: Corresponding location names for origin locations in Zone 3 (In Norwegian)

Origin location	Location name
O35	Trengereid
O36	Telavåg
O37	Skogsvåg
O38	Kausland
O39	Glesvær
O40	Trælevika
O41	Berge
O42	Forland
O43	Fardalen

B.4 Zone 4

Figure B.4 illustrates the destination locations in Zone 4, where each location is represented by a label such as *"D1"* for destination location 1. The corresponding place or location names for these destination locations are listed in Table B.4.

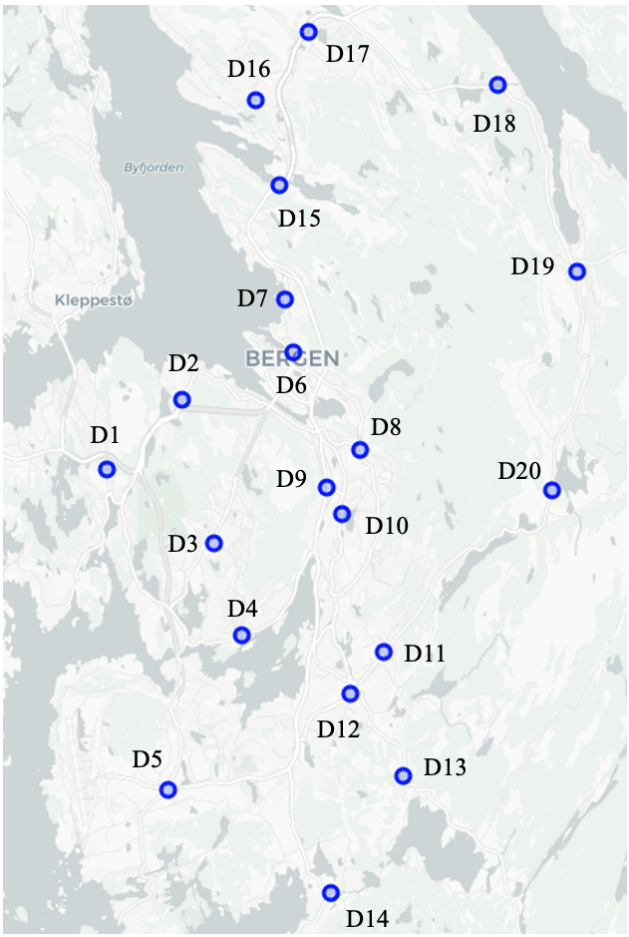


Figure B.4: Map of Zone 4 in the Bergen region, with destination locations marked by blue markers and labeled as D1, D2, etc.

Table B.4: Corresponding location names for destination locations in Zone 4 (In Norwegian)

Destination location	**Location name**	**Destination location**	**Location name**
D1	Loddefjord	D11	Landås-sædalen
D2	Laksevåg	D12	Nesttun
D3	Fyllingsdalen	D13	Kaland
D4	Bønes	D14	Fana
D5	Ytrebygda	D15	Eidsvåg
D6	Bergen sentrum	D16	Tertnes-salhus
D7	Sandviken	D17	Åsane
D8	Lone	D18	Ytre arna
D9	Solheim	D19	Indre arna
D10	Årstad	D20	Fridalen-slettebakken

Appendix C

Origin and Destination Location Coordinates

C.1 Zone 1 Coordinates

Table C.1: Coordinates of Origin Locations in Zone 1

Location	Latitude	Longitude	Location	Latitude	Longitude
O1: Hellesøy	60.653579	4.796694	O11: Vindesnes	60.422644	5.000645
O2: Seløyna	60.640977	4.803140	O12: Ongeltveit	60.411782	4.982625
O3: Tjeldstø	60.589538	4.840584	O13: Ågotnes	60.404263	5.002809
O4: Oøy	60.563607	4.861620	O14: Kårtveit	60.390493	4.994399
O5: Blomvåg	60.536478	4.874959			
O6: Rongøyna	60.507616	4.915516			
O7: Torsteinsvik	60.494483	4.943608			
O8: Turøyna	60.451152	4.919396			
O9: Misje	60.450161	4.960451			
O10: Sollsvika	60.431002	4.966409			

