Transportation Research Part E 181 (2024) 103394

Contents lists available at ScienceDirect

Transportation Research Part E

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

A Branch-and-Cut algorithm for the dial-a-ride problem with incompatible customer types

Arne Schulz [*,1], Christian Pfeiffer[1]

Universität Hamburg, Institute of Operations Management, Moorweidenstraße 18, 20148 Hamburg, Germany

ARTICLE INFO

Dataset link: https://doi.org/10.25592/uhhfdm.10389, https://doi.org/10.25592/uhhfdm.11182

Keywords:
Dial-a-ride problem
Ridepooling
Mixed-integer programming
Branch-and-Cut
Valid inequalities

ABSTRACT

Ridepooling is expected to become more and more important in people's mobility. At the same time, autonomous vehicles are expected to be widely used in such applications in the future. While the driver as a person of trust currently provides security, passengers are alone with strangers in an autonomous ridepooling vehicle. In the paper, we address this aspect by including incompatibilities between customers, meaning that incompatible customers cannot share the vehicle at the same time. Although incompatibilities can be defined in any way by the provider, they can be set, for example, if a customer feels unsafe in sharing the vehicle with customers of a certain type. As other compatible customers can replace the driver as a person of trust, we present a second setting where all customers are allowed to share the vehicle at the same time, but customers are never alone in the vehicle with incompatible customers. We present Branch-and-Cut algorithms for both problem settings including methods to improve the search with respect to the incompatibilities. We show in a computational study that the introduced methods improve the search significantly and evaluate the Branch-and-Cut algorithms on realistic instances with up to 120 customers and 10 vehicles. Moreover, our results suggest that it can be a beneficial option for the provider to allow their customers to participate in the decision process by indicating incompatibilities for types of customers, meaning that they do not want to share a vehicle with a customer of an incompatible type.

1. Introduction

In recent years, ridepooling services have been implemented in several cities (Kostorz et al., 2021) to reduce traffic, the number of vehicles, and pollution. In rural areas, which often suffer from a limited public transport system, ridepooling services have, moreover, the potential to improve accessibility to mobility services (von Mörner, 2019). However, ridepooling services typically operate with small vehicles with up to six or eight passenger seats. Thus, due to driver costs, the provider has to generate a high occupancy rate to be profitable. Therefore, ridepooling providers aim at using autonomous vehicles in the near future. As an example, MOIA, Europe's largest ridepooling provider (Kostorz et al., 2021), targets to use autonomous vehicles in Hamburg from 2025 on (MOIA, 2022). However, in an autonomous vehicle passengers are alone with the other passengers without the driver as a person of trust providing security.

Although several papers have investigated models for dial-a-ride problems (DARP) with autonomous vehicles (compare Section 2), these models focus for example on the fact that autonomous vehicles are not required to take a break (Bongiovanni, 2020). They usually do not consider that customers might feel unsafe with strangers in a driverless vehicle. In the study by Dolins et al.

* Corresponding author.
E-mail addresses: arne.schulz@uni-hamburg.de (A. Schulz), christian.pfeiffer@uni-hamburg.de (C. Pfeiffer).
[1] Both authors worked together on all parts of the paper.

https://doi.org/10.1016/j.tre.2023.103394
Received 20 September 2023; Received in revised form 12 December 2023; Accepted 13 December 2023
Available online 21 December 2023

(2021), customers had significant concerns about their co-passengers in automated vehicles. Sanguinetti et al. (2019) suggest among others to use vehicles with large windows, a human administrator who is able to observe the vehicle remotely, good lighting such that passengers can see each other, and an emergency button within reach of each passenger to improve safety.

In the paper at hand, we propose a model formulation for a static DARP including customer types. Our model ensures that customers do not share a vehicle with customers of a certain type if they specified upfront that they do not want to be in the vehicle with customers of that type. We call these customers incompatible customers. Our contribution is fourfold: Relating to methodology, (1) we propose a model variant where customers do not share the vehicle with incompatible customers in general and (2) a variant where customers are generally allowed to share a vehicle with all other customers, but they are never alone in a vehicle with incompatible customers. As customers are never alone in a vehicle with incompatible customers, there is at least one other compatible customer who replaces the driver as a person of trust. Additionally, this second variant is especially interesting from an algorithmic point of view due to the handling of simultaneous actions. (3) We present several methods to improve the search within a Branch-and-Cut algorithm for both variants and (4) answer the research question "*What is the effect of allowing customers to declare incompatibilities with other customers?*" by deriving empirically the managerial insight that considering incompatibilities between customers, i.e. exclude solutions where these customers share a ride, can be an interesting option for the provider. We clearly state that the aim of this paper is to evaluate the effect of incompatibilities. It is the provider's decision to determine which categories customers are allowed to declare incompatibilities on such that no one is discriminated. The examples we use in this paper are only based on the aspects of trust and security resulting from the missing driver in an autonomous vehicle. The purpose of these examples is only to illustrate the two considered problem variants.

The paper is structured as follows: In Section 2, we discuss the relevant literature. In Section 3, we describe the problem in detail and present a mixed-integer program for the strict problem setting without any sharing of incompatible customers. Afterwards, we present a Branch-and-Cut approach for this problem setting in Section 4 before we expand the approach to the case where incompatible customers are allowed to share a vehicle if at least one compatible customer uses the vehicle at the same time (Section 5). In the following Section 6, our approaches are evaluated in an extensive computational study. The paper closes with a conclusion in Section 7.

2. Literature review

The literature review first considers papers focusing on DARPs with autonomous vehicles as our main application. Second, we consider literature with a focus on incompatibilities. Finally, we briefly review Branch-and-Cut algorithms for the DARP.

Ridepooling systems with autonomous vehicles have been investigated in rural areas (Johnsen and Meisel, 2022) as well as in large cities like Munich (Zwick et al., 2021). Johnsen and Meisel (2022) investigated a DARP with interrelated trips and autonomous vehicles. Trips are interrelated if it is not allowed to accept only a subset of the trips, i.e. either all the interrelated trips are accepted or none of them are. Examples are that two people start at different pickup locations and want to meet at a common delivery location or a customer who has a return trip. A mixed-integer programming formulation and an adaptive variable neighborhood search heuristic were presented for the problem setting.

Bongiovanni et al. (2019b) introduced the electric autonomous DARP. They presented a two-index as well as a three-index formulation for the problem setting. Their Branch-and-Cut algorithm comprises constraints for charging beside general DARP constraints. Moreover, one part of the objective function minimizes excess user ride time which is similar to our proceeding. In a follow-up paper, a two-phase heuristic for the dynamic problem version containing an insertion step and a learning large neighborhood search algorithm for the reoptimization of vehicle plans and schedules were presented (Bongiovanni et al., 2019a). Su et al. (2023) developed a deterministic annealing local search metaheuristic for the electric autonomous DARP, which was integrated by the same authors into a column generation approach (Su et al., 2022). Pimenta et al. (2017) investigated a DARP with autonomous electric vehicles in a closed industrial site. Alonso-Mora et al. (2017) used historical data to determine the tours of autonomous taxis in a dynamic setting.

Autonomous vehicles were also investigated in Hasan and van Hentenryck (2021). Here the authors remark that their problem is a special case of the general DARP and that the autonomous vehicles make a centralized depot possible since no individual driver's start or end depots need to be considered. In our paper, we already assume a centralized provider that employs a vehicle fleet, i.e. we already have one common depot. In Hyland and Mahmassani (2018) and Hyland and Mahmassani (2020) dynamic autonomous DARPs were considered. The authors state that an autonomous vehicle fleet allows for a centralized planning approach since the central planner has perfect information about the vehicles and can give them direct orders on how to drive. This is in contrast to systems with private drivers (such as Uber) where the service provider has less control over the vehicles. However, in our ridepooling case, we assume a ridepooling provider like MOIA that has employed drivers and thus already has perfect information and control over the vehicle fleet and drivers. Therefore, in this aspect, using autonomous vehicles does not change anything in our problem setting. In the study by Zhang et al. (2022), a stochastic autonomous DARP was presented. The authors assume the autonomous vehicles to be most likely electric and thus explicitly take into account the need for battery recharging.

The paper by Li et al. (2020) focused on the assignment of customers to autonomous vehicles. They presented a method to determine the minimum number of required autonomous vehicles by classifying the customers into clusters. Liang et al. (2020) maximized the profit for a DARP with autonomous vehicles under dynamic travel times due to traffic congestion, which led to a non-linear integer programming model.

Manerba and Mansini (2015) introduced the multi-vehicle traveling purchaser problem with pairwise incompatibility constraints. Incompatible products are not allowed to be transported in the same vehicle. They presented a Branch-and-Cut algorithm for the

problem setting. In a follow-up paper, Gendreau et al. (2016) considered the multi-vehicle traveling purchaser problem with pairwise incompatibility constraints and unitary demands. They presented a column generation approach. Lokhandwala and Cai (2020) investigated a ridepooling system in which some customers allow pooling while others do not. The model presented in our paper can also be applied to a system with two customer types, one allowing pooling and one not. While the first type's customers are compatible with all customers of the same type, the customers who do not allow pooling are incompatible to all other customers, i.e. to customers of both types. Factorovich et al. (2020) introduced the pickup and delivery problem with incompatibility constraints for a single vehicle. They introduced different model formulations as well as a Branch-and-Cut algorithm with several new valid inequalities. Moreover, they mentioned that the problem is still NP-hard, as instances in which all goods are incompatible to all other goods lead to a tour where each delivery location follows directly after the corresponding pickup location, i.e. a traveling salesman instance. Applications are named in the transportation of goods, for example hazardous materials.

In the paper by Yan and Chen (2011), a many-to-many car pooling problem is investigated in which customers are assigned to different types regarding gender (male and female) and smoking (smoking and non-smoking). Customers are allowed to express preferences regarding the car's passengers. The difference to our presented problem is that in the car pooling problem, the drivers also have a transportation request that needs to be fulfilled. Moreover, Yan and Chen (2011) considered the two explicit dimensions gender and smoking, but the model can be extended with further characteristics by adding additional networks and sets of variables and constraints. We allow for any number of characteristics since they are only implicitly considered in the input data to our model. Finally, we also investigate the case where additional compatible passengers can suppress incompatibilities. The orienteering problem with incompatible nodes was studied in Palomo-Martínez et al. (2017) and Lu et al. (2018). If two nodes are incompatible, only one of them can be selected in the tour. Two different types of incompatibility have been considered by Colombi et al. (2017). In the setting of a rural postman problem, the approach includes strongly incompatible nodes which cannot be selected together and weakly incompatible nodes which can be selected together but result in penalty costs. Bernardino and Paias (2022) investigated the family traveling salesman problem with incompatibility constraints where incompatible families are not allowed to be visited in the same route.

Baldacci et al. (2004) investigated the to-work car pooling problem where a subset of employees is willing to drive to work with their own car and pick up other co-workers along the way. This problem can be modeled as a special case for the standard DARP if the vehicles are homogeneous. In Ma et al. (2021), a car pooling problem is presented where certain users are not predetermined to be drivers or passengers. Instead, there is a degree of freedom to specify which of the users should drive. However, neither paper considers potential incompatibilities between users. A survey about influencing factors for car pooling systems was conducted in Correia and Viegas (2011). According to them, an important limiting factor was the psychological barrier to share the vehicle with strangers. One way to combat this issue is to form a group where the same users repeatedly car pool together. This group is then monitored and ideally it leads to a gain of trust among the members over time. Therein lies a difference to the DARP we consider, as in an urban ridepooling context we cannot assume that requests repeat on a schedule. Thus, each time a user uses the system, they most likely interact with different other users which makes forming permanent groups difficult. This reinforces the need for an explicit inclusion of incompatibilities such that users feel safe using the service.

