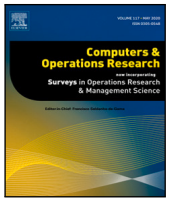
Computers & Operations Research 167 (2024) 106669

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



The static ridesharing routing problem with flexible locations: A Norwegian case study

Jacob Nitter, Shusheng Yang, Kjetil Fagerholt *, Andreas Breivik Ormevik

Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, NO 7491, Trondheim, Norway

ARTICLE INFO

Keywords:
Transportation
Routing
Ridesharing
ALNS heuristic

ABSTRACT

The municipalities in the Bergen region in Norway have recently announced a pilot project for ridesharing in the region as a means to reduce traffic congestion. As part of this project, we study the Static Ridesharing Routing Problem with Flexible Locations (SRRPFL), which aims at determining efficient routes and schedules for a set of drivers to pick up and deliver passengers at different, flexible pickup and delivery locations. We present a bi-objective mixed integer programming (MIP) model for the SRRPFL where we (lexicographically) first maximize the number of passengers serviced and then minimize the total travel times. To solve real-life instances of the SRRPFL, we propose a new Adaptive Large Neighborhood Search (ALNS) heuristic. To further improve its performance, we extend the ALNS heuristic with a local search, as well as with a set partitioning problem (denoted the Route Combination Problem) that optimally recombines the routes previously encountered in the search. The ALNS heuristic is tested on a number of test instances based on real trip data and the results demonstrate its effectiveness. The results also provide a number of insights regarding the potential benefits of ridesharing in our case study.

1. Introduction

The advantages of ridesharing, where individual travelers share vehicles, include the sharing of travel expenses such as fuel, tolls, and parking fees among participants, 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, perhaps due to the lack of efficient methods for coordinating schedules, as well as concerns regarding 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. As in many parts of the world, many people live 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. Furthermore, many places in Norway have low population density, which makes it hard to offer good public transportation services. This results in a higher dependency on private vehicles, possibly leading to increased traffic congestion (e.g., over bridges) and associated negative environmental impacts.

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 (second largest city in Norway), 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, traffic congestion is a common occurrence, causing delays and frustration for commuters. See Fig. 1 for a visualization of the geography in the area and how traffic develops at the Sotra bridge throughout a representative weekday. A majority of residents on Sotra who work in Bergen rely on this bridge to access their workplaces. Hence, 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 and emissions. 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. A well-designed ridesharing system could serve as a model for other similar regions facing similar transportation challenges, both in Norway and beyond. Moreover, ridesharing can help bridge the gap

* Corresponding author.
E-mail address: kjetil.fagerholt@ntnu.no (K. Fagerholt).

https://doi.org/10.1016/j.cor.2024.106669
Received 10 August 2023; Received in revised form 12 April 2024; Accepted 13 April 2024
Available online 16 April 2024

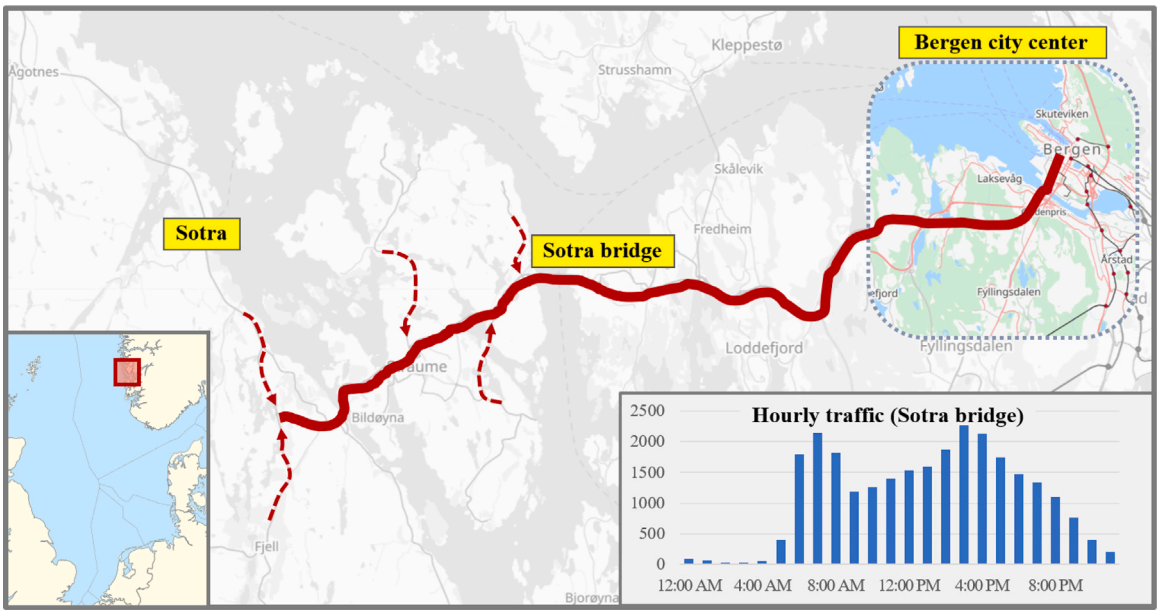


Fig. 1. Road networks on the island of Sotra. The bold line represents the heavily trafficked main road connecting the island to Bergen city center, and the hourly numbers of crossings are visualized (passenger vehicles only).

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 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, pickup and delivery locations. Recognizing the need for innovative transportation solutions, the municipalities in the Bergen region have recently engaged in discussions to explore potential strategies for addressing these challenges. Consequently, a pilot project for ridesharing in the region has been announced. As part of this initiative, with the aim to investigate how ridesharing can contribute to reducing traffic congestion in Sotra and the greater Bergen area, we consider in this paper the Static Ridesharing Routing Problem with Flexible Locations (SRRPFL). The SRRPFL aims at determining efficient routes and schedules for a set of drivers to pickup and deliver passengers at different, flexible pickup and delivery locations. The *flexible* locations means that passengers not necessarily will be picked up directly at their homes, instead they can travel a short distance to another candidate pickup 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 can be important in improving the overall efficiency of the system, enabling a better matching between drivers and passengers. The SRRPFL aims to optimize the ridesharing experience by maximizing the number of passengers that are serviced and minimizing the total travel time for all drivers. We therefore propose a bi-objective modeling approach where these two objectives are optimized in lexicographic ordering, i.e., we first maximize the number of passengers serviced while satisfying their and the drivers' timing requirements, and then minimize the total travel times for the drivers.

The main contributions of this paper can be summarized as follows: (1) We present a mixed integer programming (MIP) model for the bi-objective SRRPFL. (2) We propose a new and efficient Adaptive Large Neighborhood Search (ALNS) heuristic for solving the SRRPFL. The ALNS heuristic includes both destroy and repair operators well known from the literature, alongside new operators specifically devised for the SRRPFL. To further improve the performance of the ALNS heuristic, we incorporate a local search and a set-partitioning problem (denoted the Route Combination Problem), which optimally recombines the routes previously encountered in the search. (3) We apply the ALNS heuristic on a number of test instances for a real-world case study and show its excellent performance. (4) Through the case study, we provide a number of important managerial insights, e.g., by exploring the effects of having flexible locations and how ridesharing can contribute to reducing congestion and environmental emissions.

Section 2 reviews the related literature on ridesharing, while a definition of the SRRPFL is provided in Section 3. Section 4 presents the mathematical formulation of the problem, while Section 5 describes the ALNS heuristic. The computational study, including a description of the case study, is presented in Section 6, while concluding remarks are provided in Section 7.

2. Literature review

We start by categorizing ridesharing problems into *centralized* and *decentralized*, based on the degree of structure and formal organization present in the ridesharing arrangements. Centralized ridesharing represents scenarios where an entity (e.g., a company) coordinates and manages the ridesharing service for its customers. Examples of centralized ridesharing includes Uber, taxis and arrangements made by companies for its employees. On the other hand, decentralized ridesharing represents a more flexible approach where individuals independently rideshare based on their personal schedules and preferences. In this case, any individual driver may decide to pick up any other passengers traveling in the same direction (e.g., on its way to work). Understanding these differences is important as we delve further into the literature, as each type of ridesharing has its unique characteristic, challenge, and objective. It should also be noted that the ridesharing problem studied in this paper is decentralized, while most previous studies consider centralized versions of the problem.