C.2 Zone 2 Coordinates

Table C.2: Coordinates of Origin Locations in Zone 2

Location	Latitude	Longitude	Location	Latitude	Longitude
O15: Knappskog	60.382878	5.055929	O25: Morland	60.366201	5.064647
O16: Sekkingstad	60.359700	4.985472	O26: Kolltveit	60.351225	5.092105
O17: Algerøyna	60.351787	4.967791	O27: Bildøy	60.353390	5.107026
O18: Møvik	60.326007	5.009692	O28: Brattholmen	60.354616	5.152274
O19: Ulveset	60.327046	5.035510	O29: Straume	60.356937	5.125721
O20: Nessjøen	60.309361	5.037209	O30: Arefjord	60.358846	5.145409
O21: Fjell	60.330735	5.076383	O31: Knarrevik	60.368049	5.151220
O22: Tellnes	60.296214	5.083010	O32: Vågen	60.385109	5.119060
O23: Liaskjeret	60.306928	5.130726	O33: Foldnes	60.376362	5.107050
O24: Hovden	60.333400	5.121158	O34: Ebbesvik	60.336068	5.140128

C.3 Zone 3 Coordinates

Table C.3: Coordinates of Origin Locations in Zone 3

Location	Latitude	Longitude
O35: Trengereid	60.273042	5.060527
O36: Telavåg	60.262097	4.984143
O37: Skogsvåg	60.25994	5.097186
O38: Kausland	60.225919	5.046943
O39: Glesvær	60.211953	5.041804
O40: Trælevika	60.183544	5.047290
O41: Berge	60.208794	5.078852
O42: Forland	60.194402	5.103127
O43: Fardalen	60.186358	5.128284

C.4 Zone 4 Coordinates

Table C.4: Coordinates of Destination Locations in Zone 4

Location	Latitude	Longitude	Location	Latitude	Longitude
D1: Loddefjord	60.366936	5.234154	D11: Landås-sædalen	60.322493	5.370700
D2: Laksevåg	60.359700	5.271280	D12: Nesttun	60.312429	5.354703
D3: Fyllingsdalen	60.348946	5.287026	D13: Kaland	60.292324	5.380563
D4: Bønes	60.326543	5.301184	D14: Fana	60.263706	5.344979
D5: Ytrebygda	60.288845	5.264713	D15: Eidsvåg	60.436380	5.319374
D6: Bergen sentrum	60.395409	5.326163	D16: Tertnes-salhus	60.456868	5.307458
D7: Sandviken	60.408271	5.322390	D17: Åsane	60.473474	5.333811
D8: Lone	60.371778	5.359552	D18: Ytre arna	60.460644	5.427381
D9: Solheim	60.362707	5.342976	D19: Indre arna	60.415280	5.466286
D10: Årstad	60.356113	5.350348	D20: Fridalen-slettebakken	60.361881	5.454060

Appendix D

Test Instances

Table D.1 presents a list of all performance instances in this thesis.

Table D.1: Summary of all instances

Instance ID	Instance Group	Drivers	Passengers
S1-1D-4P-1	S1	1	4
S1-1D-4P-2	S1	1	4
S1-1D-4P-3	S1	1	4
S1-1D-4P-4	S1	1	4
S1-1D-4P-5	S1	1	4
S2-2D-6P-1	S2	2	6
S2-2D-6P-2	S2	2	6
S2-2D-6P-3	S2	2	6
S2-2D-6P-4	S2	2	6
S2-2D-6P-5	S2	2	6
S3-4D-10P-1	S3	4	10
S3-4D-10P-2	S3	4	10
S3-4D-10P-3	S3	4	10
S3-4D-10P-4	S3	4	10
S3-4D-10P-5	S3	4	10
M1-8D-20P-1	M1	8	20
M1-8D-20P-2	M1	8	20
M1-8D-20P-3	M1	8	20
M1-8D-20P-4	M1	8	20
M1-8D-20P-5	M1	8	20
M2-12D-30P-1	M2	12	30
M2-12D-30P-2	M2	12	30
M2-12D-30P-3	M2	12	30
M2-12D-30P-4	M2	12	30
M2-12D-30P-5	M2	12	30
M3-16D-42P-1	M3	16	42
M3-16D-42P-2	M3	16	42
M3-16D-42P-3	M3	16	42
M3-16D-42P-4	M3	16	42
M3-16D-42P-5	M3	16	42
L1-20D-60P-1	L1	20	60
L1-20D-60P-2	L1	20	60
L1-20D-60P-3	L1	20	60
L1-20D-60P-4	L1	20	60
L1-20D-60P-5	L1	20	60
L2-25D-75P-1	L2	25	75
L2-25D-75P-2	L2	25	75
L2-25D-75P-3	L2	25	75
L2-25D-75P-4	L2	25	75
L2-25D-75P-5	L2	25	75
L3-35D-100P-1	L3	35	100
L3-35D-100P-2	L3	35	100
L3-35D-100P-3	L3	35	100
L3-35D-100P-4	L3	35	100
L3-35D-100P-5	L3	35	100