A many-to-many car pooling problem was investigated in Yan et al. (2011). Here, both origins and destinations differ among the users. This makes the problem closer to a traditional DARP. A car pooling problem including user preferences was presented by Hsieh and Zhan (2018). The users in the system can place bids according to their preferences and constraints. The preferences are not further specified, but it would likely be possible to include personal information and thus influence the final matching if e.g. a user does not want to share the vehicle with users of a certain type. Huang et al. (2015) presented a car pooling problem where user scores are included in the objective. This is a similar direction to our research, as it includes the social component of individuals and their effect on other passengers. However, since the scores are only included in the objective, it is a soft constraint unlike our considered incompatibilities which prevent users from traveling together in any feasible solution.

Ridesharing features a problem similar to our proposed case, as strangers share a vehicle with each other without the presence of a neutral employed driver by the service provider. Thus, trust and compatibility between users is important. However, in ridesharing systems there is an inherent difference between the driver and the passenger(s). For ridesharing systems, trust between users has been investigated in several articles. Cheng et al. (2020) presented a qualitative study that examined different factors that influence users' trust. A more quantitative approach is utilized by Cheng et al. (2023). Additional constraints for the matching of users based on e.g. gender have also been considered (Agatz et al., 2012).

There are some similarities of our investigated problem to the prisoner transport problem (PTP) first proposed by Christiaens et al. (2020) and also investigated by Ferone et al. (2023). In the PTP, prisoners are transported and the prisoners can also have incompatibilities with each other which means they cannot travel together. However, the PTP differs from our problem by considering different compartments in each vehicle for the transport and a distinction whether prisoners are not allowed to travel in the same compartment or in the same vehicle. Additionally, the case presented in Section 5, where compatible passengers alleviate incompatibilities, is not present in the PTP.

Our approach is a variant of the classic DARP. Early exact approaches for the classic DARP contain the works by Psaraftis (1980, 1983). Branch-and-Cut algorithms for the classic DARP were developed by Cordeau (2006) and Ropke et al. (2007). Ropke and Cordeau (2009) implemented a Branch-and-Cut-and-Price algorithm. They introduced two variations of the pricing problem and used several of the valid inequalities introduced in the earlier papers. Some of the earlier results were also improved by the Branch-and-Cut-and-Price algorithm by Gschwind and Irnich (2015). A new variant was developed recently by Rist and Forbes (2021). They used specific tour fragments (partial tours) in their Branch-and-Cut approach. Recently, Schulz and Pfeiffer (2024) used partial fixed tours in the nodes of the Branch-and-Cut tree to fix variables and improve lower bounds on arrival times at customer locations.

Schulz and Pfeiffer (2024) also allow customer rejections to ensure feasibility (time windows). We allow customer rejections in the paper at hand for the same reason as well. In earlier works, Ropke and Pisinger (2006) put requests which go unserved into a request bank and penalized them in the objective function. Parragh et al. (2015) rejected customers if their trip is not profitable. In the selective DARP, the objective is to maximize the number of served requests (Riedler and Raidl, 2018 and Rist and Forbes, 2022).

We refer to the surveys by Parragh et al. (2008) and Ho et al. (2018) for an extensive discussion on the literature of dial-a-ride problems.

3. Problem description

We investigate a DARP with n customers each requesting a trip from a pickup to a delivery location. We have K tours starting and ending at a common depot. All K vehicles are assumed to be homogeneous with a maximum passenger capacity Q. Each of the customer requests has a number of passengers q_i. Furthermore, all pickup and delivery locations have to be visited within a time window $[e_i, l_i]$. The time window for the pickup node can be interpreted as follows: e_i is the earliest point in time the customer wants to start the ride, while our offer would not be attractive enough (e.g. in comparison to other modes of transportation) if the ride started later than l_i. Distances between each pair of locations i and j are given, denoted by t_{ij}, and fulfill the triangle inequality. Each customer has a customer type. However, the customer type is not included in the model directly. Instead, the model contains a parameter c_{ij} which is 1 if customers i and j are incompatible, i.e. cannot be transported simultaneously and 0 otherwise. Thus, the provider can define different customer types such that each customer is assigned to one type. Then, the provider can determine which customer types are allowed to share a vehicle or the provider lets the customers specify with customers of which type they are willing to share their ride. Afterwards, the provider can set parameters c_{ij} accordingly. The complete notation is summarized in the following:

sets:	
$P = \{1, \dots, n\}$	pickup locations
$D = \{n+1, \dots, 2n\}$	delivery locations
$i, j, h, k \in I$	locations, $I = \{0, 2n+1\} \cup P \cup D$, where $\{0, 2n+1\}$ represent the depot
parameters:	
c_{ij}	= 1 if customer j is incompatible from the point of view of customer i, 0 otherwise (in strict case $c_{ij} = c_{ji}$)
t_{ij}	travel time between locations i and j
q_i	demand at location i: $q_i > 0$, $q_{n+i} = -q_i \quad \forall i \in P$
e_i	start of time window for location $i \in P \cup D$
l_i	end of time window for location $i \in P \cup D$
Q	capacity of each vehicle
n	number of requests
K	number of tours/vehicles
M_{ij}	sufficiently large constant
Φ	penalty factor for rejecting a customer
variables:	
B_i	departure time from location i
X_{ij}	= 1 if a vehicle drives from location i directly to location j, 0 otherwise
Y_i	= 1 if the customer of location i is rejected, 0 otherwise
Z_{ij}	= 1 if customer $i \in P$ is in the vehicle when the vehicle leaves location $j \in I \backslash \{2n+1\}$, 0 otherwise

We present the model formulation in the following. The objective function comprises two parts and addresses the practical setting of a ridepooling provider being in competition with different modes of transportation like private cars, public transport, bicycles, e-scooters or car sharing. First, the number of accepted customers is maximized due to the sufficiently high penalty factor Φ. Second, the customer inconvenience, expressed by the relative detour, is minimized. The second part of the objective is an adapted variant of the one introduced in Pfeiffer and Schulz (2022a,b). We calculate the relative detour of a customer by computing the difference between the arrival time at the customer's delivery location and the earliest possible arrival time at the delivery location, i.e. $B_{n+i} - e_i - t_{i,n+i}$ for a customer $i \in P$. This gives us a nominal detour. Finally, we get the relative detour by dividing the nominal detour by the direct travel time between the customer's pickup and delivery location. This means that the customer has a relative detour of 0 if the vehicle reaches the customer's pickup location at the beginning of the time window and drives directly to the corresponding delivery location. Direct delivery can be understood as a benchmark for the customer, which can be reached by using a private car or a taxi.

Due to the time windows, it might be that some customer requests cannot be served by the same vehicle. Thus, it might be possible that not all customers can be served, as the number of vehicles is restricted. Therefore, we allow customer rejections like

in Schulz and Pfeiffer (2024) to ensure that a feasible solution always exists such that we can evaluate different degrees of customer incompatibility. However, it should always be superior to serve as many customers as possible. Therefore, we set the penalty factor Φ to $\sum_{i \in P} q_i \frac{l_{n+i} - e_i - t_{i,n+i}}{t_{i,n+i}} + 1$, which is strictly larger than the highest value the sum of relative detours of all customers can reach. Thus, our model finds a solution in which as many customers as possible are served and identifies the customers who should not be served such that the remaining customers are served in the best possible way.

As the vehicle can start its tour sufficiently early, we assume that each pickup location can be reached on time at the beginning of the time window, i.e. $e_i \geq t_{0i}$ for every pickup location. Furthermore, we assume that a vehicle never has to wait at a delivery location, which means that $e_{n+i} = e_i + t_{i,n+i}$. We can further assume that $l_{n+i} \geq l_i + t_{i,n+i}$, since otherwise the pickup location's time window can be tightened.

The presented model formulation is based on the 2-index formulation by Ropke et al. (2007). $\mathcal{S}$ is the set of all sets S with $0 \in S$, $2n+1 \notin S$, and at least one customer request i where the delivery vertex is in S, but the pickup vertex is not, i.e., $\mathcal{S} = \{S : 0 \in S \wedge 2n+1 \notin S \wedge \exists i : (i \notin S \wedge n+i \in S)\}$. The model formulation follows as:

$$\min \quad \sum_{i \in P} q_i \frac{B_{n+i} - e_i - t_{i,n+i}}{t_{i,n+i}} + \sum_{i \in P} \Phi \cdot q_i \cdot Y_i \tag{1}$$

$$\sum_{i \in I} X_{ij} + Y_j = 1 \qquad \forall j \in P \cup D \tag{2}$$

$$\sum_{j \in I} X_{ij} + Y_i = 1 \qquad \forall i \in P \cup D \tag{3}$$

$$\sum_{j \in P} X_{0j} \leq K \tag{4}$$

$$\sum_{i,j \in S} X_{ij} \leq |S| - 2 \qquad \forall S \in \mathcal{S} \tag{5}$$

$$B_i + t_{ij} - M_{ij}(1 - X_{ij}) \leq B_j \qquad \forall i \in P \cup D, j \in P \cup D \tag{6}$$

$$Z_{ik} \leq Z_{ij} + (1 - X_{kj}) \qquad \forall i \in P, k \in P \cup D, j \in P \cup D \backslash \{i, n+i\} \tag{7}$$

$$Z_{ii} = 1 \qquad \forall i \in P \tag{8}$$

$$Z_{i,n+i} = 0 \qquad \forall i \in P \tag{9}$$

$$Z_{ik} + Z_{jk} \leq 1 \qquad \forall i, j \in P : c_{ij} = 1 \wedge i < j, k \in P \cup D \tag{10}$$

$$\sum_{i \in P} q_i \cdot Z_{ij} \leq Q \qquad \forall j \in P \cup D \tag{11}$$

$$Y_i = Y_{n+i} \qquad \forall i \in P \tag{12}$$

$$e_i \leq B_i \leq l_i \qquad \forall i \in P \cup D \tag{13}$$

$$X_{ij} \in \{0, 1\} \qquad \forall i \in I, j \in I \tag{14}$$

$$Y_i \geq 0 \qquad \forall i \in P \cup D \tag{15}$$

$$Z_{ij} \geq 0 \qquad \forall i \in P, j \in P \cup D \tag{16}$$

Objective function (1) minimizes the number of rejected requests, which is weighted sufficiently high (Φ) such that it is optimal to serve as many customers as possible. Then, the relative detours of the accepted customers weighted with the number of passengers in the request are minimized. The relative detour is always non-negative, as $e_i + t_{i,n+i}$ is a lower bound for B_{n+i}. Constraints (2) and (3) ensure each pickup and delivery location is visited and left exactly once if the customer is served. If the customer is not served, Y_i takes the value 1. Constraint (4) ensures that at most K vehicles leave the depot. Constraints (5) take care that precedence constraints are fulfilled and that a customer's pickup and delivery location are located in the same tour. Due to Constraints (6), time variables are set correctly. M_{ij} can be set to $\max(0, l_i + t_{ij} - e_j)$.

The following block of constraints, (7)–(10), ensures that incompatible customers do not share a vehicle at the same time. While Constraints (7) take care that customers stay in the vehicle when the next location is reached, the corresponding customer enters the vehicle at its pickup location due to Constraints (8) and leaves it again at its delivery location because of Constraints (9). However, incompatible customers cannot stay in a vehicle at the same time, which is ensured by Constraints (10). As Z_{ij} denotes whether customer i is in the vehicle when location j is left, Constraints (11) enforce that the vehicle capacity is never exceeded. Because of Constraints (12) a customer request is either accepted or rejected, i.e. if the customer is rejected, then its pickup as well as its delivery location are rejected. The variables for the delivery locations Y_i for all $i \in D$ can be omitted from the model and replaced by the corresponding pickup variable in an actual implementation.