Agatz et al. (2012) provide a comprehensive review of various types of ridesharing optimization problems. One key aspect of ridesharing systems is the underlying network structure, which determines the relationships between pickup and delivery locations, as well as the routes that drivers can take. The degree of dynamism is another important aspect, which can range from fully static (all information is known beforehand) to highly dynamic (information about riders and drivers

becomes available over time). The SRRPFL studied in this paper is a static ridesharing problem, as we assume that all participants (drivers and passengers) make available their requirements in due time before the transportation will take place.

Carpooling, as a specific form of ridesharing, was studied by Baldacci et al. (2004), who focus on matching passengers with drivers organized by companies that encourage their employees to pick up colleagues while driving to and from work. The objective is to minimize the sum of the costs to reach the workplace and the cost deriving from the penalties of the unserviced clients, while taking into account constraints such as vehicle capacities, time windows, and maximum travel times. The vanpool assignment problem, another specific form of centralized ridesharing, was studied by Kaan and Olinick (2013). They focus on assigning passengers to larger vehicles (vanpools). Similarly to the SRRPFL, they include alternative meeting points for passengers, which can accommodate a wider range of passenger preferences, potentially increasing the adoption of vanpooling as a sustainable transportation alternative.

He et al. (2023) address a first-mile transportation problem to intercity transportation hubs such as railway stations and airports. They dynamically group requests based on their arrival times and constraints such as maximum travel time requirements. They minimize the total transportation cost for the (centralized) ridesharing service provider. Auad-Perez and Hentenryck (2022) explore the concept of on-demand multimodal transit systems, which integrate bus or rail routes between transit hubs with on-demand shuttles. The authors incorporate ridesharing into shuttle rides and introduce new fleet-sizing algorithms to determine the necessary number of shuttles to transport a set of passengers. Zheng and Pantuso (2023) focus on a centralized ridesharing problem, which involves determining optimal routes for a fleet of vehicles to transport customers to a common destination via shared trips. The objectives are to minimize transportation costs and maximize service rates. They present an evolutionary algorithm based on Pareto dominance to solve their problem and perform tests on real-life data.

On-demand (dynamic) ridesharing services have gained popularity in recent years, presenting new challenges and opportunities for optimization. Fielbaum et al. (2021) study an on-demand (centralized) ridesharing systems with flexible locations. Their objective is to minimize routing and passenger walking cost, and to maximize passenger participation. Ghandeharioun and Kouvelas (2023) explore on-demand ridesharing by creating a real-time simulation framework and an optimization algorithm aimed at enhancing ridesharing operations. They develop a modular real-time simulation framework, and by using a New York City taxi dataset, the authors demonstrate that their algorithm outperforms the current taxi fleet in terms of service rate. Pelzer et al. (2015) present a partition-based matchmaking algorithm for dynamic ridesharing, i.e., to match passengers and drivers in real-time. The objective is to maximize mileage savings by sharing rides. Their results demonstrate that their 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.

Lin et al. (2019) explore a probabilistic demand-aware approach. To handle uncertain future demand, the authors consider the probabilities of future requests. They conduct numerical experiments based on real-world travel requests in Manhattan.

Most studies on ridesharing focus 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, Hsieh (2020) suggests a monetary incentive. Li et al. (2023) developed a generalized stochastic user equilibrium model, which formulates travelers' mode and route choice behavior. Recognizing the impact of ridesharing compensation on individual travel choices, 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.

Minimizing CO_2 emissions has become a vital concern, and Bruck et al. (2017) explore this issue in the context of a practical daily decentralized carpooling problem, used by companies to organize carpooling for its employees on a daily basis. Their study aims at providing an environmentally friendly approach to carpooling, emphasizing the importance of reducing the carbon footprint of transportation activities. The results indicate that the proposed approach can substantially reduce CO_2 emissions compared to individual commuting, demonstrating the environmental benefits of carpooling. Similar to the SRRPFL studied in this paper, Stiglic et al. (2015) recognize that traditional door-to-door ridesharing systems can result in significant detours for drivers and increased travel times for passengers. Hence, they investigate the advantages of incorporating meeting points, which serve as flexible pickup and delivery locations, facilitating more efficient routes and minimizing detours for drivers. Their 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) address a decentralized ridesharing problem for optimizing ride-matching and routing in ridesharing systems. The objective is to minimize total traveling distance for both drivers and passengers. The authors evaluate their proposed Large Neighborhood Search heuristic through computational experiments on a set of randomly generated instances, as well as real-world data sets, and demonstrate that it can provide high-quality solutions within reasonable computational times. Smet (2021) explores a large-scale decentralized ridesharing problem involving flexible drivers and the use of flexible locations, proposing a metaheuristic approach to address this challenge. Sun et al. (2020) study two versions of a non-profit peer-to-peer (decentralized) ridesharing problem, i.e., a static and a dynamic version. The authors propose an exact solution algorithm and a column generation based heuristic for the static version, while the dynamic version of the problem is tackled with two dynamic dispatching policies.

The ridesharing literature reviewed above is summarized in Table 1. The specific characteristics and contributions of the SRRPFL studied in this paper is highlighted in the final row of the table. We can note from the table that most studies are on centralized ridesharing problems. Among the studies on decentralized versions, only Stiglic et al. (2015), Bruck et al. (2017) and Smet (2021) consider flexible locations. Hence, these are the ones that are most similar to the SRRPFL. However, there are also some important differences.

In the ridesharing problem studied by Stiglic et al. (2015), a driver can only be matched with multiple passengers as long as these are picked up and delivered at the same locations (and at the same time). This means that there is always at most one pickup and delivery point along each driver route. Even though this results in rides that are easy to execute, it also represents a simplification compared to the SRRPFL. Bruck et al. (2017) consider a carpooling case where all participants share the same destination. This is in contrast to the SRRPFL, where the participants have individual destinations. Furthermore, Bruck et al. (2017) minimize the CO2 emissions, while the SRRPFL has two lexicographically ordered objectives (i.e., maximizing the number of serviced passengers and minimizing the total travel time for the drivers). The ridesharing problem studied by Smet (2021) is perhaps the one that is most similar to the SRRPFL. However, that problem does not include travel time constraints, and it has a single (weighted) objective. Smet (2021) does not present a detailed arc-flow formulation of the problem as we do here, and in contrast to our ALNS heuristic, their proposed

Table 1
Literature table. RS = Ridesharing, CP = Car pooling, VP = Vanpooling, Decen. = Decentr. ridesharing, Centr. = Centr. 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	Ccentr. vs. decentr.	Stat. vs. dyn.	Con.	FL	Objective	Case study	Solution method
Agatz et al. (2012)	Review: RS	-	Stat., dyn.	Multiple	✓	Multiple		Multiple
Baldacci et al. (2004)	RS/CP	Centr.	Static	TW, TT		RC, AR		Lagrangean column generation
Kaan and Olinick (2013)	RS/VP	Centr.	Static	TW	✓	RC	✓	Insertion heuristics
He et al. (2023)	RS	Centr.	Static	TW, TT, Cap		RC	✓	ALNS
Auad-Perez and Hentenryck (2022)	RS	Centr.	Static	TW, TT		RC	✓	Graph reformulation
Zheng and Pantuso (2023)	RS	Centr.	Static	TW		RC, AR	✓	Evolutionary algorithm
Fielbaum et al. (2021)	RS	Centr.	Static		✓	RC, WT, AR	✓	Graph heuristics
Ghandeharioun and Kouvelas (2023)	RS	Centr.	Dynamic	TW		US	✓	Simulation
Pelzer et al. (2015)	RS	Centr.	Dynamic			RC	✓	Partition-based
Lin et al. (2019)	RS	Centr.	Static	TW, MT		AR	✓	Probabilistic approach
Hsieh (2020)	RS	Centr.	Static			RC	✓	Swarm, evolutionary and firefly algorithms
Li et al. (2023)	RS	Centr.	Static			RC, CO_2		Genetic algorithm
Bruck et al. (2017)	RS/CP	Decentr.	Static	MD	✓	CO_2	✓	Insertion heuristic and local search
Stiglic et al. (2015)	RS	Decentr.	Static	TW	✓	RC, AR	✓	Insertion heuristic
Hou et al. (2018)	RS	Decentr.	Static	TW		RC		LNS
Sun et al. (2020)	RS	Decentr.	Stat., dyn.	TW, MP		RC, CS	✓	Column generation
Smet (2021)	RS	Decentr.	Static	TW	✓	RC, AR	✓	Late acceptance hill climbing
Our contribution	**RS**	**Decentr.**	**Static**	**TW, TT**	✓	**RC, AR**	✓	**ALNS**

metaheuristic produces solutions with rather large gaps (up to 20%) from the optimal ones (where these could be obtained).