Appendix E

Parameter Tuning

E.1 Percentage Removals of Passengers Parameter (γ)

Modifying the γ parameter influences how many passengers are destroyed in each iteration of the ALNS heuristic. The tested γ parameter include: $\gamma \in [5\%, 15\%]$, $\gamma \in [10\%, 20\%]$, $\gamma \in [15\%, 25\%]$, $\gamma \in [20\%, 35\%]$, and $\gamma \in [35\%, 50\%]$. The results of this tuning process are displayed in Table E.1.

The objective values and times presented in Table E.3 are the averages obtained from running each instance five times under each setting. The number of ALNS iterations (I^{ALNS}) was fixed at 5000. The results showed a high degree of similarity in both the average objective values. However, the runtimes increases significantly across the different settings. Therefore, the chosen value for the γ parameter is $\gamma \in [5\%, 15\%]$.

Table E.1: Results from tuning the γ parameter. Each instance was run five times for each setting of γ. $\overline{\textbf{Obj. 1}}$ and $\overline{\textbf{Obj. 2}}$ represent the average objective values for Objectives 1 and 2 for each instance group, respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds

Instances	$\gamma \in [5\%, 15\%]$			$\gamma \in [10\%, 20\%]$			$\gamma \in [15\%, 25\%]$			$\gamma \in [20\%, 35\%]$			$\gamma \in [35\%, 50\%]$		
	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$
1-5D-18P	10.0	123.37	224.6	10.0	123.37	305.3	10.0	123.40	310.0	10.0	123.37	340.3	10.0	123.40	675.4
2-6D-20P	18.0	248.10	334.7	18.0	248.10	460.5	18.0	248.10	440.0	18.0	248.10	523.1	18.0	248.10	805.7
3-8D-25P	23.0	344.53	579.3	23.0	345.23	713.2	23.0	344.20	826.4	23.0	344.53	903.5	23.0	346.27	1497.8
4-9D-28P	26.0	326.90	639.7	26.0	326.90	927.7	26.0	326.90	1023.4	26.0	326.90	1150.6	26.0	326.90	1605.7
5-10D-30P	30.0	355.10	500.5	30.0	354.53	984.0	30.0	355.03	1226.7	30.0	355.10	1123.7	30.0	354.57	1704.6
6-12D-36P	30.0	384.20	1084.7	30.0	384.20	1585.4	30.0	384.20	1550.5	30.0	384.20	1704.3	30.0	384.90	2534.3
7-13D-41P	37.0	396.90	1387.1	37.0	396.90	1702.3	37.0	397.23	1853.3	37.0	396.90	1905.7	37.0	396.90	2805.3
8-13D-46P	37.0	456.70	1354.8	37.0	456.70	1734.2	37.0	456.70	2213.3	37.0	456.70	2296.3	37.0	456.70	3304.2
Average	26.4	329.48	763.1	26.4	329.49	1051.6	26.4	329.47	1180.4	26.4	329.48	1243.4	26.4	329.72	1866.6

E.2 Adaptive Weight Score Parameters (σ)

Adjusting the adaptive weights used in the ALNS heuristic is done through the use of score parameters. σ_1 is the reward for finding a new global best solution. σ_2 is the reward for finding a new candidate solution that is better than the current solution, but not better than the best global solution found so far. σ_3 is the reward for finding a new candidate solution that is worse than the current solution, but is accepted by the simulated annealing acceptance criterion. σ_4 is the penalty for finding a new candidate solution that is worse than the current solution, and is rejected by the simulated annealing acceptance criterion. These are collectively denoted by $\sigma = [\sigma_1, \sigma_2, \sigma_3, \sigma_4]$. The score parameters tested include: $\sigma = [100, 30, 10, -5]$, $\sigma = [90, 50, 20, -20]$, $\sigma = [80, 20, 10, -5]$, $\sigma = [50, 10, 5, -2]$, and $\sigma = [30, 5, 3, 0]$. The results of this tuning process are presented in Table E.2.