The final block of constraints, (13)–(16), restricts the variable domains. Due to Constraints (2) and (3), variables Y_i will always take binary values in an integer feasible solution and do not need to be further constrained. Because of Constraints (7) and (8) a customer is in the vehicle at its pickup location and at the following locations until it leaves the vehicle again at the delivery location due to Constraints (9) such that Z_{ij} is enforced to be 1 between the customer's pickup and delivery location. At all other

locations the variables need not to be controlled, but it is beneficial if they are 0 because of incompatibilities (Constraints (10)) and capacity Constraints (11).

Like in Ropke et al. (2007), Constraints (5) are not all added at the start but instead are checked every time an integer feasible solution is found. We do this by solving a maximum flow problem for every customer to check whether the pickup and delivery location are visited in the right order and in the same tour. Customers who are rejected in the current solution do not need to be checked.

4. Branch-and-Cut approach

We present the Branch-and-Cut procedure in this section. We introduce several classes of new valid inequalities and procedures to improve the search focusing on incompatible customers while the fixed-path procedure is adapted from the Branch-and-Cut approach by Schulz and Pfeiffer (2024). Moreover, we add valid inequalities from the literature (including those previously introduced in Cordeau, 2006, Ropke et al., 2007 and Lysgaard, 2006) as introduced and mentioned in their papers. This means that we separate the following inequalities in each fractional node: The lifted subtour elimination constraints, the generalized order constraints, the strengthened capacity constraints, the reachability constraints, and the fork constraints. The inequalities are all separated heuristically and our heuristics follow the described implementation in Ropke et al. (2007). Branching decisions are left at the CPLEX default values. This implies that we use a best bound search, i.e. the node with the smallest objective value in its linear programming (LP) relaxation is chosen when evaluating which branch should be examined next.

One change compared to the literature is the removal of the preprocessing inequalities in Section 3.2 of Schulz and Pfeiffer (2024): The matching constraints cannot be used in our problem setting since the constraints rely on the idea that we can identify situations where it is optimal to visit a customer in between two other customers where no customer is worse off. However, it is possible that visiting another customer beforehand that is incompatible with the customer we pushed in between is even better and therefore the constraint could cut off the optimal solution. Additionally, whenever a sequence is checked for feasibility (e.g. in the reachability or fork constraints), the incompatibilities between customers can also be examined to declare certain sequences infeasible. This ensures that these cutting planes retain their effectiveness.

The section is structured as follows: First, we fix additional arc variables (Section 4.1) and add new valid inequalities (Section 4.2) in the preprocessing. Then, we briefly describe the fixed-path procedure by Schulz and Pfeiffer (2024) (Section 4.3). In the main part of the section, we present methods to improve the search in Branch-and-Cut nodes (Section 4.4).

4.1. Fixing additional arc variables

Incompatibilities of customers lead to further sequence variables which cannot be selected, as the corresponding arc leads to an infeasible solution. If customers i and j are incompatible ($c_{ij} = c_{ji} = 1$), a vehicle cannot visit the pickup location of one of them and afterwards a location of the other one. This means that variables X_{ij}, $X_{i,n+j}$, X_{ji}, and $X_{j,n+i}$ can be fixed to 0. Moreover, it is not possible to visit the delivery locations of both customers in a row, as due to the precedence constraints the pickup location of the customer visited second has to be visited before the delivery location of the other customer. Thus, $X_{n+i,n+j}$ and $X_{n+j,n+i}$ can also be fixed to 0.

4.2. Valid inequalities

If customers i and j are incompatible and customer i is in the vehicle when location k is left, it is not feasible to visit any of the locations associated with customer j. Thus,

$$Z_{ik} + X_{kj} + X_{k,n+j} \leq 1 \quad \forall k \in P \cup D \tag{17}$$

is a valid inequality.

Moreover, there are some combinations of the locations of customers i and j which cannot both be part in a sequence of three successively visited locations. If customer i is picked up and the vehicle visits a location k which is not location $n+i$ afterwards, customer i is still in the vehicle when leaving k. Thus, it is infeasible to visit locations j or $n+j$ afterwards if customers i and j are incompatible, which leads to the valid inequality

$$X_{ik} + X_{kj} + X_{k,n+j} \leq 1 \quad \forall k \in P \cup D \backslash \{n+i\}. \tag{18}$$

Analogously, locations j and $n+j$ cannot be visited before location k if location $n+i$ is visited after k and customers i and j are incompatible. Thus,

$$X_{k,n+i} + X_{jk} + X_{n+j,k} \leq 1 \quad \forall k \in P \cup D \backslash \{i\}. \tag{19}$$

is also a valid inequality. The proofs for the correctness of valid Inequalities (17)–(19) can be found in Appendix A.

4.3. Fixed-path procedure

The Branch-and-Cut procedure relies on the notion of a fixed path (Schulz and Pfeiffer, 2024). These fixed paths are created whenever an arc variable X_{ij} is fixed to 1 in one of the nodes in the Branch-and-Cut tree. A fixed path R_{ij} is a path that starts at vertex i, goes through one or multiple fixed arcs, and then arrives at vertex j, i.e. no variable of an ingoing arc of i and no variable of an outgoing arc of j are fixed to 1 while all arc variables in between are. For each path R_{ij}, V_{ij} describes the ordered sequence of vertices in the path. In the root node, the procedure starts with $2n$ paths R_{ii} (one for each vertex $i \in P \cup D$). Every time an arc (i, j) is fixed to 1 when creating a new branch, the two corresponding paths R_{ki} and R_{jh} are merged to form the new path R_{kh}. Precedence relations can be utilized to add valid inequalities that improve the bounds for the departure time variables B_i, as the pickup locations of all customers present in the vehicle have to be visited before location k if they are not part of the path R_{kh}. Should there be a precedence relation between two paths R_{ki} and R_{jh}, but the variable for the connecting arc (i, j) has been fixed to 0, a shortest detour to reach vertex j can be calculated to further improve bounds for variables B_i. For a more in-depth description and further algorithmic steps, we refer to Schulz and Pfeiffer (2024). In the following, we explain how we can further enhance the approach by including incompatibilities between customers.

4.4. Implementation of incompatibilities

We have $n \cdot 2n \cdot 2(n-1) = 4n^2(n-1)$ Constraints in (7). Although this is clearly polynomial in the number of requests, a reasonable size of 100 customers already leads to almost 4,000,000 constraints of type (7). Thus, it is not useful to add all these constraints upfront. Constraints (7) are unnecessary if X_{kj} is 0. So, we do not need to consider these constraints if we fixed X_{kj} to 0 in preprocessing.

We show in the following (Section 4.4.1) how to ensure that we never have a fixed path which is infeasible due to incompatibilities. Moreover, we implement infeasible path inequalities to cut off integer feasible LP solutions violating incompatibility constraints (Section 4.4.2). Thus, it is actually not necessary to add any of the Constraints (7)–(10). We therefore also omit Constraints (17) since they require the Z_{ij} variables.

If we omit Constraints (7)–(10), Constraints (11) do not work correctly any more. Thus, we replace Constraints (11) and (16) by the classic DARP capacity constraints (see Ropke et al., 2007)

$$Q_i + q_j - Q(1 - X_{ij}) \leq Q_j \quad \forall i, j \in P \cup D \tag{20}$$

and

$$\max\{0, q_i\} \leq Q_i \leq \min\{Q, Q + q_i\} \quad \forall i \in P \cup D, \tag{21}$$

where Q_i is the number of passengers in the vehicle after leaving location i.

4.4.1. Incompatibilities by merging

Let R_{hk} and R_{lm} be two paths and i and j two incompatible customers. Let V_{hk} (V_{lm}) be the ordered set of all locations visited in path R_{hk} (R_{lm}). We can fix the variable X_{kl} to 0 if either

$$i \in V_{hk} \wedge n + i \notin V_{hk} \wedge n + i \notin V_{lm} \wedge (j \in V_{lm} \vee n + j \in V_{lm}) \text{ or} \tag{22}$$

$$(i \in V_{hk} \vee n + i \in V_{hk}) \wedge j \notin V_{hk} \wedge j \notin V_{lm} \wedge n + j \in V_{lm}. \tag{23}$$

Conditions (22) cover situations where customer i enters in the first path and does not leave until meeting the pickup or delivery of j in the second path. Conditions (23) cover situations where the delivery vertex $n + j$ is visited in the second path and therefore j would have to be in the vehicle before entering the first path (and thus meeting i). Note that the roles of i and j also have to be swapped in Conditions (22) and (23). We refer to the procedure of checking these two constraints and potentially fixing the arc to 0 as *after merge fixing* or *AMF* for short.

The following theorem shows that Conditions (22)–(23) ensure that no path R_{hk} contains an incompatibility during our Branch-and-Cut procedure, i.e. whenever two paths are merged, it is guaranteed that there is no incompatibility within the new path if the two merged paths do not contain an incompatibility. As the procedure begins with $2n$ single location paths (one for each location $i \in P \cup D$), which are feasible, we never get a path with incompatible customers.

Theorem 1. *Conditions (22)–(23) (also with swapped roles) ensure together with precedence constraints that no two paths R_{hk} and R_{lm} are merged such that incompatible customers are simultaneously in the vehicle, i.e. we never have a path R_{hm} of fixed arcs in our Branch-and-Cut tree which is infeasible due to incompatibility.*

We prove the theorem by a case-by-case analysis. The proof can be found in Appendix B.

4.4.2. Integer LP solution

If we omit Constraints (7)–(10), we might have a situation in which we obtain an integer feasible LP solution with incompatible customers who are served simultaneously by the same vehicle. This can happen in two different cases

1. $i \to ... \to j \to ... \to n+j \to ... \to n+i$
2. $i \to ... \to j \to ... \to n+i \to ... \to n+j$

In the first case, we have the infeasible sequences $i \to ... \to j$ and $n+j \to ... \to n+i$. In the second case, the sequences $i \to ... \to j$, $j \to ... \to n+i$, and $n+i \to ... \to n+j$ are infeasible.

Let $R^X_{hk} = \{(i,j) : i \in V_{hk} \wedge j \in V_{hk} \wedge X_{ij} = 1\}$, i.e. all pairs of locations in the sequence which are directly connected by a selected arc. We can then add an infeasible path constraint (see Ascheuer et al., 2000): For each of the sequences mentioned above we introduce the following valid inequality in the corresponding node

$$\sum_{(i,j)\in R^X_{hk}} X_{ij} \leq \left|R^X_{hk}\right| - 1, \tag{24}$$

where h is the first and k the last location of the corresponding sequence, to cut off the LP solution violating incompatibility constraints.