There are also several other routing applications with flexible locations. One example of this is Dragomir et al. (2022) who consider a pickup and delivery problem with an application from online second-hand marketplaces. In this study, both the seller and buyer can specify a detailed plan of places scheduled for visit during the day. The transport provider can then choose between the different locations and corresponding time windows for pickup and delivery of the parcel to minimize its transportation cost. The SRRPFL does also share similarities with other routing problems from the sharing economy. Examples of this are crowdsourced deliveries and the use of occasional drivers, e.g., see the recent survey by Savelsbergh and Ulmer (2022) and Archetti et al. (2016). However, a crowdsourced delivery environment typically has a three-sided market (retailers, customers, couriers) as opposed to the two-sided market (customers, drivers) seen in most ridesharing applications. In the former, the price the customer pays is for the goods (which is also the primary reason for the transaction) and the delivery, while in the two-sided (ridesharing) market the price is for the transportation only. Crowdsourced deliveries are usually also centralized operations, in contrast to the SRRPFL.

3. Problem description

The Static Ridesharing Routing Problem with Flexible Locations (SRRPFL) involves multiple individuals participating in a decentralized ridesharing system, taking on one of two roles: a *driver* or a *passenger*. A driver is a participant who drives its 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.

Prior to when the ridesharing is supposed to take place (e.g., the evening before), we assume that all drivers and passengers submit their travel information, which is used as the input to the SRRPFL. The travel information for each driver/passenger includes an origin (e.g., its residence) and a destination location (e.g., its workplace), a time window for the delivery at the destination, and a maximum travel (ride) time. Hence, we assume that all necessary information is available as input before the SRRPFL is solved, which makes it a static problem. Furthermore, we assume that all drivers and passengers will participate in the ridesharing as long as the SRRPFL solution satisfies the conditions given by the input parameters (i.e., time windows and maximum travel time).

Each passenger has a given individual set of candidate pickup and delivery locations, typically a bus terminal or parking lots designated for commuters. Candidate pickup 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 on its own to its destination. A driver's (passenger's) time window includes the earliest and the latest time a driver (passenger) can be at their destination location, while the maximum travel time is the maximum time each driver (passenger) is willing to spend from its origin to its destination location. If a passenger is either picked up or delivered at a candidate location, the time it takes to travel to or from these candidate locations is included in the calculation of the total travel time. For drivers, the capacity represents the number of free seats in their cars or the number of passengers they are willing to pick up.

The SRRPFL involves the following decisions: (1) which passengers to be assigned to which driver or, alternatively, establish that a passenger will remain unserviced; (2) the pickup location for each designated passenger; (3) the delivery location for each passenger; and (4) the sequence of pickup and delivery locations for each passenger along each driver's route. All these decisions are made while keeping in mind its two objectives that are sorted lexicographically. The first and primary objective is to maximize the number of passengers that are serviced by any driver. The second (and secondary) 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 serviced.

Based on the geography and the travel pattern in our case study, we assume that each driver's route is split into two distinct phases: a pickup and a delivery phase. All pickups are first made during the pickup phase before all deliveries then are made during the delivery phase. Even though this assumption is made because we are focusing on a particular case study, it should be emphasized that the ALNS heuristic proposed in Section 5 can also handle the more general case where the pickups and deliveries are intertwined. The model in the next section can also easily be extended for that case.

Fig. 2 illustrates a tiny example problem based on our case study, including only one driver and two passengers, with a corresponding possible solution. Driver 1 starts driving from its own origin in *OD1* in Blomvåg. Driver 1 picks up passenger 1 before picking up passenger 2. Passenger 1 travels to its candidate pickup location *CP2P1* in Ågotnes

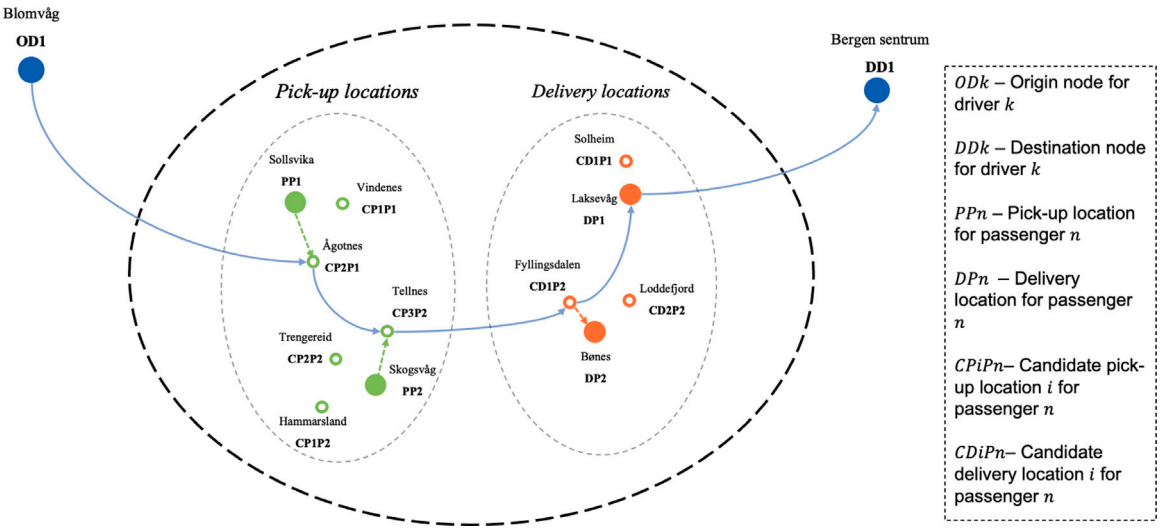


Fig. 2. Illustration of the SRRPFL with one driver and two passengers (location names from the case study).

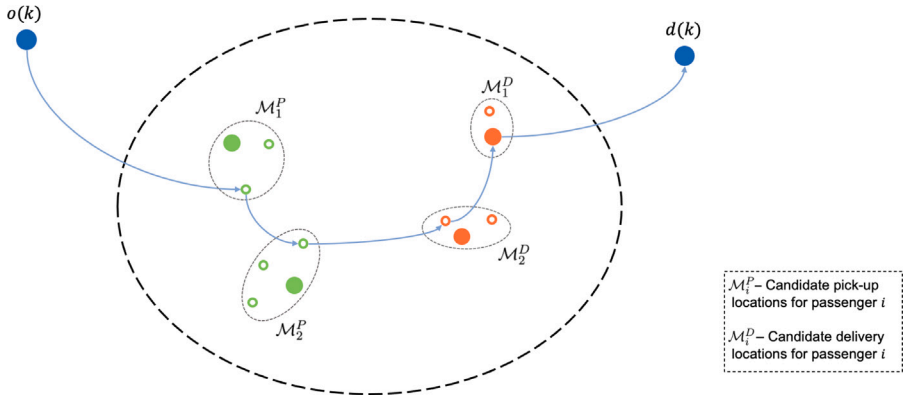


Fig. 3. Visual illustration of $\mathcal{M}_i^P$ and $\mathcal{M}_i^D$. Here, we have passenger $i = 1$ and $i = 2$.

from its origin in Sollsvika, while passenger 2 travels to its pickup location *CP3P2* in Tellnes from Skogsvåg. After the pickup phase, passenger 2 is the first to be delivered. Passenger 2 is delivered at its candidate delivery location *CD1P2* in Fyllingsdalen, before it travels to its destination, *DP2* in Bønes. Furthermore, passenger 1 is delivered at its destination, *DP1* in Laksevåg. Lastly, the driver travels to its destination, *DD1* in Bergen sentrum.