The objective values and times presented in Table E.2 are the averages obtained from running each instance five times under each setting. The number of ALNS iterations (I^{ALNS}) was fixed at 5000. The results showed a high degree of similarity in both the average objective values and runtimes across different settings. However, the parameter set $\sigma = [100, 30, 10, -5]$ yielded the best overall average values for both Objective 1 and Objective 2. As such, $\sigma = [100, 30, 10, -5]$ was chosen, owing to its superior performance in terms of average objective values.

Table E.2: Results from tuning the σ parameters. Each instance was run five times for each setting of σ. $\overline{\textbf{Obj. 1}}$ and $\overline{\textbf{Obj. 2}}$ represent the average objective values for Objectives 1 and 2 for each instance group, respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds

Instances	$\sigma = [100, 30, 10, -5]$			$\sigma = [90, 50, 20, -20]$			$\sigma = [80, 20, 10, -5]$			$\sigma = [50, 10, 5, -2]$			$\sigma = [30, 5, 3, 0]$		
	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$
1-5D-18P	10.0	123.37	224.6	10.0	123.40	215.3	10.0	123.40	262.6	9.7	118.93	207.0	10.0	123.40	214.4
2-6D-20P	18.0	248.10	334.7	18.0	250.30	334.5	18.0	248.10	396.1	18.0	248.10	386.2	18.0	248.10	315.3
3-8D-25P	23.0	344.53	579.3	23.0	346.50	453.0	23.0	346.00	633.0	23.0	346.00	504.7	23.0	356.27	465.7
4-9D-28P	26.0	326.90	639.7	26.0	326.90	644.3	26.0	326.90	756.1	26.0	326.90	595.1	26.0	326.90	670.8
5-10D-30P	30.0	355.10	500.5	30.0	359.10	454.9	30.0	354.77	671.1	30.0	357.03	560.5	30.0	354.57	601.7
6-12D-36P	30.0	384.20	1084.7	30.0	387.30	977.7	30.0	385.60	1276.3	30.0	384.20	1077.7	30.0	384.90	1127.7
7-13D-41P	37.0	387.90	1387.1	37.0	396.90	1086.2	37.0	396.90	1364.2	37.0	396.90	1148.8	37.0	396.90	1256.0
8-13D-46P	37.0	456.70	1354.8	37.0	456.70	1259.8	37.0	456.87	1737.5	37.0	441.10	1414.0	37.0	456.70	1634.9
Average	26.4	328.35	763.1	26.4	330.89	678.2	26.4	329.82	885.8	26.3	327.40	727.8	26.4	329.72	785.3

E.3 Percentage Factor (δ)

Modifying the δ parameter influences how the ALNS heuristic determines whether a candidate solution is promising. The tested δ parameters include: $\delta = 80\%$, $\delta = 85\%$, $\delta = 90\%$, and $\delta = 95\%$. The results of this tuning process are displayed in Table E.3.

The objective values and times presented in Table E.3 are the averages obtained from running each instance five times under each setting. The number of ALNS iterations (I^{ALNS}) was fixed at 5000. The results showed a high degree of similarity in both the average objective values and runtimes across different settings. However, the parameter set $\delta = 90\%$ yielded the best overall average values for both Objective 1 and Objective 2. As such, $\delta = 90\%$ was chosen, owing to its superior performance in terms of average objective values.

Table E.3: Results from tuning the δ parameter. Each instance was run five times for each setting of δ. $\overline{\textbf{Obj. 1}}$ and $\overline{\textbf{Obj. 2}}$ represent the average objective values for Objectives 1 and 2 for each instance group, respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds

Instances	$\delta = 80\%$			$\delta = 85\%$			$\delta = 90\%$			$\delta = 95\%$		
	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$
1-5D-18P	10.0	123.40	235.7	10.0	123.40	213.1	10.0	123.40	224.6	10.0	123.40	272.0
2-6D-20P	18.0	252.50	296.7	18.0	250.30	363.1	18.0	248.10	334.7	18.0	248.10	332.9
3-8D-25P	23.0	345.13	545.3	23.0	345.13	605.1	23.0	344.53	579.3	23.0	344.67	486.5
4-9D-28P	26.0	326.90	579.0	26.0	326.90	589.5	26.0	326.90	639.7	26.0	327.20	591.9
5-10D-30P	30.0	354.83	600.5	30.0	354.83	534.0	30.0	355.10	500.5	30.0	355.10	533.6
6-12D-36P	30.0	388.30	1080.5	30.0	384.90	1120.7	30.0	384.20	1084.7	30.0	384.90	1108.2
7-13D-41P	37.0	397.57	1480.5	37.0	396.90	1210.6	37.0	396.90	1387.1	37.0	397.23	1225.4
8-13D-46P	37.0	456.70	1871.8	37.0	456.70	1393.7	37.0	456.70	1354.8	37.0	456.70	1326.6
Average	26.4	330.67	836.24	26.4	329.88	753.73	26.4	329.48	763.18	26.4	329.66	734.64

E.4 Reaction Factor (r)

Modifying the reaction factor r parameter influences how the ALNS heuristic updates its adaptive weights. The tested r parameters include: $r = 0.10$, $r = 0.20$, $r = 0.30$, $r = 0.50$ and $r = 1.00$. The results of this tuning process are displayed in Table E.4.