5. Incompatible customers sharing with further customers

In this section, we expand our approach to the case where incompatible customers can share a vehicle if at least one compatible customer from the point of view of each of them is present in the vehicle at the same time. As in non-autonomous vehicles the driver is a compatible person, it seems to be that a compatible customer can replace the driver in this function. We denote this use case as the *shared* case while the previous use case is referred to as *strict*. Thus, we have to adapt Constraints (10). Let W_{ik} be a binary variable which is 1 if there is a compatible customer for customer $i \in P$ in the vehicle when it leaves location $k \in P \cup D$. To set W_{ik} correctly, we add the following constraints

$$W_{ik} \leq \sum_{j\in P: c_{ij}=0 \wedge j\neq i} Z_{jk} \quad \forall i \in P, k \in P \cup D. \tag{25}$$

Then, we can replace Constraints (10) by

$$Z_{ik} + Z_{jk} \leq 1 + W_{ik} \quad \forall i,j \in P : c_{ij} = 1, k \in P \cup D. \tag{26}$$

Note that Inequalities (26) exist once for customers i and j and once for the same two customers in reverse order such that Inequalities (26) reduce to Inequalities (10) if at least for one of them no compatible customer is present when leaving location k. Variable ranges for W_{ik} do not need to be restricted because it is sufficient that, in order to set Inequalities (26) correctly, they can be at least 1 if the right side of Inequalities (25) is at least 1 and that they can be 0 otherwise. Note that Inequalities (25) and (26) can be combined by replacing W_{ik} in Inequalities (26) by the sum on the right side of Inequalities (25) (with another index instead of j in the sum).

Moreover, we have to adapt the techniques introduced in Section 4. As customers i and j are now allowed to be in the vehicle simultaneously although they are incompatible (if corresponding further customers are present), an incompatibility of customer i with customer j does not directly imply that j has an incompatibility with i as well. Let us consider the case where i does not want to be alone in the vehicle with j, but for j it is fine to share the ride with i, and we have another customer k who wants to share the ride with j but not with i while j does not want to share the ride with k and i wants to. Thus, $c_{ij} = c_{jk} = c_{ki} = 1$ and $c_{ji} = c_{ik} = c_{kj} = 0$. This means that no pair of these three customers can use the vehicle simultaneously on their own but all three of them can. We need a further appropriate customer l in the vehicle before the second of i, j, and k enters it and another one m in the vehicle before the first of them leaves it again. Consider the sequence

$$P_i \to P_l \to P_j \to P_k \to D_l \to P_m \to ...,$$

where P_a is the pickup location of customer a and D_a their delivery location. This sequence would be feasible although customers i, j, and k are all three in the vehicle between locations D_l and P_m but without a further customer. Thus, we cannot assume incompatibilities to be symmetric in this section.

As customers i and j are now allowed to be in the vehicle simultaneously although they are incompatible, variable fixings in Section 4.1 are not correct in this setting.

In the actual implementation, we proceed as described in Section 4.4 and omit Z_{ij} variables as well as all corresponding constraints. Thus, we can also not adapt valid Inequalities (18) and (19) and have to omit them as well, as we do not know whether compatible customers are in the vehicle without Z variables. Instead, feasibility with regard to the shared incompatibilities is guaranteed through a procedure which is described in further detail in Section 5.2.

If a fixed path R_{kh} is connected to one of the depots, we can check connections to any other path for possible violations of the incompatibility constraints since the other end of the path is effectively fixed. This means that if we have a newly merged path R_{0h}, we can check the sequence V_{0h} followed by V_{lm} for every other path R_{lm} (for which the connecting arc (h,l) is not already fixed to 0). If there arises a conflict in the sequence, the variable X_{hl} can be fixed to 0. The procedure works analogously for paths connected to the end depot.

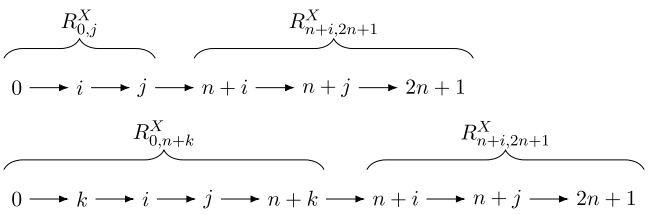

Fig. 1. Two infeasible paths and the respective sets of arcs R^X which need to be cut off.

Furthermore, we can add the following valid inequalities for all pairs of customers i and j with $c_{ij} = 1$ and/or $c_{ji} = 1$ in preprocessing:

$$X_{0i} + X_{ij} \leq 1 \qquad \forall i \in P, j \in P : c_{ij} \vee c_{ji} = 1 \tag{27}$$

$$X_{n+i,n+j} + X_{n+j,2n+1} \leq 1 \qquad \forall i \in P, j \in P : c_{ij} \vee c_{ji} = 1 \tag{28}$$

5.1. Path merging

We can adapt the *AMF* procedure from the strict case to the shared case: Whenever a path R_{hk} is merged, we can use Conditions (22) and (23) to check whether there would be a potential conflict if we visit path R_{lm} right after R_{hk}, i.e. whether there is an incompatibility in the sequence V_{hk} followed by V_{lm} which is not resolved by other customers in the merged path. If there is an incompatibility, we know that in order for the variable X_{kl} to take the value 1, at least one compatible customer has to enter the vehicle beforehand and exit it afterwards. If we can identify a conflict for which there is provably no feasible customer that can be visited, we can fix the arc variable X_{kl} to 0. For this, we go through all pairs of conflicting customers in the two sequences V_{hk} and V_{lm}. Let (i, j) be such a pair with $c_{ij} = 1$. If the conflict is one-sided, i.e. $c_{ji} = 0$, only a single other customer is required for feasibility. Let R^{P_g} (R^{D_g}) be the path containing the pickup (delivery) location of customer g. We check the sequence $R^{P_g} \to R_{hk} \to R_{lm} \to R^{D_g}$ for every customer g that is compatible with i for feasibility of time windows and precedence constraints. If the pair (i, j) is a two-sided conflict, i.e. $c_{ij} = c_{ji} = 1$, we follow the same principle, but we might need to include more paths. In the most simple case, we find a customer g that is compatible with both i and j and then check the sequence $R^{P_g} \to R_{hk} \to R_{lm} \to R^{D_g}$ for feasibility as before. If g is only compatible with i however, another customer f is required that is compatible with j. If f is picked up and dropped off in the same paths as g, we can treat the case as before and only need to check a single sequence. In the most extreme case, the pickup and delivery locations for the requests g and f are in four different paths, however. In such a case, all four possible sequences

$$\begin{aligned}
&R^{P_g} \to R^{P_f} \to R_{hk} \to R_{lm} \to R^{D_g} \to R^{D_f},\\
&R^{P_g} \to R^{P_f} \to R_{hk} \to R_{lm} \to R^{D_f} \to R^{D_g},\\
&R^{P_f} \to R^{P_g} \to R_{hk} \to R_{lm} \to R^{D_g} \to R^{D_f}, \text{ and}\\
&R^{P_f} \to R^{P_g} \to R_{hk} \to R_{lm} \to R^{D_f} \to R^{D_g}
\end{aligned} \tag{29}$$

are checked for feasibility. If at least one of them is feasible, the conflict is resolved. Otherwise, if among all other paths we cannot find customers g (and f) such that the conflict is avoided, the variable X_{kl} can be fixed to 0.

5.2. Integer LP solution

If we obtain an integer solution, we check it for feasibility regarding incompatibilities and potentially add inequalities of type (24). Let i and j be two customers with at least one incompatibility between them (i.e. $c_{ij} + c_{ji} \geq 1$) and let i be the customer that enters the vehicle first. In contrast to Inequality (24), a part of the path between incompatible customers i and j is not necessarily infeasible because compatible customers can enter the vehicle before j and leave it after $n+i$ or $n+j$. We can apply Inequality (24) only to the path between the start depot and j, i.e. $R^X_{0,j}$, if within this path no compatible customer enters the vehicle to prevent the conflict. Correspondingly, we can apply Inequality (24) only to the part of the tour starting with the delivery location of the customer i or j who leaves the vehicle first, i.e. $R^X_{n+i,2n+1}$ or $R^X_{n+j,2n+1}$, if no compatible customer leaves the vehicle afterwards. If none of the two cases occur, the conflict happens after visiting j but before visiting one of the delivery vertices $n+i$ or $n+j$. This happens if the compatible customer k, which prevented the conflict, leaves after j but before $n+i$ and $n+j$. Then, the infeasible path takes the form $R^X_{0,n+k}$. This is visualized in Fig. 1 where the upper path shows the case where no compatible customers are present and the lower path shows the case where a compatible customer k is present. The figure is not exhaustive: Further sequences (e.g. where a compatible customer is only present for the delivery vertices) are possible.

The inequalities to cut off integer solutions can further be tightened as follows: Let i and j again be the pair of incompatible customers and i be the customer that is visited first. Instead of cutting off the sequence $R^X_{0,j}$, we may cut off the shorter sequence $R^X_{i,j}$ if we can prove that there is no other compatible customer that could resolve the incompatibility. There is no such customer l if the sequence $l \to i \to \ldots \to j \to n+l$ is infeasible due to time windows or capacity for every compatible customer l. Although this

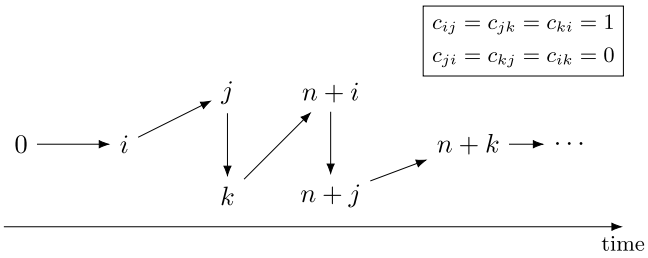

Fig. 2. A tour where the sequences $j \to k$ and $n+i \to n+j$ are ZDCs which causes the tour to be feasible despite the pairwise incompatibilities.

sequence does not prevent the incompatibility on its own, as another customer g needs to enter the vehicle before l leaves, there cannot be such a customer g leading to a feasible sequence due to the triangle inequality. This tightening works analogously for the second half of the path $R^X_{n+i,2n+1}$ and also if the original sequence takes the form $R^X_{0,n+k}$ (both like in Fig. 1). Since this tightening lifts the original lazy constraints, we refer to these constraints as *lazy lifting* constraints.

5.3. Variable fixing in preprocessing

Furthermore, in the preprocessing we can use this idea to fix variables X_{ij} ($X_{n+i,n+j}$) between incompatible customers to 0 if we cannot find another customer k that can be visited before and afterwards according to time windows. In the preprocessing, this condition can also be strengthened by including all the delivery vertices. If none of the sequences

$$\begin{aligned}
&k \to i \to j \to n+i \to n+j \to n+k,\\
&k \to i \to j \to n+i \to n+k \to n+j,\\
&k \to i \to j \to n+j \to n+i \to n+k,\\
&k \to i \to j \to n+j \to n+k \to n+i,\\
&k \to i \to j \to n+k \to n+i \to n+j \text{ or}\\
&k \to i \to j \to n+k \to n+j \to n+i
\end{aligned} \tag{30}$$

is feasible for any k compatible with i or j, the variable X_{ij} can be fixed to 0 (analogously for the variable $X_{n+i,n+j}$). The last two sequences where $n+k$ is the first visited delivery vertex would not be feasible in regard to the incompatibilities on their own, but it could be feasible to visit another compatible customer l between j and $n+k$ that leaves afterwards. We only check these six sequences in order to not have to test these additional cases. If the sequences (30) are all infeasible due to time window or capacity constraints, the arc can definitely be fixed to 0 because an additional customer would not help with those violations (triangle inequality).

5.4. Customers at the same locations

Due to the discretization of potential stops, it is possible for multiple separate customers to share a pickup or delivery location, i.e. $t_{ij} = 0$ for any $i, j \in P \cup D, j \neq n+i$. Normally, this case does not require special attention, however, it becomes relevant in the setting where incompatible customers can share a vehicle if compatible customers are present. If there are multiple actions (pickups and/or deliveries) happening at the same location and at the same time, we can assume those to happen simultaneously. Think back to the example of the three customers i, j, and k where $c_{ij} = c_{jk} = c_{ki} = 1$ and $c_{ji} = c_{kj} = c_{ik} = 0$. If these three pickups are all at the same location and the time windows allow for a simultaneous pickup, the three customers could all enter the vehicle at the same time and no additional fourth customer would be needed. In fact, if only two of the three customers share the same location and the third enters the vehicle beforehand, it would also work, since there is no point in time in which a customer has to share the vehicle with an incompatible customer without at least one other compatible customer present.