4. Mathematical model

In this section, we formulate the SRRPFL as an arc flow MIP model. Since the SRRPFL has similarities with the dial-a-ride problem (e.g., Ho et al., 2018) and the pickup and delivery problem with time windows (e.g., Parragh et al., 2008), we base our model partly on formulations for those problems, though extended with the alternative candidate locations for pickup and delivery. Section 4.1 introduces the notation, while the model is presented in Section 4.2.

4.1. Notation

Sets

The set of drivers is defined as D. This set also represents the drivers' origin locations. The set of passengers is represented as the set of the different passengers' origin locations, $\mathcal{P}^P$. The passengers also have designated destination locations, e.g., their workplace, and these are represented by 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 pickup locations, $\mathcal{M}_i^P$, and a set of candidate delivery locations, $\mathcal{M}_i^D$, for passenger $i \in \mathcal{P}^P$.

Since each passenger may have several alternative candidate locations for pickup and/or delivery, we have chosen to let a node in our network be represented as a combination of a passenger i and a candidate pickup/delivery location m of that passenger. Hence, the set of passenger pickup 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 pickup node is $(i, 0)$. The set of passenger delivery nodes is similarly 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 passengers can either be picked up or delivered. Note that the origin node $o(k)$ and destination node $d(k)$ for driver $k \in D$ is not a part of this set, thus, $\mathcal{N}^R = \mathcal{N}^P \cup \mathcal{N}^D$.

Table 2 summarizes all sets used in the SRRPFL, while Fig. 3, following the example in Fig. 2, further illustrates the sets $\mathcal{M}_i^P$ and $\mathcal{M}_i^D$. In this example, when the driver is en route to pick up passenger 1, there are three distinct candidate pickup nodes ($\mathcal{M}_1^P$) to choose among: one node representing the passenger's origin and two other candidate pickup nodes. For passenger 2, 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 pickup 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 ($\mathcal{M}_1^D$).

Table 2
All sets defined for the mathematical formulation of the SRRPFL.

Notation	Explanation
$\mathcal{D}$	Set of drivers $k \in \{0, 1, \dots, \lvert D\rvert\}$
$\mathcal{P}^P$	Set of passenger origin locations $i \in \{1, 2, \dots, N\}$
$\mathcal{P}^D$	Set of passenger destination locations $j \in \{N+1, N+2, \dots, 2N\}$
$\mathcal{M}_i^P$	Set of candidate pickup locations $m \in \{0, 1, \dots, \lvert\mathcal{M}_i^P\rvert\}$ for passengers $i \in \mathcal{P}^P$
$\mathcal{M}_i^D$	Set of candidate delivery locations $n \in \{0, 1, \dots, \lvert\mathcal{M}_i^D\rvert\}$ for passengers $i \in \mathcal{P}^P$
$\mathcal{N}^P$	Set of passenger pickup 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 $\mathcal{N} = \mathcal{N}^R \cup \{o(k)\} \cup \{d(k)\}$

Table 3
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 $k \in \mathcal{D}$

Table 4
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 $k \in \mathcal{D} \cup \mathcal{P}^P$ enters/leaves node $(i,m) \in \mathcal{N}$

Parameters

All parameters are summarized in Table 3.

Variables

The binary flow variable x_{kimjn} takes the value 1 if driver $k \in \mathcal{D}$ travels directly from node (i,m) to node (j,n), and 0 otherwise. Note that this variable is defined only between ridesharing nodes, $\mathcal{N}^R$. The binary variable x^S_{kim} is equal to 1 if driver k drives from its origin $o(k)$ to a candidate pickup node (i,m), and 0 otherwise. The binary variable x^E_{kjn} is equal to 1 if driver k travels from candidate delivery node (j,n) to its destination node $d(k)$, and 0 otherwise. The binary variable x^{OD}_k is 1 if driver k 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 does not pick up any passengers. The binary variable y_{im} is 1 if passenger i is picked up/delivered at candidate location (i,m), and 0 otherwise. The binary variable z_{ki} is 1 if driver k services passenger i, and 0 otherwise. Finally, the continuous variable t_{kim} defines the time when driver k enters node (i,m). It should be noted that since the service time in the nodes are assumed to be zero and no waiting is allowed, the variable t_{kim} also defines the time when driver k leaves node (i,m).

Table 4 summarizes all variables, while Fig. 4 illustrates the flow 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.

4.2. Mathematical formulation

Objective functions

The model is formulated as a bi-objective optimization problem with lexicographical ordering, where one objective is considered to be of much higher importance than the second. Here, the first and most important objective function (1) maximizes the number of passengers that are picked up (serviced), while the second objective function (2) minimizes the total travel time for the drivers (while keeping the first objective at its optimal value).

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

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

Routing constraints

Constraints (3) and (4) ensure that each driver must leave its origin and arrive at its destination node exactly once, respectively.

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

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

Constraints (5) and (6) ensure flow conservation for pickup and delivery nodes, respectively.

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

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

Constraints (7) and (8) ensure that each passenger is picked up and delivered by at most one driver, respectively, while constraints (9) make sure that each passenger is picked up or delivered at no more than one candidate location (i.e., serviced at most once). Constraints (10) connect the z_{ki} and y_{kim} variables.

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

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

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

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

Coupling and precedence constraints

Constraints (11) ensure that any driver who picks up a passenger also delivers the passenger, while constraints (12) enforce that deliveries must occur after the corresponding pickups.

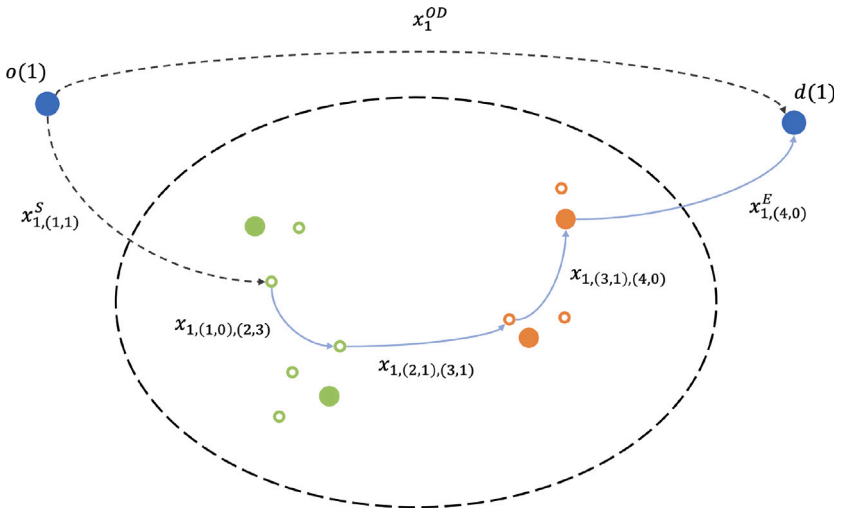


Fig. 4. Visual representation of the flow variables x^S_{kim}, x_{kimjn}, x^E_{kjn}, and x^{OD}_k.

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

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

Time constraints

Constraints (13)–(20) keep track of time along each driver's route. Constraints (13)–(14) are valid for the case when traveling between two ridesharing nodes; constraints (15)–(16) are for the case when a driver enters a ridesharing node directly from its origin node; constraints (17)–(18) for the case when a driver travels from a ridesharing node to its destination; while constraints (19)–(20) are for the case when a driver travel directly from its origin to its destination without picking up any passengers. It should be noted that the combination of these pairwise set of constraints prohibits drivers from waiting at any node, which would not be practical in such a decentralized ridesharing application. The parameters M (with any indices) in the following constraints are big-M parameters, which are given suitable and sufficiently large values.

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

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

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

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

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

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

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

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

Constraints (21) ensure that each driver arrives at its destination within its time windows, while constraints (22) do the same for all passengers.

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

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

Constraints (23) ensure that the total travel time does not exceed the maximum travel time for drivers, while constraints (24) do the same for passengers.

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

$$t_{k,N+i,0} - t_{ki0} \le T^M_i + M_{ki0}(1 - z_{ki}), \quad k\in\mathcal{D}, i\in\mathcal{P}^P \tag{24}$$

Constraints (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. Similarly, constraints (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.

$$t_{ki0} \le t_{kim} - T^C_{im} y_{kim}, \quad k\in\mathcal{D}, (i,m)\in\mathcal{N}^P \tag{25}$$

$$t_{kj0} \ge t_{kjn} + T^C_{jn} y_{kjn}, \quad k\in\mathcal{D}, (j,n)\in\mathcal{N}^D \tag{26}$$

Capacity constraints

Constraints (27) ensure that each driver's capacity is never exceeded.