The objective values and times presented in Table E.4 are the averages obtained from running each instance five times under each setting. The number of ALNS iterations (I^{ALNS}) was fixed at 5000. The results showed a high degree of similarity in both the average objective values and runtimes across different settings. However, the parameter set $r = 0.10$ yielded the best overall average values for both Objective 1 and Objective 2. As such, $r = 0.10$ was chosen, owing to its superior performance in terms of average objective values.

Table E.4: Results from tuning the r parameter. Each instance was run five times for each setting of r. $\overline{\textbf{Obj. 1}}$ and $\overline{\textbf{Obj. 2}}$ represent the average objective values for Objectives 1 and 2 for each instance group, respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds

Instances	$r = 0.10$			$r = 0.20$			$r = 0.30$			$r = 0.50$			$r = 1.00$		
	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	$\overline{\textbf{Time [s]}}$
1-5D-18P	10.0	123.40	218.3	10.0	123.40	209.7	9.7	118.97	206.5	9.7	118.93	209.7	10.0	123.40	219.9
2-6D-20P	18.0	248.10	318.0	18.0	248.10	312.2	18.0	248.10	330.4	18.0	248.33	332.5	18.0	251.47	331.4
3-8D-25P	23.0	344.77	567.0	23.0	345.57	480.7	23.0	344.23	650.1	23.0	344.67	551.5	23.0	344.67	545.3
4-9D-28P	26.0	326.90	639.7	26.0	326.90	665.5	26.0	326.90	704.1	26.0	326.90	612.7	26.0	326.90	575.0
5-10D-30P	30.0	355.10	500.5	30.0	355.73	477.6	30.0	355.10	597.5	30.0	354.83	542.9	30.0	354.83	567.9
6-12D-36P	30.0	384.20	1084.7	30.0	384.20	1190.5	30.0	384.90	1018.0	30.0	384.20	1109.4	30.0	386.60	960.0
7-13D-41P	37.0	387.90	1387.1	37.0	396.90	1096.3	37.0	396.90	1115.3	37.0	396.90	1092.4	37.0	396.90	1233.5
8-13D-46P	37.0	456.70	1354.8	37.0	456.70	1305.7	37.0	456.70	1715.9	37.0	456.70	1308.7	37.0	456.70	1389.4
Average	26.4	329.51	758.77	26.4	329.70	717.26	26.3	328.98	792.22	26.3	328.93	719.97	26.4	330.18	727.80

Appendix F

Adaptive Weights Development

In this appendix, we demonstrate the development of weights for the destroy and repair operators within the ALNS heuristic (ALNS + LS + RCP). As the development of weights exhibits a similar pattern for all instances across all instance groups, we only display the development for a single instance, specifically, M3-16D-42P-1 (Refer to Appendix D). The weight development for destroy and repair operators are depicted in Figure F.1 and Figure F.2, respectively. Please note that in Figure F.1, "shaw removal" represents the Relatedness Removal operator from Subsection 5.4.1.