5.4.1. Zero distance chains

We call a sequence where two or more customer actions happen simultaneously a zero distance chain (ZDC). Between visiting the first location and the last location in a ZDC, no time passes. Fig. 2 visualizes the aforementioned example: The sequences $j \to k$ and $n+i \to n+j$ are assumed to be ZDCs. Customers j and k enter the vehicle at the same time while customers i and j later leave the vehicle at the same time. Since there is no point in time where only two customers are in the vehicle, the tour is feasible. ZDCs are only relevant for the shared problem setting in Section 5.

Whenever we check a sequence for feasibility of incompatibilities, we therefore need to consider ZDCs. More formally:

Definition 1 (*Zero Distance Chain (ZDC)*)**.** The sequence $i \to j$ with $i, j \in P \cup D$ is a ZDC if $t_{ij} = 0, e_i \leq l_j$, and $e_i \geq e_j$.

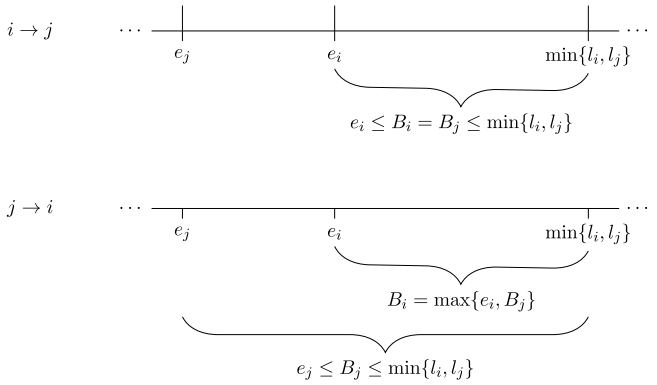

Fig. 3. The sequences $i \to j$ and $j \to i$ and the effect on the optimal arrival times B_i and B_j (assuming $t_{ij} = t_{ji} = 0$).

These conditions enforce that i and j share the same location, that the time windows overlap, and that the vehicle never has to wait when arriving at location j. Otherwise, if waiting occurs, there would potentially be a strictly positive time between the visits of i and j and the actions would not be simultaneous. Note that it is important here that in our setting it is never useful to delay the pickup of a customer by waiting at the location. The definition would not work if it was beneficial to visit a customer location later (e.g. with maximum ride time constraints), as it could be beneficial to wait before reaching location j.

The definition of ZDCs is exemplified in Fig. 3. For $t_{ij} = 0$ the upper part of the figure shows the ZDC as defined above. Location i is visited before location j. Thus, location i has to be visited in the interval $[e_i, \min\{l_i, l_j\}]$ to ensure that j is visited on time afterwards. Due to our objective (1), j is visited as early as possible if $j \in D$. If $j \in P$, it might be that waiting occurs before the next pickup location such that there is a degree of freedom in the departure time at j. Nevertheless, we can assume that j is visited directly after i, i.e. $B_i = B_j$, such that $i \to j$ is definitely a ZDC if $X_{ij} = 1$. If $e_j \neq e_i$, this is not true for the opposite direction $j \to i$ as the lower part of Fig. 3 shows. We argued already that it is never disadvantageous to visit a location as early as possible. Thus, $B_j < e_i$ and therefore $B_j < B_i$ might hold such that $j \to i$ is no ZDC. If $B_j \geq e_i$, $j \to i$ would also be a ZDC although not included in Definition 1. However, then the solutions are symmetric such that we do not need to consider $j \to i$ for $B_j \geq e_i$. A further advantage of excluding $j \to i$ for $B_j \geq e_i$ is that ZDCs are defined solely through the input data in Definition 1 (travel times and time windows). Therefore, all ZDCs can be identified in the preprocessing and are not reliant on the current arrival time in a solution. However, simultaneous actions in ZDCs make it more challenging to handle capacity constraints, which is addressed in Section 5.4.3 after we discussed how ZDCs can be handled in our Branch-and-Cut approach.

5.4.2. Handling of ZDCs

When adding a constraint to cut off an integer solution due to a violation of an incompatibility constraint, ZDCs need to be taken into account. Let i and j be incompatible and consider the following sequence which we might find in an integer feasible solution: $0 \to i \to j \to l \to \dots$ (with $t_{jl} > 0$, i.e. no ZDC). If ZDCs are not taken into account, we would add an inequality of the form $X_{0,i} + X_{ij} \leq 1$. However, if there is a possibility for a ZDC after location j, this inequality is not valid. Let k be a customer such that the sequence $j \to k$ is a ZDC and such that both i and j are compatible with k. The feasible start to a tour might then look like $0 \to i \to j \to k \to \dots$ where j and k enter the vehicle simultaneously. This sequence would be wrongly cut off with the original constraint. Therefore, if such a ZDC possibility exists, we add a constraint that includes the next vertex that prevents a ZDC, e.g. in this case $X_{0,i} + X_{ij} + X_{jl} \leq 2$.

When checking for ZDCs, two cases need to be distinguished: In order to prevent premature declarations of infeasibility for a sequence, the start or end vertex might need to be checked for potential ZDC candidates that could make the sequence feasible after all. This includes the previously mentioned checks for violations in integer solutions as well as Constraints (27) and (28), which we are only allowed to add if no ZDC possibility exists after j in Constraints (27) and before $n + i$ in Constraints (28). In fact, Constraints (27) would otherwise also (wrongly) cut off the previous example sequence $0 \to i \to j$. The same goes for the preprocessing fixing including the sequences (30) where ZDCs need to be considered beforehand: If a ZDC can prevent the conflict, we cannot fix the respective arc to 0. Similarly, before checking the sequences (29) in the shared *AMF*, we need to ensure that there is no ZDC possibility. If there is one, it would allow for a feasible resolution of the conflict which does not involve a customer entering the vehicle before the first path.

The second case of ZDC checks is done within the sequence itself which might already contain one or more ZDCs. Consider the sequence $0 \to i \to j \to k \to \dots$ where as before i and j are incompatible, $j \to k$ form a ZDC, and both i and j are compatible with k. When checking whether this sequence is feasible, we would normally track the current passengers in the vehicle and check for conflicts after each visited location since each additional pickup (or delivery) vertex might lead to incompatibilities. ZDCs can be included rather easily: It is sufficient to check that j and k form a ZDC and then the incompatibility check at j can be foregone. Instead, incompatibilities only need to be checked at the end of a ZDC at which point all of the simultaneous pickup and delivery activities are also reflected in the tracked current passengers. This is exemplified as pseudo code in Algorithm 1.

Algorithm 1 Checking for shared incompatibilities under consideration of ZDCs

```
function IsSequenceFeasible(sequence V)
    G ← list of passengers at the beginning of sequence V
    for all i in V do
        if i is pickup location then
            G ← G ∪ i
        else if i is delivery location then
            G ← G\{i−n}
        end if
        j ← location succeeding i in V
        if i and j do not form ZDC then
            if there is conflict between passengers in G then
                return false
            end if
        end if
    end for
    return true
end function
```

5.4.3. Handling of capacity constraints in ZDCs

In the sequence $i \to j$, if $i \in P$ and $j \in D$, both the pickup of i and the delivery of j would happen simultaneously in a ZDC. However, in such a case, the Inequalities (20) and (21) can cause an infeasibility if the vehicle already contains too many passengers to pick up i before j leaves. Since we assume simultaneous actions in a ZDC, this should, nevertheless, be feasible. Therefore, we omit Constraints (20) and (21) (and all Q_i variables) from the model in this setting and enforce the capacity constraints by checking our tours in each integer feasible solution for capacity violations (similar to how Ropke et al., 2007 omit the Q_i variables in their second presented model formulation). In these manual checks, we can then consider ZDCs.

When checking whether a sequence is feasible in regard to the capacity under consideration of ZDCs, the same principle can be applied as in the last paragraph of the previous section, i.e. capacity constraints only need to be checked at the end of a ZDC (and of course at all locations not included in a ZDC). This principle gives also a hint on how ZDCs can be implemented in the mixed-integer programming formulation: by excluding all beside the last location of a ZDC from Constraints (20)–(21) and (25)–(26).

5.5. Additional changes to the branch-and-Cut algorithm

ZDCs as well as preventing incompatibility by a third customer lead to the fact that some of the valid inequalities from the literature are not applicable anymore. In the preprocessing, it is not permissible to assume that visiting a delivery vertex as early as possible is always superior since we might require the customer in the vehicle to avoid an incompatibility. This concerns the first paragraphs in Section 3.2 in Schulz and Pfeiffer (2024). Additionally, the ZDCs cause issues for the strengthened capacity constraints by Ropke et al. (2007). A cut derived in such a manner is not guaranteed to be feasible, as a ZDC could connect a pickup and a delivery vertex to reduce the maximum number of passengers and avoid the capacity violation (effectively, the new passengers enter at the same time as the old passengers leave). We therefore omit these constraints in this setting.

6. Computational study

The procedure was implemented in C++ (Visual Studio 2017 version 15.9.8). The CPLEX API was used with CPLEX version 12.9. All tests were run on an AMD Ryzen Threadripper 3990X with 2.9 GHz (single-threaded) and 256 GB RAM.

With the computational study we aim at answering two questions regarding the performance of our algorithms as well as our research question:

1. Do our introduced methods improve the search significantly?
2. How do our algorithms perform?
3. What is the effect of allowing customers to declare incompatibilities with other customers?

Prior to our main computational study, we have run a series of tests to show the value of our algorithmic components to answer Question 1. As can be seen in the detailed descriptions in Appendix C, all introduced components – namely replacing Z_{ij} variables in the strict case, using the *AMF* procedure in the strict and the shared case, and the *lazy lifting* in the shared case – improve the search significantly.

Table 1
The combinations of n and K for which we generated instances.

$n \backslash K$	1	2	3	4	5	6	7	8	9	10
10	•	•	•	•	•					
15	•	•	•	•	•					
20		•	•	•	•	•				
25		•	•	•	•	•				
30			•	•	•	•	•			
35			•	•	•	•	•			
40				•	•	•	•	•		
45				•	•	•	•	•		
50					•	•	•	•	•	
60						•	•	•	•	•
70						•	•	•	•	•
80						•	•	•	•	•
90						•	•	•	•	•
100						•	•	•	•	•
120						•	•	•	•	•

Table 2
Average results for each set of 4,500 instances. Opt refers to the number of runs that finished with a proven optimal solution, Gap† to the gap over all instances not solved to optimality and Time [s]* to the average time over all optimally solved instances in seconds.

Incompatibility type	β	Opt	Gap†	Time [s]*
none	0	3638	0.2768	124.3428
strict	0.2	3863	0.2493	102.2373
strict	0.4	4101	0.2300	84.2188
shared	0.2	3406	0.3892	112.3915
shared	0.4	3340	0.4406	111.7044

6.1. Instances

The instances are taken from Schulz and Pfeiffer (2024) (available under https://doi.org/10.25592/uhhfdm.10389) and enhanced with incompatibility relations. A brief explanation of the instances follows: The distances are based on the virtual stops of a ridepooling provider in Hamburg, Germany. The number of customers and tours for the instances is varied according to Table 1, i.e. for each dot in the table a set of instances was generated. For each set, additionally the time window length was varied according to a factor α, where a larger α implies longer time windows. The three used settings of α were 1.1, 1.3, and 1.5 and for each combination of (n, K, α) 20 random instances were created, resulting in 4500 instances in total. The passenger capacity for each vehicle was set to six. For a more detailed explanation of the instances we refer to Schulz and Pfeiffer (2024).