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

Variable definitions and domains

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

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

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

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

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

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

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

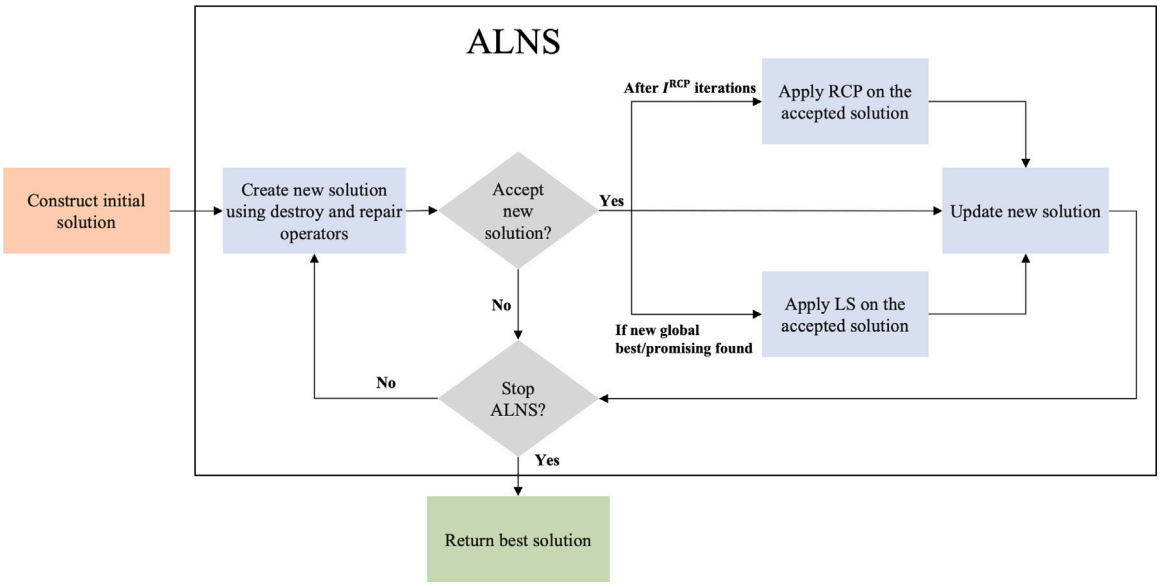


Fig. 5. Flowchart representing the processes in the ALNS heuristic. LS = Local search. RCP = Route combination problem.

When defining the flow variables, we can efficiently use the time and precedence constraints to reduce the size of the network, and hence the number of variables. Note also that, following the problem definition in Section 3, we assume in the model above that each driver's route is split into a distinct pickup and delivery phase. The model can however easily be extended to the situation with intertwined pickups and deliveries. In that case, we would need additional load variables and constraints to keep track of each driver's load at any node, similarly to how this is modeled in the pickup and delivery problem (e.g., Desrosiers et al., 1995.)

5. Adaptive large neighborhood search heuristic

This section describes the proposed Adaptive Large Neighborhood Search (ALNS) heuristic. The choice of using an ALNS heuristic is mainly motivated by its track record in successfully solving similar routing problems. Additionally, the flexibility of ALNS heuristics allows for the development of tailored destroy and repair operators specific to our problem.

Section 5.1 provides an overview of the ALNS heuristic, while the following subsections describe its different components in more detail.

5.1. Overview of the ALNS

The ALNS heuristic builds upon the research of Ropke and Pisinger (2006), who developed an ALNS heuristic for the pickup and delivery problem with time windows. The ALNS explores the solution space by iteratively destroying and repairing solutions using a set of destroy and repair operators. An acceptance criterion is used to determine whether to accept a repaired solution or not. Additionally, once a solution is accepted, a local search (LS) heuristic is here integrated within the ALNS to help refine and improve the solutions generated by the ALNS. The LS allows for intensifying the search and find local improvements in the vicinity of a solution. Furthermore, we also extend the ALNS heuristic by including what we denote as the route combination problem (RCP), where individual driver routes found during the search can be recombined to find new and improved solutions. Fig. 5 shows a flowchart illustrating the interaction between the different components of the proposed ALNS heuristic.

The ALNS algorithm is further outlined in Algorithm 1. Initially, the algorithm sets the current solution x by first constructing a feasible initial solution, as described in Section 5.2. This is followed by setting the global best solution x^* to the current solution x. In each iteration, the algorithm selects a destroy and a repair operator among a set of destroy and repair operators Ω_i^- and Ω_i^+, respectively. 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, similar to Ropke and Pisinger (2006). If the candidate solution x' is accepted as the new global best solution, the LS is applied to search in the local neighborhood for improved solutions as described in Section 5.5. 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^*, the LS is applied if x' is considered a *promising solution*. A promising solution is one with an objective value within a predefined threshold $\delta\%$ of the current best global solution's objective value. If the candidate solution x' is accepted as the new global best solution after the LS, both the current solution x and the global best solution x^* are updated accordingly, along with the objective value. 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 still accepted by the acceptance criterion, 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 in a similar way as in Ropke and Pisinger (2006). After completing a given number of iterations, the algorithm updates the destroy and repair operator weights for the next segment, $p_{i,m+1}^-$ and $p_{i,m+1}^+$, also following the procedure by Ropke and Pisinger (2006). Then, the destroy and repair operators are chosen based on a roulette wheel selection where their probabilities for being chosen are based on the operators' weights, which again are based on their previous performances. If a specific operator repeatedly results in new and better solutions, it gets a higher probability for being selected later, and vice versa. After every I^{RCP} iterations, the algorithm solves the RCP on x' to potentially find a new global best solution, further described in Section 5.6. The ALNS algorithm terminates when all iterations are completed, and the global best solution x^* is returned as the output.

In the following, we describe the different components of the ALNS heuristic in more detail.

Algorithm 1 ALNS

Input: Total number of ALNS iterations (I^{ALNS}) and number of iterations between solving the RCP (I^{RCP})
Set current solution x by constructing a feasible initial solution *(Section 5.2)*
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^-
Initialize repair operators Ω_i^+, operator weights $p_{i,m}^+$ and operator scores s_i^+
for *iteration* $= 1$ to I^{ALNS} **do**
    Select destroy and repair operators $d \in \Omega_i^-$ and $r \in \Omega_i^+$ using the weights $p_{i,m}^-$ and $p_{i,m}^+$
    Generate a candidate solution x' from the current solution x using d and r
    Generate acceptance probability P^{SA} by simulated annealing (SA)
    if x' is accepted as the new global best solution **then**
        Apply local search for improving candidate solution x' *(Section 5.5)*
        $x \leftarrow x'$
        $x^* \leftarrow x'$
        $f(x^*) \leftarrow f(x')$
        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 5.5)*
        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 5.6)*
    end if
end for
Output x^*

5.2. Construction of an initial solution

The algorithm for constructing an initial feasible solution 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 travel time. It does this by examining all possible positions for the pickup and delivery nodes of the passenger into each driver's route, while ensuring feasibility with respect to capacity, maximum travel times, and time windows. Then, the algorithm calculates the cost increase between the new route with the inserted pickup 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 pickup 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.

5.3. ALNS destroy operators

In the following, we describe our destroy operators in the set Ω_i^- designed to destroy portions of the current solution. Some of the following destroy operators are standard ones defined by Ropke and Pisinger (2006), while others are new ones designed specifically for the SRRPFL.

Random removal

The random removal operator removes a specified number of passengers 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 pickup 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 negative 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. 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, defined as $R(i,j) = T^D_{imjn} + T^D_{i+N,m,j+N,n}$, where T^D_{imjn} is the direct travel time between pick up nodes (i,m) and (j,n) and $T^D_{i+N,m,j+N,n}$ is the direct travel time between delivery nodes $(i+N,m)$ and $(j+N,n)$.

The relatedness removal consists of finding passengers that have close pickup and delivery nodes to the seed passenger's pickup and delivery nodes. 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 candidate pickup 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 pickup location of each passenger in the selected driver's route and the pickup 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 pickup 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 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.