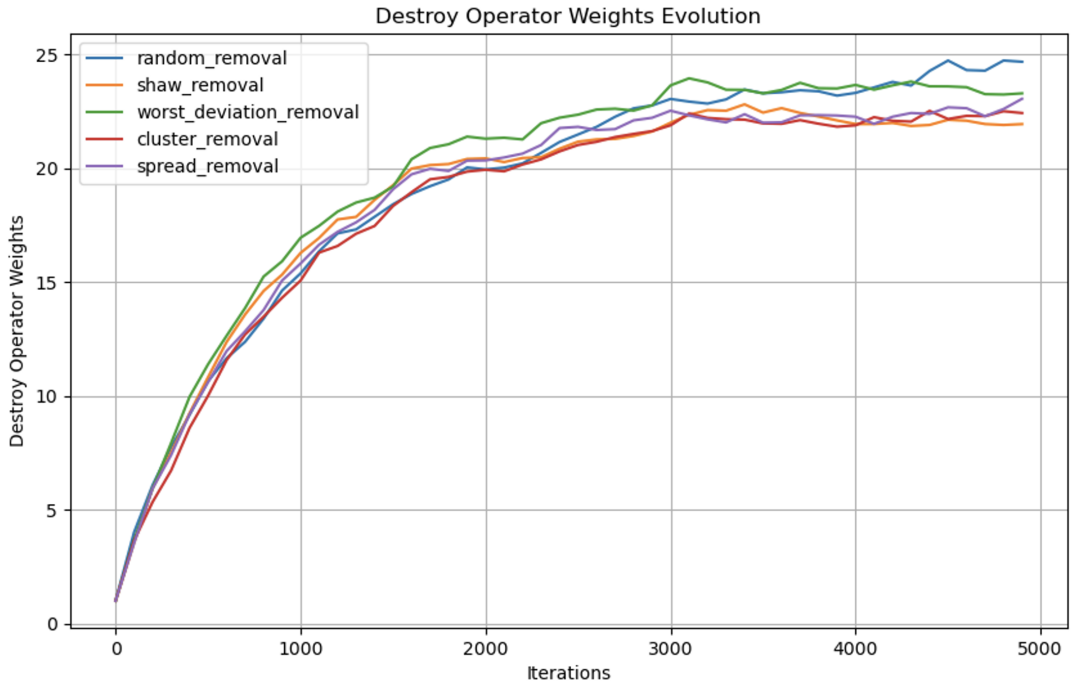


Figure F.1: Development of weights for the destroy operators over the duration of the ALNS process for instance M3-16D-42P-1. "shaw removal" represents Relatedness Removal operator

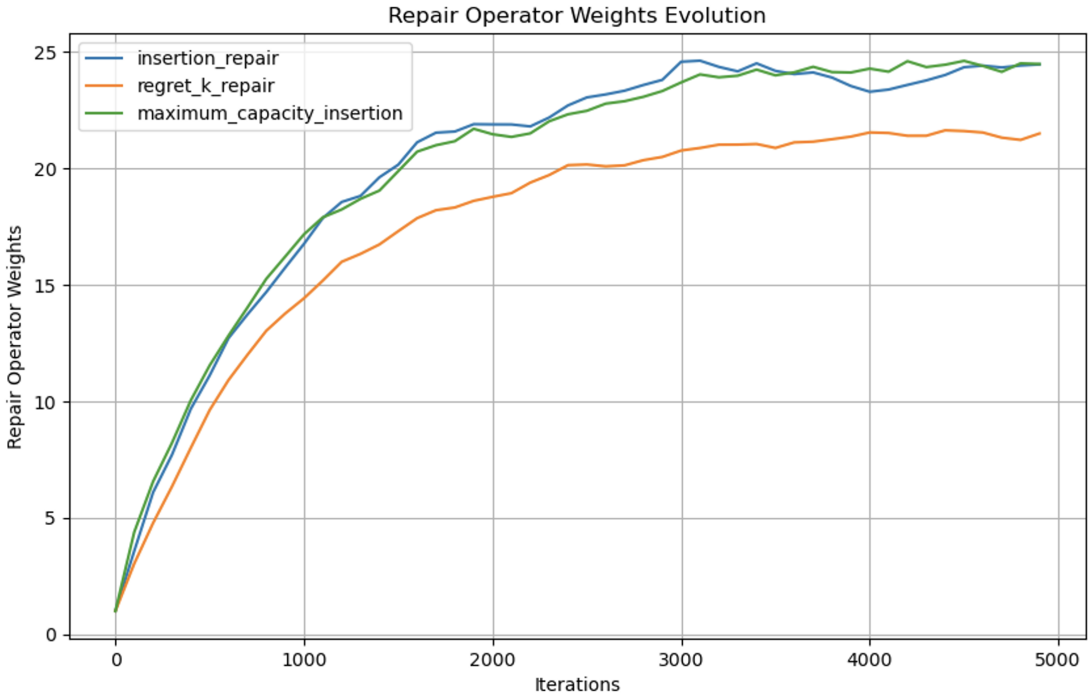


Figure F.2: Development of weights for the repair operators over the duration of the ALNS process for instance M3-16D-42P-1

Appendix G

Comparing ALNS to the Construction Heuristic

In this appendix, we aim to investigate the convergence of the ALNS algorithm towards solutions with satisfactory objective values by comparing the results of the best configuration of the ALNS heuristic (ALNS + LS + RCP) to the initial solutions provided by the construction heuristic (Section 5.3). This allows us to evaluate the effectiveness of the ALNS algorithm in solving the problem at hand and its ability to improve upon the solutions generated by the construction heuristic. Moreover, this comparative analysis provides valuable insights into the overall performance exhibited by both methods, underscoring their individual strengths and potential areas for improvement. It is noteworthy that the construction heuristic operates on a deterministic basis; hence, the coefficient of variation registers as 0.00% across all instance groups. This is due to the fact that each instance necessitates a single run due to the deterministic nature of the construction heuristic.