We then added incompatibility relations to the instances. Let $0 \leq \beta \leq 1$ be a parameter that gives the average probability that a customer $i \in P$ is incompatible with a customer $j \in P, j \neq i$. We generated instances for $\beta \in \{0, 0.2, 0.4\}$. For $\beta = 0$, no incompatibilities are present and the strict and shared instances coincide. For each $\beta \in \{0.2, 0.4\}$, we generated for each of the 4500 instances a strict and a shared instance. Since in the strict setting the incompatibilities are symmetrical, we only generate incompatibilities for each customer pair $i, j \in P$ with $i < j$ since they count double when taking into account the reverse relation. In total, the testbed contains $5 \cdot 4500 = 22{,}500$ instances. All used instances with incompatibilities can be found under the following link: https://doi.org/10.25592/uhhfdm.11182.

Each run was limited to one hour of computational time. A trivial starting solution where every customer is rejected was given to CPLEX. For the separation heuristic to generate the fork constraints, we consider paths with a length of up to four locations while the separation heuristic for the reachability constraints considers incompatible groups of up to two locations. Otherwise, the separation procedures for the cutting planes from the literature follow the descriptions of Ropke et al. (2007).

6.2. Results

In the following, z refers to the (average) objective value and LB refers to the (average) lower bound. Gap refers to the CPLEX gap which is calculated as $(z - \text{LB})/(z + 1^{-10})$ for every instance. In Table 2, we can see the aggregated results for the strict and shared settings depending on the average percentage of incompatible relations in the instances β. The results from the row with incompatibility type *none* are from Schulz and Pfeiffer (2024) where incompatibilities were not considered. The Gap column is aggregated only over the non-optimal solutions, the Time column only over the optimal solutions. The strict instances appear to be easier to solve if a higher β is present since more arcs can be excluded in the preprocessing. Not only does $\beta = 0.4$ lead to more optimally solved instances in the strict case than $\beta = 0$ or $\beta = 0.2$, the optimality is proven faster as well and even the instances not solved to optimality feature smaller gaps at the end of the search.

Table 3
Comparison of the instances where the instance without incompatibilities (none), the strict, and the shared instance were all three solved to optimality. z^* is the average objective value over these instances of the respective type. Note that the objective values for the base case "none" can differ when the β level changes even though the instances are not affected by the β level. This is due to the considered instances which can change between the rows, as they depend on the instances solved to optimality in the other two settings. Time [s] is the average time in seconds of the respective type for the same instances.

n	β	z^*			Time [s]			# of instances
		none	strict	shared	none	strict	shared	
10	0.2	85.62	88.99	90.48	0.19	0.13	0.10	300
10	0.4	85.62	92.93	97.35	0.19	0.07	0.05	300
15	0.2	210.56	219.30	225.98	4.16	1.45	2.38	300
15	0.4	210.56	231.03	242.02	4.16	0.66	1.09	300
20	0.2	157.53	170.52	179.09	53.39	11.80	31.86	298
20	0.4	157.53	185.33	204.10	53.39	4.89	19.34	298
25	0.2	260.64	282.58	290.01	183.20	37.23	143.75	271
25	0.4	266.06	302.27	332.07	177.37	18.31	93.35	270
30	0.2	101.36	115.06	123.79	183.29	51.55	239.75	261
30	0.4	99.99	125.23	148.48	179.80	42.16	167.91	261
35	0.2	110.92	128.53	144.44	167.01	58.03	173.70	226
35	0.4	113.64	153.53	174.24	175.79	26.55	244.75	227
40	0.2	19.49	23.09	25.05	89.52	49.68	138.34	220
40	0.4	19.31	27.05	36.16	87.05	20.28	158.46	213
45	0.2	23.10	31.59	32.94	132.04	59.55	173.49	200
45	0.4	27.87	47.29	58.11	109.51	24.59	191.13	181
50	0.2	7.23	8.80	11.33	88.66	38.60	110.01	211
50	0.4	4.98	8.87	11.89	80.71	21.70	131.83	200
60	0.2	1.28	1.34	1.43	14.39	20.89	46.56	218
60	0.4	1.31	1.50	2.47	13.46	4.28	58.52	223
70	0.2	1.54	1.62	1.69	47.09	30.53	66.79	197
70	0.4	1.17	1.26	1.41	8.07	4.80	97.07	181
80	0.2	1.44	1.47	1.59	20.72	22.68	63.56	173
80	0.4	1.55	1.74	1.89	30.19	10.15	88.74	171
90	0.2	1.47	1.53	1.64	38.60	34.12	79.25	166
90	0.4	1.17	1.28	1.35	17.56	5.71	76.31	155
100	0.2	1.30	1.34	1.36	47.48	23.13	49.46	145
100	0.4	1.26	1.40	3.41	35.62	7.42	41.79	147
120	0.2	1.39	1.44	1.48	26.44	14.44	82.40	119
120	0.4	1.28	1.37	1.41	16.83	6.67	49.79	111

For the shared instances, the results do not follow the same pattern. Even for $\beta = 0.2$, the number of instances not solved to optimality and their average gap is much larger than in the case without incompatibilites and this holds for $\beta = 0.4$ as well. The larger gaps stem from very high rejections rates in many of the non-optimal solutions in the shared case: The upper bounds for these solutions are large, but the corresponding lower bounds are comparatively weak, i.e. it could not be proven that there is no solution with no rejections. There are likely multiple reasons for this effect: Firstly, unlike the strict case, the inequalities we add in the shared case are generally weaker and it is much more difficult to set an arc to 0 in the preprocessing. So, it is possible that the performance of the algorithm is simply worse. Secondly, the instances also have an impact. While the β values are the same for strict and shared, the actual result on the instance is different. In the strict case, due to the symmetries, an incompatible relation (i, j) also implies an incompatible relation (j, i). Therefore, two relations are incompatible but only one customer pair is affected. In the shared case, the incompatibilities are less local: Two incompatibility relations might affect three customers: Customer i is incompatible with j and j with k. Thus, there is generally a larger effect on the solution space in the shared case than in the strict one for the same level of β.

In summary, Table 2 gives us an answer to Question 2: Due to variable fixing in the preprocessing, the strict case is easier to solve than the base case. Besides, our algorithm for the shared case is weaker than for the strict case if we consider instances with the same number of incompatibilities.

Table 3 focuses only on optimal solutions. For each of the 4500 base instances without incompatibilities and each level of β, we have one strict and one shared instance that share the same underlying set of customers (including distances and time windows). Whenever all three – the base, the strict, and the shared instance belonging to this base instance – were solved to optimality, we included them in the average in the table.

We have several observations in Table 3:

(a) The objective value in the base setting without incompatibilities (none) is never higher than in one of the others. This is due to construction, as all instances are solved to optimality and added incompatibilities restrict the solution space.
(b) Average objective values decrease for increasing n. This means that the average relative detour of customers decreases which makes the service more attractive for customers if the system's size is larger. This is an observation also obtained in the studies by Pfeiffer and Schulz (2022a) and Schulz and Pfeiffer (2024).

(c) The absolute pairwise differences between the average objective values of the three settings decrease for increasing n. Although this is affected by the decreasing objective values, the relative differences increase first up to 40 customers and decrease afterwards (exceptions: 60 customers and $\beta = 0.4$ and 100 customers and $\beta = 0.4$ for the shared setting). A likely reason is that in the larger instances, the effect of incompatibilities is lessened by construction: As we increase the number of vehicles for a larger number of customers (see Table 1), the number of customers per vehicle does not increase strongly, i.e. the number of customer pairs not sharing a vehicle increases disproportionately high. Thus, the chance is higher that an incompatibility does not have an effect, as the customers would not have shared a vehicle in an optimal solution without incompatibilities anyway. This point leads to the managerial implication that allowing customers to codetermine to a certain degree the passengers possibly sharing the vehicle with them does not deteriorate the solution quality significantly. Involving the customers in the decision on their possible co-passengers can therefore be attractive for a ridepooling provider when taking account of the concerns customers had on their co-passengers in the study by Dolins et al. (2021).

(d) The strict setting is on average always better than the shared setting. However, this is not true in general. We also have instances which were solved optimally in the strict and the shared setting but not in the base setting and the shared setting led to a better objective value. The better results in the strict setting mean that the opportunity to share a vehicle for incompatible customers if a compatible customer is present does not compensate for the effect that probably fewer customer pairs are affected by incompatibilities in the strict setting.

(e) The computational times show that the strict instances can on average be solved faster than the corresponding base and shared instances. Moreover, the base instances can be solved faster on average than the corresponding shared instances for at least 40 customers. While the restrictions due to incompatibility reduce the solution space in the strict setting, reasons for the shared instances taking longer are likely found in the aforementioned difficulty in constructing tight cuts (in part due to the ZDCs which further weaken the constraints). Nevertheless, at least for the strict setting, shorter computation times confirm the benefits for a provider to allow their customers to participate in the decision process by indicating with which customers they want to share a ride.

While Observation (a) is obvious, Observation (b) confirms findings from the literature. Observations (c) and (d) give us a part of the answer to Question 3. (I) They show that allowing customers to declare incompatibilities has a smaller effect on the solution quality the larger the instances are. Moreover, Observation (e) confirms our findings for Question 2.

In Table 4, the rejection rate in the optimal solutions for each instance is examined. Due to the symmetries in the strict case, the rejection rates between the strict and shared cases should not be compared directly. We analyze these differences more thoroughly in the next paragraph. The rejection rate is higher in instances with higher β, which is our (II) insight to Question 3, and higher in shared than in strict instances. Expectedly, if α increases, i.e. the time windows are larger, the number of rejections goes down across all cases. It can also be seen that the number of rejections decreases with larger instance size. Additionally, the differences between the incompatibility cases decrease in the large instances as well. This is likely due to the larger number of vehicles which decreases the chance that two incompatible customers ever have to be in the same vehicle at the same time (see also Observation (c) for Table 3). There might be another reason as well: Since we only present the instances which were solved to optimality in all five cases, the displayed instances might be easier to solve than the others. This easiness might come from the fact that they allow for solutions in which no customers are rejected. It is possible that the optimal solutions for the more difficult instances would also display differences in the rejection rates between the incompatibility cases again.

So far, we have compared our algorithms for the strict and the shared case for the fraction of incompatibilities β. However, as already argued, incompatibilities are symmetric in the strict case. Thus, the number of affected customer pairs is probably larger in the shared variant instance than in the strict variant instance if β is identical. This introduces a bias in the comparison. In order to eliminate this bias, we solved our instances with the incompatibilities of the strict variant again but with the algorithm for the shared variant. This way, both versions have the same incompatibilities, but all incompatibilities are pairwise. Hence, the comparison is fair, as both algorithms solve the same instances. The results can be found in Table 5. In order to account for the worse algorithmic performance of the shared case, the table only includes the instances that were solved to optimality in both cases. This way, differences in the optimal objective values between the two cases are due to the case itself and not due to the algorithmic performance. The shared variant is effectively a relaxation of the strict variant, i.e. every solution feasible for the strict variant is also feasible in the shared case. Thus, an optimal shared solution can never have a worse objective value than the corresponding strict one. Overall, the differences in the objective values are not very high and decrease with increasing n. On the other hand, computational times in the shared case are significantly larger, further showing that the algorithmic performance is worse. In total, we found 419 of the 9000 instances (4.65%) where the shared case finished with an objective value that was smaller than the lower bound in the strict case, i.e. in these cases we could prove that the shared setting leads to a better objective value. As we have already seen in Table 3, optimal objective values vary only slightly between the three variants for larger instances. Thus, we can conclude (III) that allowing incompatible customers to share a vehicle if a further compatible customer is present only has a small effect on the solution quality. On the other hand, the algorithmic performance clearly deteriorates.