5.4. ALNS repair operators

In the following, we describe our 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 two first ones are similar to the ones defined by Ropke and Pisinger (2006), while the third and last one is specially designed for the SRRPFL.

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 pickup locations, but with a randomized order of evaluation. For each feasible pickup insertion, a temporary route is created, and the operator proceeds to examine all possible corresponding delivery locations.

For each feasible delivery insertion, the operator calculates the cost increase associated with the insertion of the passenger at the given pickup 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, pickup 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 calculated as $max\{\sum_{j=1}^{k-1}(c_i - c_{i+j})\}$ 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 any driver with 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.

5.5. Local search

We integrate LS within the ALNS heuristic to enhance the discovery of *promising solutions*. As described in Section 5.1, the LS 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 LS is initiated under specific conditions. The candidate solution x' must pick up the same number of passengers as the current global best solution (objective 1), and its total travel time should be within a predefined threshold $\delta\%$ of the current best global solution's total travel time (objective 2). If one of these conditions (i.e., the solution is considered promising), the LS is conducted, which either results in an improved solution to x' or no change at all. When the LS is initiated, the LS Operators (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.

In the following, we describe the LSOs that are used within our ALNS heuristic.

Intra-passenger swap

The *intra-passenger swap* operator modifies the order of passenger pickups and deliveries within each driver route. It consists of two separate sequential procedures. *Passenger pickup swap* modifies the order of passenger pickups within a driver's route. For each driver route, it iterates through all possible pairs of pickup 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. To further explore the solution space, the operator also considers all candidate pickup locations for the swapped nodes. After the pickup swap is done, *passenger delivery swap* modifies the order of passenger deliveries within a driver's route, similarly as the pickup swap.

Inter-passenger relocate

This operator swaps pickup and delivery nodes between two different drivers' routes. It starts by filtering out the available drivers' routes based on their available capacities. Then, for each driver, it iterates through its pickup nodes and its corresponding delivery nodes. Next, it selects another driver and does the same. The pickup 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 it checks for feasibility and calculates the new objective values for both objectives. If the new solution is improved, the new routes are accepted, and the process continues iterating. The inter-passenger swap operator continues until no further improvements can be found.

Candidate location-shift

This operator shifts the candidate location for a passenger's pickup and delivery location within a driver's route. For each driver route, it iterates through all pickup nodes and checks the other candidate locations for the same passenger. Once all passenger pickup nodes have been considered in a route, the operator does the same with the delivery nodes.

5.6. Route combination problem

Each driver has a route in any 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 further enhance the performance of the ALNS heuristic, we introduce the Route Combination Problem (RCP), following the ideas of Homsi et al. (2020) and Ulsrud et al. (2022).

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 some of the notation presented in Section 4.1 along with the following additional notation. The set $\mathcal{R}_k$ includes all routes previously identified as available for driver k during the ALNS search. The parameter N_{kr} defines the number of passengers picked up by driver k on its route r, while T_{kr} is the travel time for driver k on route r. The binary parameter A_{ikr} is 1 if passenger i is picked up by driver k on its route r, and 0 otherwise. Finally, we define the binary decision variable x_{kr}, which takes the value 1 if driver k travels its route r, and 0 otherwise.

Now, we can define the RCP as follows. Similar to the objective functions described in Section 4.2, the RCP is also formulated as a bi-objective optimization problem with the following two objective functions, lexicographically ordered, where objective function (35) maximizes the number of passengers that is serviced, while objective function (36) minimizes the total travel time for drivers.

$$\max z_1 = \sum_{k\in D}\sum_{r\in\mathcal{R}_k} N_{kr}x_{kr} \tag{35}$$

$$\min z_2 = \sum_{k\in D}\sum_{r\in\mathcal{R}_k} T_{kr}x_{kr} \tag{36}$$

The two objective functions are maximized/minimized subject to the following constraints, where constraints (37) make sure that each passenger is a part of at most one route, while constraints (38) ensure that exactly one route is selected for every driver. Lastly, constraints (39) put a binary requirement on the route variables.

$$\sum_{k\in D}\sum_{r\in\mathcal{R}_k} A_{ikr}x_{kr} \le 1, \quad i \in \mathcal{P}^P \tag{37}$$

$$\sum_{r\in\mathcal{R}_k} x_{kr} = 1, \quad k \in \mathcal{D} \tag{38}$$

$$x_{kr} \in \{0,1\}, \quad k \in \mathcal{D}, r \in \mathcal{R}_k \tag{39}$$

6. Computational study

In this section, we present the computational study. 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 with four 2.2 GHz AMD Opteron 6274 processors with 16 cores each and 128 GB of RAM. The ALNS heuristic and instance generator were programmed in Python v3.9.6. The arc-flow MIP model (Section 4) and the model for the route combination problem (RCP) (Section 5.6) were run using Gurobi v9.5. The ALNS heuristic has two stopping criteria: (1) a predetermined number of I^{ALNS} iterations have been performed, or (2) the runtime reaches 3600 s. 3600 s (one hour) is assumed to be a practical maximum run time in our case study since the SRRPFL is supposed to be solved on a daily basis (e.g., the evening before the commutes take place). Similarly, the maximum runtime for the commercial solver is set to 3600 s. Due to the randomness of the ALNS heuristic, each test instance is run five times to obtain a reliable estimate of the algorithm's average performance. Thus, when presenting the results from running the ALNS heuristic on a test instance, we report the average results over five independent runs.

The case study and the test instances generated based on this are described in Section 6.1. In Section 6.2, we test different configurations of the ALNS heuristic and select the best performing one for further experiments. Section 6.3 compares the performance of the ALNS heuristic to a commercial MIP solver. Finally, in Section 6.4 we conduct some additional analyses to provide managerial insights.

6.1. Case study and test instances

As pointed out in Section 1, we consider the case study of ridesharing in the region of Bergen, Norway's second largest city. More specifically, we consider the ridesharing for the morning traffic from Sotra to the greater Bergen area. To generate realistic test instances, we use relevant trip data provided by Telia, a Nordic telecom company, which includes hourly trip counts for each calendar day. This data enables us to pinpoint the origin and destination locations for each trip from Sotra to the greater Bergen area in the morning hours. Fig. 6 shows the most important origin and destination locations which we use when generating the test instances. The 43 origin locations on Sotra are divided into three distinct zones, whereas the 20 destination locations are contained within a single zone.

Based on this trip data, we randomly generate a number of test instances of different sizes. The main set of instances contain 45 instances with up to 35 drivers and 100 passengers. Additionally, we also generate a smaller set of instances used for tuning and setting up the ALNS heuristic. We use the filtered travel data for Sotra's residents from 06:00 to 10:00 between October 3rd, 2022, and October 30th, 2022, to generate trips, both for the drivers and passengers. Each driver and passenger in each instance has a maximum travel time and a time window for when to arrive at its delivery location. The time window is randomly chosen within 06:00 and 10:00 with a width of 30 min. We set the capacity of each driver's vehicle to four.

For each passenger trip with a given origin location, we define the set of candidate pickup and candidate delivery locations, defined by the sets $\mathcal{M}_i^P$ and $\mathcal{M}_i^D$, respectively. The set $\mathcal{M}_i^P$ includes the origin location

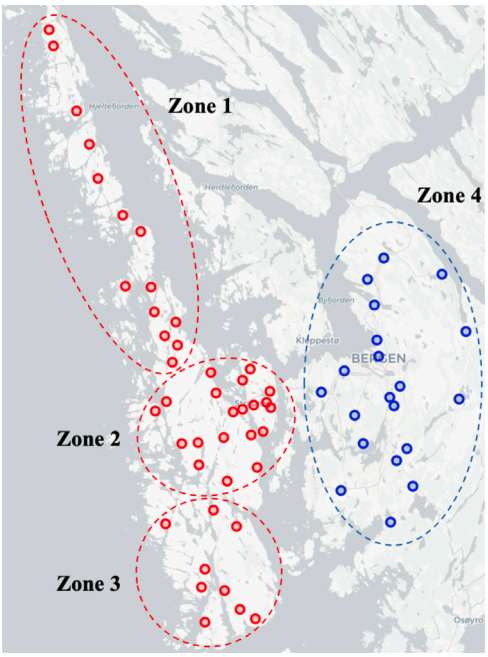


Fig. 6. Map of the Sotra region, showing the origin (red markers) and destination (blue markers) locations in their respective zones.