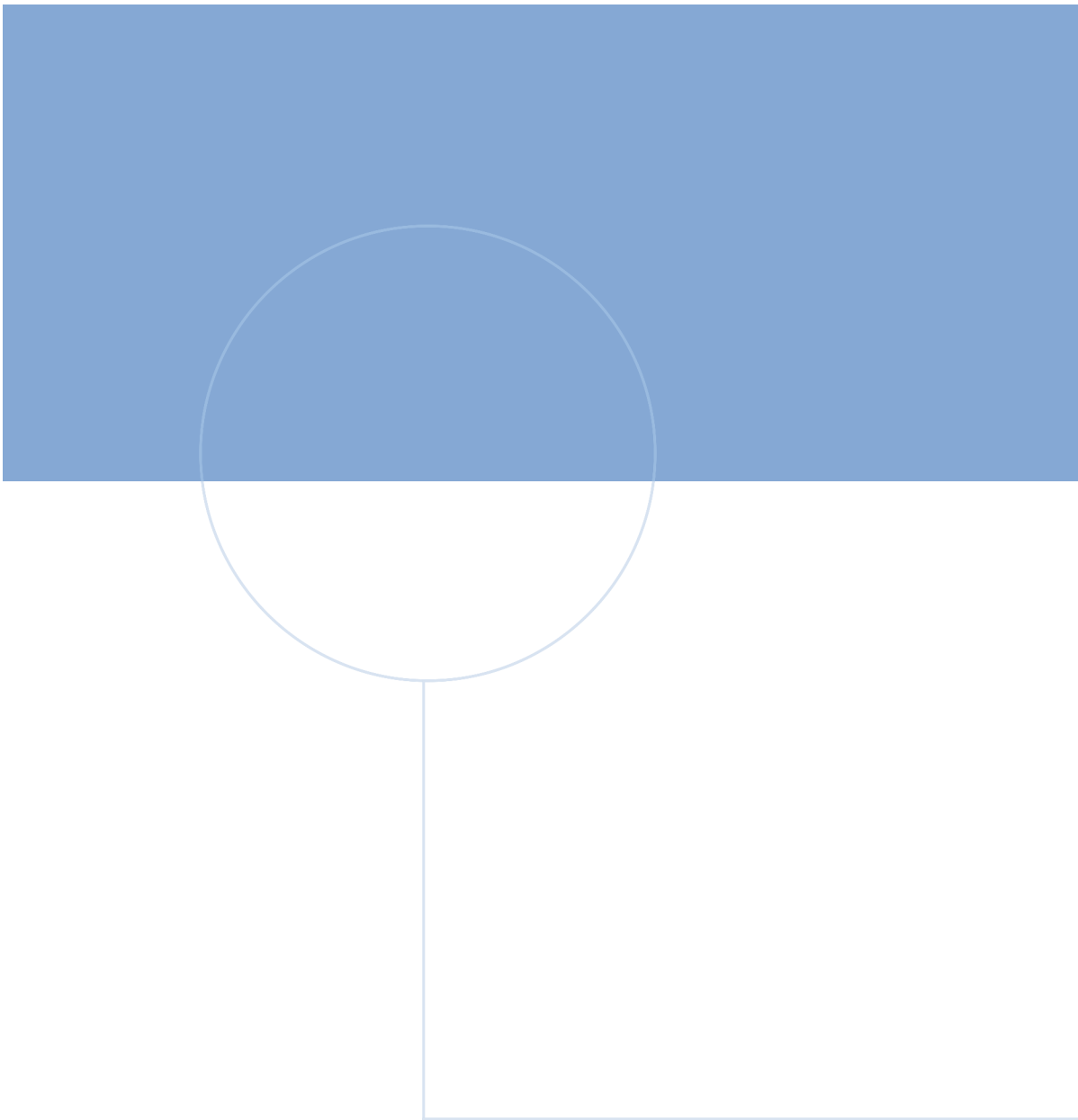
Table G.1 demonstrates a significant improvement in average objective values for all instance groups when using the ALNS heuristic. The construction heuristic identifies optimal solutions for the S1 instance group. For other instances, the construction heuristic generates objective values comparable to those found by the ALNS heuristic for smaller instance groups. However, it is unable to replicate the superior solutions generated by the ALNS heuristic. As the instance group size increases, the construction heuristic's ability to find high-quality objective values compared to the ALNS heuristic begins to diminish. For instance, the average gap for Objective 1, Gap^{Obj1}, stands at 6.87% for the construction heuristic as compared to a minimal 0.03% for the ALNS heuristic, a difference that is statistically significant. This suggests that the construction heuristic falls short in finding solutions where more passengers are picked up. Furthermore, the average gap for Objective 2, Gap^{Obj2}, stands at 3.85% for the construction heuristic, indicating that, in addition to picking up fewer passengers, it also tends to follow a longer average route.