In summary, we derived the following answers to our research question, Question 3: (I) the difference in the solution quality between the cases decreases if the instance size increases. Moreover, (II) if the share of incompatibilities becomes large, our results show that the solution quality deteriorates and the number of rejections increases. Finally, (III) allowing a compatible customer to prevent the conflict of incompatible customers only has a small effect on the solution quality. However, this is not surprising, as in the best case all conflicts can be prevented by adding a compatible customer, but our first insight shows already that including incompatibilities only has a small effect on the solution quality if the number of customers in the instance is large.

Table 4
Average number of rejected customer requests for different n and α. The benchmark without incompatibilities is referred to by none, strict$^{0.2}$ refers to the strict instances with $\beta = 0.2$. Instances are only included if they were solved to optimality by all five solved variants.

n	α	Average # of rejected customers					# of instances
		none	strict$^{0.2}$	strict$^{0.4}$	shared$^{0.2}$	shared$^{0.4}$	
10	1.1	1.94	2.02	2.12	2.06	2.22	100
10	1.3	1.86	1.92	2.08	1.95	2.16	100
10	1.5	1.69	1.80	1.91	1.84	2.02	100
15	1.1	3.45	3.58	3.80	3.78	3.92	100
15	1.3	3.13	3.27	3.51	3.34	3.68	100
15	1.5	2.96	3.21	3.35	3.25	3.52	100
20	1.1	2.24	2.41	2.63	2.47	2.80	100
20	1.3	1.83	2.07	2.25	2.05	2.52	100
20	1.5	1.73	1.91	2.22	2.11	2.43	98
25	1.1	3.06	3.26	3.36	3.37	3.71	100
25	1.3	3.08	3.31	3.48	3.37	3.82	93
25	1.5	1.92	2.16	2.45	2.29	2.83	75
30	1.1	1.53	1.66	1.87	1.74	1.98	90
30	1.3	0.85	0.97	1.10	1.03	1.30	89
30	1.5	0.35	0.48	0.56	0.56	0.81	77
35	1.1	1.22	1.34	1.64	1.62	1.76	85
35	1.3	0.75	0.92	1.00	1.01	1.24	71
35	1.5	0.52	0.68	0.82	0.71	1.03	62
40	1.1	0.25	0.31	0.34	0.31	0.47	80
40	1.3	0.06	0.06	0.10	0.09	0.16	69
40	1.5	0.00	0.00	0.00	0.00	0.00	54
45	1.1	0.35	0.47	0.56	0.47	0.72	72
45	1.3	0.04	0.08	0.15	0.09	0.17	53
45	1.5	0.00	0.02	0.02	0.02	0.02	47
50	1.1	0.01	0.01	0.06	0.07	0.11	71
50	1.3	0.00	0.00	0.02	0.00	0.02	61
50	1.5	0.00	0.00	0.00	0.00	0.00	61
60	1.1	0.00	0.00	0.00	0.00	0.00	77
60	1.3	0.00	0.00	0.00	0.00	0.00	67
60	1.5	0.00	0.00	0.00	0.00	0.00	67
70	1.1	0.00	0.00	0.00	0.00	0.00	70
70	1.3	0.00	0.00	0.00	0.00	0.00	54
70	1.5	0.00	0.00	0.00	0.00	0.00	51
80	1.1	0.00	0.00	0.00	0.00	0.00	63
80	1.3	0.00	0.00	0.00	0.00	0.00	55
80	1.5	0.00	0.00	0.00	0.00	0.00	41
90	1.1	0.00	0.00	0.00	0.00	0.00	50
90	1.3	0.00	0.00	0.00	0.00	0.00	52
90	1.5	0.00	0.00	0.00	0.00	0.00	44
100	1.1	0.00	0.00	0.00	0.00	0.02	49
100	1.3	0.00	0.00	0.00	0.00	0.00	46
100	1.5	0.00	0.00	0.00	0.00	0.00	39
120	1.1	0.00	0.00	0.00	0.00	0.00	43
120	1.3	0.00	0.00	0.00	0.00	0.00	33
120	1.5	0.00	0.00	0.00	0.00	0.00	27

7. Conclusion

Ridepooling becomes more and more important and will presumably be applied with autonomous vehicles in the future. In an autonomous vehicle, the driver is missing as a person of trust which might result in customers feeling unsafe sharing the autonomous vehicle with customers of certain types. Therefore, the paper introduces a DARP with incompatibilities. We adapted the DARP to allow the provider to forbid certain customers to share the vehicle in general or at least if no suitable third customer is present in the vehicle. We introduced Branch-and-Cut algorithms for both problem variants as well as several methods to improve the search. Our computational results show that both approaches are able to solve realistic instances of reasonable size to optimality within one hour. Moreover, the strict case is easier to solve than the base case without incompatibilities while the base case is easier to solve than the shared case. Regarding our research question "*What is the effect of allowing customers to declare incompatibilities with other customers?*" we obtained the following results:

- The difference in solution quality decreases if the instance size increases.
- If the share of incompatibilities becomes large, the rejection rate increases and the solution quality deteriorates.
- Allowing incompatible customers to share a vehicle if a further compatible customer is present only has a small effect on the solution quality.

Table 5
Comparison of the strict instances solved by the strict algorithm and the same instances solved by the shared algorithm. Instances are only included if they were solved to optimality by both approaches.

n	strict		shared		# of instances
	z^*	Time [s]	z^*	Time [s]	
10	90.96	0.10	90.60	0.11	600
15	225.17	1.05	222.89	2.31	600
20	177.87	9.03	175.55	55.74	598
25	298.78	31.92	294.97	191.55	554
30	123.50	34.86	121.91	201.62	527
35	149.73	56.15	146.69	233.22	478
40	26.29	25.88	25.81	170.09	451
45	39.25	47.06	38.37	231.58	417
50	12.52	28.34	11.86	145.13	436
60	1.60	11.34	1.58	44.32	450
70	2.79	26.06	2.78	105.86	413
80	2.37	16.96	2.37	54.35	366
90	2.39	33.29	2.39	88.21	360
100	1.97	47.54	1.97	153.37	327
120	1.83	21.80	1.82	122.02	272

The results show that – at least for a larger system with a sufficiently large number of customers and vehicles – it can be beneficial for the provider to allow their customers to participate in the decision on their possible co-passengers. By including incompatibilities, the provider can address the customers' concerns about their co-passengers obtained in the study by Dolins et al. (2021).

Moreover, the results lead to the following managerial insight for ridepooling providers: If they want to allow their customers to declare incompatibilities with other customers, the effect on the solution quality depends on the number of declared incompatibilities and on the system's size (number of customers and vehicles). The larger the system is, the less likely it is that incompatible customers would be served together in one vehicle in an optimal solution without the incompatibilities. On the other hand, this becomes more likely if more incompatibilities are declared. The provider has to answer the question how large the fraction of incompatibilities should be depending on the system's size such that the effect on the solution quality stays moderate while the positive effect for the customers is maximized. However, we have to limit our findings: The incompatibilities were randomly distributed in our experiments. We did not evaluate how structures in the distribution of incompatibilities might affect the solution quality.

As the literature shows (Dolins et al., 2021) it is important to consider the composition of customers sharing an autonomous vehicle simultaneously. Thus, we see a need for further research with a focus on different aspects: First, we assumed incompatibilities to be randomly distributed. Moreover, we assumed all incompatibilities to be independent of each other. However, this might not hold in practice and should be evaluated in a concrete practical setting. Second, our approach can be extended. We assumed that from the point of view of one customer all other customers are either compatible or incompatible. However, a customer might want to share the vehicle with another customer but not with them and a third customer, i.e. the first customer does not want to share the vehicle with the third customer although the second customer is present. If customers can be compatible, but it is not possible to share the vehicle with them and an incompatible customer, we have a third category. By this, the number of combinations of customers who can share a vehicle in the shared case decreases, which might improve the search. Effectively, this combines the strict and the shared setting to a new setting where some incompatibility relations are strict and others are shared. Third, the problem setting can be applied to other DARP variants. Especially the avoidance of ZDCs might become more complicated if it is not beneficial to visit customer locations always as early as possible. Fourth, heuristic solution approaches can be developed and, finally, the shared setting can also be transferred to car pooling applications.

CRediT authorship contribution statement

Arne Schulz: Conceptualization, Methodology, Project administration, Writing – original draft, Writing – review & editing, Validation, Visualization. **Christian Pfeiffer:** Conceptualization, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Instances are available under the following links: https://doi.org/10.25592/uhhfdm.10389 (base instances), https://doi.org/10.25592/uhhfdm.11182 (instances with incompatibilities).

Appendix A. Proofs of valid inequalities

We prove that valid Inequalities (17)–(19) are valid for the strict incompatibility case in this section. This means that in any integer feasible solution, the valid inequalities are not violated. However, when solving a linear programming relaxation, they can further constrain the solution space and thus cut off fractional solutions. Let i and j be incompatible customers, i.e. $c_{ij} = 1$. In all three proofs, we show that no pair of two variables in the inequality can be 1 at the same time. Thus, we have three cases in each proof.

Proof that inequality (17) is valid:

- Assume that $Z_{ik} = 1$ and $X_{kj} = 1$. Because of (8), $Z_{jj} = 1$. Moreover, (7) implies $Z_{ij} \geq 1$ with $X_{kj} = 1$ and $Z_{ik} = 1$. However, $Z_{ij} + Z_{jj} \geq 2$ is a contradiction to (10).
- Assume that $Z_{ik} = 1$ and $X_{k,n+j} = 1$. If $X_{k,n+j} = 1$, location j has to be visited before location k (due to Constraints (5)). Because of (7) and (8), $Z_{jk} \geq 1$ follows. However, $Z_{ik} + Z_{jk} \geq 2$ is a contradiction to (10).
- Assume that $X_{kj} = 1$ and $X_{k,n+j} = 1$. This is a contradiction to (3).

Proof that inequality (18) is valid:

- Assume that $X_{ik} = 1$ and $X_{kj} = 1$ with $k \neq n + i$. With (7), (8), and $X_{ik} = 1$, $Z_{ik} \geq 1$ follows. Then, $Z_{ik} \geq 1$, $X_{kj} = 1$, and (7) imply $Z_{ij} \geq 1$. Because of (8), $Z_{jj} = 1$. However, $Z_{ij} + Z_{jj} \geq 2$ is a contradiction to (10).
- Assume that $X_{ik} = 1$ and $X_{k,n+j} = 1$ with $k \neq n + i$. If $X_{k,n+j} = 1$, location j has to be visited before location k (due to Constraints (5)). Because of (7) and (8), $Z_{jk} \geq 1$ follows. With (7), (8), and $X_{ik} = 1$, $Z_{ik} \geq 1$ follows. However, $Z_{ik} + Z_{jk} \geq 2$ is a contradiction to (10).
- Assume that $X_{kj} = 1$ and $X_{k,n+j} = 1$. This is a contradiction to (3).