Table 5
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

of passenger i as well as up to the θ closest origin locations for other passengers and drivers that are within ρ minutes of travel time (by car) from the origin location of passenger i. Unless not stated otherwise, $\theta = 3$, while ρ is set to 10 min. For the set of candidate delivery locations $\mathcal{M}_i^D$, we choose not to include additional candidate destination locations, i.e., the set consists only of the passenger's delivery location. This is based on the assumption that even though passengers can be willing to make a short travel to be picked up, they generally prefer to arrive at their exact destination locations.

The instances are, as summarized in Table 5, grouped into three categories based on their sizes, i.e., S (Small) – with one to four drivers and four to 10 passengers, M (Medium) – with eight to 16 drivers and 20 to 42 passengers, and L (Large) – with 20 to 35 drivers and 60 to 100 passengers. The Instance ID column displays how the different instances in each instance group are identified, where X indicates a unique instance. Five instances are generated in each group.

6.2. Configuration and assessment of the ALNS heuristic

There is a number of parameters in the ALNS heuristic in addition to setting up the extensions regarding the local search (LS) and the Route Combination Problem (RCP). Since we experienced that the ALNS parameters did not affect the solutions much compared to the LS and RCP, we focus on these two extensions in the following. The values for the other ALNS parameters were, after thorough testing, mainly chosen as in Ropke and Pisinger (2006), except for the following ones. We chose 5000 iterations for the ALNS (I^{ALNS}) to balance the trade-off between solution quality and computational time. The weights of the destroy and repair operators were updated after each 100 iterations. The frequency of solving the RCP in the ALNS heuristic is a trade-off between the solution time of the RCP and taking advantage of the new information from its solution. Based on preliminary testing, we set the RCP to be solved every 300 iterations (I^{RCP}). For the regret-k destroy operator (Section 5.3), we set the parameter $k = 3$. The percentage factor δ was set to 90%, which means that a candidate solution is considered promising if its value for objective 2 is within 90% of the global best's value for that objective.

In the following we test the effect of the LS and RCP extensions.

Table 6 provides a detailed comparison of the performance of the ALNS heuristic without and with its LS extension, i.e., ALNS and ALNS + LS, respectively, across the instance groups (S1 to L3). Table 7 shows the same for the extensions with RCP both without and with LS, i.e., ALNS + RCP and ALNS + LS + RCP, respectively. Each extension's effectiveness is evaluated based on these parameters: the average coefficients of variation across all instances within each instance group for Objectives 1 and 2, respectively (CV^{Obj1} and CV^{Obj2}), the average gap for Objectives 1 and 2 (Gap^{Obj1} and Gap^{Obj2}) to the best known solution (obtained across all runs by the best version of the ALNS heuristic), and the average computational time.

The results in Table 6 show that the ALNS + LS (ALNS heuristic with local search) outperforms the basic ALNS in terms of average gap for both objectives. However, the results in Table 7 show that adding the RCP improves the results further. Comparing the performance of ALNS + RCP against ALNS + LS, as shown in Table 6, we observe that ALNS + RCP demonstrates superior performance, both in the two objective values, as well as in the coefficients of variation. Furthermore, the results in Table 7 also demonstrate that when we add the LS to the ALNS + RCP, the results become even better. This configuration (ALNS + LS + RCP) yields the lowest average gaps at only 0.03% for objective 1 and 0.33% for objective 2 on average across all five runs on the 45 test instances. The very low coefficients of variation (0.04% for objective 1 and 0.27% for objective 2) also demonstrate the stability of this configuration's performance.

Based on the findings presented in this subsection, we observe that the ALNS configuration that includes both LS and RCP (i.e., ALNS + LS + RCP) has the best average performance. Hence, we conclude that this is the best configuration that we use in the remaining analyses, and for simplicity, we refer to it in the following just as the ALNS heuristic (even though it includes the LS and RCP extensions).

6.3. Comparing the ALNS heuristic with a commercial solver

This section provides a comparison of the solutions produced by the ALNS heuristic and the commercial solver, Gurobi. As outlined previously, the maximum run times of both approaches are set to 3600 s. Since the SRRPFL is a bi-objective optimization problem, the current version of Gurobi cannot produce gaps, upper, and lower bounds for both objectives. The gaps, referred to as Gap^{GObj1} and Gap^{GObj2}, therefore represent the relative difference between the average objective value found by the ALNS heuristic and that found by Gurobi (negative values mean that the ALNS heuristic obtains a better solution than Gurobi).

Table 8 presents the results from this comparison.

As can be observed from the results in Table 8, the commercial solver obtains optimal solutions for the small instances in the groups S1, S2, and S3. Comparatively, the ALNS heuristic finds the same objective values for instance groups S1 and S2, but not for S3. For S3, the ALNS heuristic finds a slightly lower (worse) objective 1 than Gurobi in two out of the five runs of one of the five instance. In other words, the ALNS heuristic does obtain the optimal value to objective 1 in three out of five runs for that instance and in all five runs for the other four instances in that group. Furthermore, we see that the ALNS heuristic

Table 6
Comparison of results for the ALNS heuristic with and without the LS extension. 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. $\overline{\textbf{Time[s]}}$ 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

Table 7
Comparison of results for the ALNS heuristic with the RCP extension. 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. $\overline{\textbf{Time[s]}}$ 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

Table 8
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 (with LS and 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

finds on average slightly better solutions regarding objective 2 for instance group S3. It should be noted that, since the objectives are lexicographically ordered, this might come as a result of that the ALNS could not find the best solution for first objective 1 in the few cases mentioned above.

For the instance group M1, the commercial solver reaches its time limit of 3600 s 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%. This demonstrates that the ALNS heuristic produces solutions of substantially higher quality than the commercial solver within the 3600 s time limit. If we consider the prioritized objective 1, i.e., the number of passengers serviced, we see from Table 8 that the average objective value across all M1 instances is 18.8. Since the M1 instances contain 20 passengers, we know that the upper bound on objective 1 cannot be higher than 20. Hence, we can conclude that we are on average at most 1.2 serviced passenger away from the optimal solution (and probably less than that). Similarly, if we look at the L3 instances, the upper bound for objective 1 is 100 (since there are 100 passengers), and we see that the ALNS solution is at most 1% away from this upper bound. For the subsequent instance groups, the commercial solver is unable to find feasible solutions within the 3600 s time limit as the problem size increases. For these instances, we also let the commercial solver run for over 48 h without finding any feasible solutions.

By comparing the ALNS heuristic with the commercial solver, it is clear that the ALNS heuristic demonstrates promising capabilities, potentially offering high-quality solutions even for complex and large-scale instances. Since we do not know the optimal solutions for the largest instances, we do not know exactly how well the ALNS heuristic performs on these, but the high stability of the solutions (given by the low coefficients of variations) indicate that also these are very good. Furthermore and more importantly, as shown in Table 8, our solution values for objective 1 are close to the number of passengers in the

Table 9
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 (in minutes) it takes for passengers who are picked up at a candidate location to travel to that specific candidate location.

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

instances (shown in the parentheses of each Instance Group in the table). The number of passengers in each instance is the theoretical maximal upper bound for objective 1 since we maximize the number of serviced passengers. For example, over the five instances in the largest group L3, we service on average 99.0 out of the 100 passengers, which means that our solutions are at most 1% away from the optimal ones on average. This further demonstrates the effectiveness of the ALNS heuristic.

6.4. Value of having candidate locations

The motivation to incorporate candidate locations into the ridesharing system is to enhance the flexibility for both drivers and passengers. Table 9 provides a comparison of the results obtained with the ALNS heuristic, with ($\rho = 10$ min) and without ($\rho = 0$) candidate locations. As recalled from Section 6.1, ρ denotes the maximum time a passenger is willing to travel from its origin to reach a candidate pickup location. Here, $\rho = 0$ min implies that the only pickup location for each passengers is its origin location.

The results in Table 9 clearly illustrate the impact of introducing candidate locations. There is an increase in the average number of passengers serviced across all instance groups from 35.1 without to 37.3 with candidate locations. Furthermore, candidate locations contribute to shorter average travel times for drivers, where the average value of across all instance groups decreases from 477.25 without to 418.46 with candidate locations.