In conclusion, while the construction heuristic provides a reasonable starting point for smaller instances, it evidently struggles as the instance group size increases, failing to find high-quality solutions consistently. This can be attributed to the simple and deterministic nature of the construction heuristic. As the complexity of the problem increases with the size of the instance groups, the heuristic's inability to adapt and explore various solution spaces becomes apparent. The ALNS heuristic, on the other hand, demonstrates greater performance, delivering superior solutions across all instance groups. In light of this, we

deduce that a simple greedy insertion construction heuristic is inadequate for tackling larger instances, and more sophisticated, adaptive algorithms like the ALNS are more equipped for such challenges.

Table G.1: Comparison of results for the construction heuristic and the ALNS heuristic with LS and RCP. $\overline{\textbf{Obj. 1}}$ and $\overline{\textbf{Obj. 2}}$ represent the average objective values for Objectives 1 and 2 for each instance group, respectively. CV^{Obj1} and CV^{Obj2} represent the average coefficient of variation for Objectives 1 and 2 for each instance group, respectively. Gap^{Obj1} and Gap^{Obj2} represent the average gap for Objectives 1 and 2 for each instance group across the construction heuristic and the ALNS heuristic, respectively. $\overline{\textbf{Time [s]}}$ represents the average time for the runs in each instance group, measured in seconds

Instance	**Construction Heuristic**							**ALNS + LS + RCP**						
Group	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	CV^{Obj1}	CV^{Obj2}	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$	$\overline{\textbf{Obj. 1}}$	$\overline{\textbf{Obj. 2}}$	CV^{Obj1}	CV^{Obj2}	Gap^{Obj1}	Gap^{Obj2}	$\overline{\textbf{Time [s]}}$
S1	3.6	42.50	0.00%	0.00%	0.00%	0.00%	0.0	3.6	42.50	0.00%	0.00%	0.00%	0.00%	53.5
S2	5.0	63.11	0.00%	0.00%	13.00%	4.37%	0.0	5.6	66.42	0.00%	0.00%	0.00%	0.00%	109.7
S3	8.8	128.61	0.00%	0.00%	6.94%	4.82%	0.0	9.4	132.42	0.00%	0.08%	0.00%	0.04%	274.6
M1	18.2	266.32	0.00%	0.00%	3.60%	3.58%	0.0	18.8	265.96	0.00%	0.07%	0.00%	0.03%	1408.9
M2	25.8	375.37	0.00%	0.00%	4.81%	3.05%	0.0	27.0	374.20	0.00%	0.05%	0.00%	0.10%	2990.2
M3	38.6	502.45	0.00%	0.00%	7.32%	3.27%	0.0	41.4	507.91	0.21%	0.78%	0.10%	1.23%	3400.2
L1	55.0	627.55	0.00%	0.00%	7.41%	5.86%	0.0	59.0	616.53	0.00%	0.18%	0.00%	0.19%	3600.0
L2	64.4	716.51	0.00%	0.00%	12.06%	4.85%	0.0	71.8	721.01	0.00%	0.49%	0.00%	0.38%	3600.0
L3	92.8	1029.83	0.00%	0.00%	6.73%	4.89%	0.0	99.0	1039.19	0.18%	0.75%	0.21%	0.99%	3600.0
Average	34.7	416.92	0.00%	0.00%	6.87%	3.85%	0.0	37.3	418.46	0.04%	0.27%	0.03%	0.33%	2115.2



NTNU

Norwegian University of
Science and Technology