Candidate locations, when available, are frequently used, as indicated by the high average **CP** value of 82.82%. Furthermore, the average travel time for a passenger to reach a candidate location is not excessively long at 4.15 min, and in any case no longer than 10 min for any passenger (since $\rho = 10$). This indicates that candidate locations do not necessarily 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. The average computational time, $\overline{\textbf{Time [s]}}$, nearly doubles from 1154.6 s without to 2115.2 s with candidate locations. Despite the increased computational time, this analysis highlights the usefulness of having candidate locations in ridesharing. Their use presents an effective strategy for balancing the dual objectives of maximizing passenger pickups and minimizing the driver travel times.

7. Summary and concluding remarks

Recognizing the need for innovative transportation solutions to reduce road congestion, the municipalities in the Bergen region in Norway have recently announced a pilot project for ridesharing in the region, including Sotra and the greater Bergen area. As part of this project, we have studied the Static Ridesharing Routing Problem with Flexible Locations (SRRPFL). The SRRPFL aims at determining efficient routes and schedules for a set of drivers to pick up and deliver passengers at different, flexible pickup and delivery locations.

We presented an arc-flow bi-objective mixed integer programming (MIP) model for the SRRPFL where we (lexicographically) first maximize the number of passengers serviced and then minimize the total travel times. Due to the size and complexity of the problem, commercial MIP solvers are only able to solve tiny instances. Therefore, we also proposed a new Adaptive Large Neighborhood Search (ALNS) heuristic for solving the SRRPFL. The ALNS heuristic includes well known destroy and repair operators from the literature, alongside new operators specifically devised for the SRRPFL. To further improve the performance of the ALNS heuristic, we also added local search (LS) and a set partitioning problem, denoted the Route Combination Problem (RCP), which optimally recombines the routes previously encountered in the search.

The ALNS heuristic was tested on a number of test instances based on real trip data. The results on 45 test instances of different sizes showed that incorporating the LS and RCP extensions to the ALNS heuristic significantly improves the quality of the solutions. When comparing with optimal solutions from the commercial MIP solver on small test instances and a maximal upper bound (for the primary objective) on the larger ones, we can conclude that the obtained solutions are of very high quality.

The results also provide valuable insights regarding the potential benefits of ridesharing for our case study. Through ridesharing, the number of cars on the road can be significantly reduced by 64 to 74% compared to the situation without ridesharing, thus leading to reduced congestion. Furthermore, the total distance driven is reduced by more than 60% in the largest test instances, thus reducing the overall carbon footprint. Finally, the results demonstrate the benefits of having flexible and multiple candidate pickup locations for the passengers, where having flexible pickup locations increased the number of passengers serviced by around 6% and reduced the total travel time by more than 12% on average over all 45 test instances.

Based on the above, we think that the ALNS heuristic proposed in this paper can be a valuable tool in ridesharing systems, both in our case study and beyond. Moreover, our analyses may provide important insights about the potential benefits of ridesharing, which can be of high value, e.g., in policy-making in the Bergen region.

CRediT authorship contribution statement

Jacob Nitter: Writing – review & editing, Visualization, Validation, Software, Methodology, Formal analysis, Data curation. **Shusheng Yang:** Writing – review & editing, Validation, Software, Methodology, Investigation, Formal analysis, Data curation. **Kjetil Fagerholt:** Writing – original draft, Supervision, Methodology, Conceptualization. **Andreas Breivik Ormevik:** Writing – review & editing, Supervision, Data curation, Conceptualization.

Data availability

Data will be made available on request.

Acknowledgments

The authors would like to thank Telia and their Crowd Insights service for providing us with valuable trip data so that we could generate realistic test instances for our case study. The suggestions from the two anonymous reviewers also helped us improve the manuscript and are highly appreciated.

References

Agatz, N., Erera, A., Savelsbergh, M., Wang, X., 2012. Optimization for dynamic ride-sharing: A review. European J. Oper. Res. 223 (2), 295–303.

Archetti, C., Savelsbergh, M., Speranza, M.G., 2016. The vehicle routing problem with occasional drivers. European J. Oper. Res. 254 (2), 472–480.

Auad-Perez, R., Hentenryck, P.V., 2022. Ridesharing and fleet sizing for on-demand multimodal transit systems. Transp. Res. C 138, 103594.

Baldacci, R., Maniezzo, V., Mingozzi, A., 2004. An exact method for the car pooling problem based on Lagrangean column generation. Oper. Res. 52 (3), 422–439.

Bruck, B.P., Incerti, V., Iori, M., Vignoli, M., 2017. Minimizing CO2 emissions in a practical daily carpooling problem. Comput. Oper. Res. 81, 40–50.

Desrosiers, J., Dumas, Y., Solomon, M.M., Soumis, F., 1995. Time constrained routing and scheduling. Handbooks Oper. Res. Management Sci. 8, 35–139.

Dragomir, A.G., Van Woensel, T., Doerner, K.F., 2022. The pickup and delivery problem with alternative locations and overlapping time windows. Comput. Oper. Res. 143, 105758.

Fielbaum, A., Bai, X., Alonso-Mora, J., 2021. On-demand ridesharing with optimized pick-up and drop-off walking locations. Transp. Res. C 126, 103061.

Ghandeharioun, Z., Kouvelas, A., 2023. Real-time ridesharing operations for on-demand capacitated systems considering dynamic travel time information. Transp. Res. C 151, 104115.

He, P., Jin, J.G., Schulte, F., Trépanier, M., 2023. Optimizing first-mile ridesharing services to intercity transit hubs. Transp. Res. C 150, 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. Transp. Res. B 111, 395–421.

Homsi, G., Martinelli, R., Vidal, T., Fagerholt, K., 2020. Industrial and tramp ship routing problems: Closing the gap for real-scale instances. European J. Oper. Res. 283 (3), 972–990.

Hou, L., Li, D., Zhang, D., 2018. Ride-matching and routing optimisation: Models and a large neighbourhood search heuristic. Transp. Res. Part E: Logist. Transp. Rev. 118, 143–162.

Hsieh, F.-S., 2020. A comparative study of several metaheuristic algorithms to optimize monetary incentive in ridesharing systems. ISPRS Int. J. Geo-Inf. 9 (10).

Kaan, L., Olinick, E.V., 2013. The vanpool assignment problem: Optimization models and solution algorithms. Comput. Ind. Eng. 66 (1), 24–40.

Li, T., Xu, M., Sun, H., Xiong, J., Dou, X., 2023. Stochastic ridesharing equilibrium problem with compensation optimization. Transp. Res. Part E: Logist. Transp. Rev. 170, 102999.

Lin, Q., Xu, W., Chen, M., Lin, X., 2019. A probabilistic approach for demand-aware ride-sharing optimization. In: Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing. Association for Computing Machinery, New York, NY, USA, pp. 141–150, URL https://doi.org/10.1145/3323679.3326512.

Parragh, S.N., Doerner, K.F., Hartl, R.F., 2008. A survey on pickup and delivery problems. J. für Betriebswirtschaft 58 (2), 81–117.

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 Trans. Intell. Transp. Syst. 16 (5), 2587–2598. http://dx.doi.org/10.1109/TITS.2015.2413453.

Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. 40 (4), 455–472.

Savelsbergh, M.W., Ulmer, M.W., 2022. Challenges and opportunities in crowdsourced delivery planning and operations. 4OR 20 (1), 1–21.

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

Stiglic, M., Agatz, N., Savelsbergh, M., Gradisar, M., 2015. The benefits of meeting points in ride-sharing systems. Transp. Res. B 82, 36–53.

Sun, Y., Chen, Z.-L., Zhang, L., 2020. Nonprofit peer-to-peer ridesharing optimization. Transp. Res. Part E: Logist. Transp. Rev. 142, 102053.

Ulsrud, K.P., Vandvik, A.H., Ormevik, A.B., Fagerholt, K., Meisel, F., 2022. A time-dependent vessel routing problem with speed optimization. European J. Oper. Res. 303 (2), 891–907.

Zheng, M., Pantuso, G., 2023. Trading off costs and service rates in a first-mile ride-sharing service. Transp. Res. C 150, 104099.