Proof that inequality (19) is valid:

- Assume that $X_{k,n+i} = 1$ and $X_{jk} = 1$ with $k \neq i$. If $X_{k,n+i} = 1$, location i has to be visited before location k (due to Constraints (5)). Because of (7) and (8), $Z_{ik} \geq 1$ follows. With (7), (8), and $X_{jk} = 1$, $Z_{jk} \geq 1$ follows. However, $Z_{ik} + Z_{jk} \geq 2$ is a contradiction to (10).
- Assume that $X_{k,n+i} = 1$ and $X_{n+j,k} = 1$ with $k \neq i$. If $X_{k,n+i} = 1$ and $X_{n+j,k} = 1$, locations i and j have to be visited before location $n + j$ (due to Constraints (5)). Let k' be the location visited directly before $n + j$, i.e. $X_{k',n+j} = 1$. Because of (7) and (8), $Z_{ik'} \geq 1$ and $Z_{jk'} \geq 1$. However, $Z_{ik'} + Z_{jk'} \geq 2$ is a contradiction to (10).
- Assume that $X_{jk} = 1$ and $X_{n+j,k} = 1$. This is a contradiction to (2).

Appendix B. Proof of Theorem 1

In the Branch-and-Cut procedure, we start with paths R_{ii} containing only location i for each location. Thus, we can assume that we have feasible paths R_{hk} and R_{lm} (induction hypothesis) and have to show that we never get an infeasibility due to incompatibility if they are merged. Therefore, we consider all cases for the position of two incompatible customers i and j in a merge of two paths R_{hk} and R_{lm}. Table B.1 pictures all cases, whether they are feasible or not, and in the latter case why they are not feasible. Thereby, an entry i, $n + i$ means that i and $n + i$ are in this sequence part of the corresponding path, but it might be that different locations are visited before, in between or afterwards. However, locations i, j, $n + i$, and $n + j$ are not part of the paths if they are not mentioned. Note that sometimes the roles of i and j are swapped in Conditions (22) and (23) and that we only mention one reason for infeasibility although there might be more than one. Moreover, we exclude cases in which either R_{hk} or R_{lm} are empty, as the merge is feasible if the non-empty path is feasible. If R_{hk} or R_{lm} is already infeasible, this is a contradiction to our induction hypothesis.

The remaining cases (R_{hk} starts with j or $n + j$) can be obtained by swapping the roles of customers i and j.

Appendix C. Component testing

In order to show the contribution of our various algorithmic components, we have run a series of tests. The instances for these tests have been generated in the same way as the instances we used for the computational study (see Section 6.1). However, a different random seed has been used in the generation so that the instances were generated independently. The instances for the strict and for the shared setting only differ in the incompatibilities. The other parts of each instance (distances, time windows) are identical. We generated instances with $n \in \{90, 100\}$, $k \in \{6, 7, 8, 9, 10\}$, $\alpha \in \{1.1, 1.3, 1.5\}$, and $\beta = 0.2$ and 20 instances per setting, i.e. 600 instances for the strict tests and 600 instances for the shared tests. As the components take advantage of the incompatibilities between customers, we would expect that for instances with a large β, the advantage of the components increases as well. Therefore, we only run these tests for a setting of 0.2 since if a component can significantly improve the results for $\beta = 0.2$, it will likely also hold for larger values.

The instances for these tests can be found under the same link as the ones for the main computational study. Each run had an hour of computational time.

Table B.1
Cases for the position of incompatible customers i and j in a merge of paths R_{hk} and R_{lm}.

R_{hk}	R_{lm}	feasibility	reason
i	j	infeasible	infeasible due to (22)
i	$n+i$	feasible	–
i	$n+j$	infeasible	infeasible due to (22)
i,j	$n+i$	infeasible	R_{hk} is infeasible due to (22)
i	$j,n+i$	infeasible	R_{lm} is infeasible due to (22)
$i,n+i$	j	feasible	–
i	$n+i,j$	feasible	–
i,j	$n+j$	infeasible	R_{hk} is infeasible due to (22)
i	$j,n+j$	infeasible	infeasible due to (22)
$i,n+j$	j	infeasible	R_{hk} is infeasible due to (22)
i	$n+j,j$	infeasible	R_{lm} is infeasible due to precedence constraints
$i,n+i$	$n+j$	infeasible	infeasible due to (23)
i	$n+i,n+j$	infeasible	R_{lm} is infeasible due to (23)
$i,n+j$	$n+i$	infeasible	R_{hk} is infeasible due to (22)
i	$n+j,n+i$	infeasible	R_{lm} is infeasible due to (23)
$i,j,n+i$	$n+j$	infeasible	R_{hk} is infeasible due to (22)
i,j	$n+i,n+j$	infeasible	R_{hk} is infeasible due to (22)
i	$j,n+i,n+j$	infeasible	R_{lm} is infeasible due to (22)
$i,j,n+j$	$n+i$	infeasible	R_{hk} is infeasible due to (22)
i,j	$n+j,n+i$	infeasible	R_{hk} is infeasible due to (22)
i	$j,n+j,n+i$	infeasible	R_{lm} is infeasible due to (23)
$i,n+i,j$	$n+j$	feasible	–
$i,n+i$	$j,n+j$	feasible	–
i	$n+i,j,n+j$	feasible	–
$i,n+i,n+j$	j	infeasible	R_{hk} is infeasible due to (23)
$i,n+i$	$n+j,j$	infeasible	R_{lm} is infeasible due to precedence constraints
i	$n+i,n+j,j$	infeasible	R_{lm} is infeasible due to precedence constraints
$i,n+j,j$	$n+i$	infeasible	R_{hk} is infeasible due to precedence constraints
$i,n+j$	$j,n+i$	infeasible	R_{hk} is infeasible due to (22)
i	$n+j,j,n+i$	infeasible	R_{lm} is infeasible due to precedence constraints
$i,n+j,n+i$	j	infeasible	R_{hk} is infeasible due to (23)
$i,n+j$	$n+i,j$	infeasible	R_{hk} is infeasible due to (22)
i	$n+j,n+i,j$	infeasible	R_{lm} is infeasible due to precedence constraints
$n+i$	j	feasible	–
$n+i$	i	infeasible	infeasible due to precedence constraints
$n+i$	$n+j$	infeasible	infeasible due to (23)
$n+i,j$	i	infeasible	infeasible due to precedence constraints
$n+i$	j,i	infeasible	R_{lm} is infeasible due to (22)
$n+i,i$	j	infeasible	R_{hk} is infeasible due to precedence constraints
$n+i$	i,j	infeasible	R_{lm} is infeasible due to (22)
$n+i,j$	$n+j$	feasible	–
$n+i$	$j,n+j$	feasible	–
$n+i,n+j$	j	infeasible	R_{hk} is infeasible due to (23)
$n+i$	$n+j,j$	infeasible	R_{lm} is infeasible due to precedence constraints
$n+i,i$	$n+j$	infeasible	R_{hk} is infeasible due to precedence constraints
$n+i$	$i,n+j$	infeasible	R_{lm} is infeasible due to (22)
$n+i,n+j$	i	infeasible	R_{hk} is infeasible due to (23)
$n+i$	$n+j,i$	infeasible	infeasible due to (23)
$n+i,j,i$	$n+j$	infeasible	R_{hk} is infeasible due to precedence constraints
$n+i,j$	$i,n+j$	infeasible	R_{lm} is infeasible due to (22)
$n+i$	$j,i,n+j$	infeasible	R_{lm} is infeasible due to (22)
$n+i,j,n+j$	i	infeasible	infeasible due to precedence constraints
$n+i,j$	$n+j,i$	infeasible	infeasible due to precedence constraints
$n+i$	$j,n+j,i$	infeasible	infeasible due to precedence constraints
$n+i,i,j$	$n+j$	infeasible	R_{hk} is infeasible due to precedence constraints
$n+i,i$	$j,n+j$	infeasible	R_{hk} is infeasible due to precedence constraints
$n+i$	$i,j,n+j$	infeasible	R_{lm} is infeasible due to (22)
$n+i,i,n+j$	j	infeasible	R_{hk} is infeasible due to precedence constraints
$n+i,i$	$n+j,j$	infeasible	R_{hk} is infeasible due to precedence constraints
$n+i$	$i,n+j,j$	infeasible	R_{lm} is infeasible due to precedence constraints
$n+i,n+j,j$	i	infeasible	R_{hk} is infeasible due to precedence constraints
$n+i,n+j$	j,i	infeasible	R_{hk} is infeasible due to (23)
$n+i$	$n+j,j,i$	infeasible	R_{lm} is infeasible due to precedence constraints
$n+i,n+j,i$	j	infeasible	R_{hk} is infeasible due to precedence constraints
$n+i,n+j$	i,j	infeasible	R_{hk} is infeasible due to (23)
$n+i$	$n+j,i,j$	infeasible	R_{lm} is infeasible due to precedence constraints

Table C.1
The average mixed integer programming (MIP) gap when the components for the strict test are turned on or off. The gap is calculated as $(z^{UB} - z^{LB})/z^{UB}$ where z^{UB} (z^{LB}) is the upper (lower) bound for the optimal objective value. Note that the gap is reported as a percentage. The asterisk indicates the p-value in a paired, two-sided t-test between the instances where the component was turned on and where it was turned off.

component	average gap [%]	
	turned on	turned off
replace Z_{ij}	4.97	8.78**
strict AMF	6.75	7.01*

* :$p < 0.05$.
** :$p < 0.001$.

Table C.2
The average MIP gap when the components for the shared test are turned on or off. The gap is calculated as $(z^{UB} - z^{LB})/z^{UB}$ where z^{UB} (z^{LB}) is the upper (lower) bound for the optimal objective value. Note that the gap is reported as a percentage. The asterisk indicates the p-value in a paired, two-sided t-test between the instances where the component was turned on and where it was turned off.

Component	average gap [%]	
	turned on	turned off
shared AMF	18.46	21.79**
lazy lifting	17.77	22.48**

** :$p < 0.001$.

C.1. Strict setting

For the strict setting, we tested two components: We first tested whether it is worthwhile to use the Z_{ij} variables in the original model or to replace them by the capacity variables Q_i and the integer solution verification in Section 4.4.2 (*replace* Z_{ij}). Secondly, we tested whether the *AMF* procedure in Section 4.4.1 is useful (*strict AMF*). We did not test any of the preprocessing steps, as their computational effort is negligible.

The effects of the components on the average gap are shown in Table C.1. Replacing the Z_{ij} variables can drastically improve the performance, as the average gap goes from 8.78 to 4.97 if they are replaced. This is likely due to Constraints (10), as the average number of processed nodes goes down from 9478 to 2357 when they are included, implying a slowdown in the solution time for the simplex algorithm. The effect on the average gap for *strict AMF* is smaller but nonetheless statistically significant ($p < 0.05$).

C.2. Shared setting

In the shared setting, we tested two components as well. The first component is the shared *AMF* procedure described in Section 5.1 (*shared AMF*). The second component is the lifting of the lazy constraints in Section 5.2 (*lazy lifting*). While the constraints in this section are necessary for the correctness of our algorithm, the tightening described in the last paragraph is not and can thus be evaluated.

Table C.2 shows the results. For both components we have a significant reduction in the average gap when the component is utilized. In particular, the lifting of the constraints that cut off infeasible integer solutions prove to be useful, as they cause a larger difference in the average gap when turned off.

References

Agatz, N., Erera, A., Savelsbergh, M.W., Wang, X., 2012. Optimization for dynamic ride-sharing: A review. European J. Oper. Res. 223 (2), 295–303.
Alonso-Mora, J., Wallar, A., Rus, D., 2017. Predictive routing for autonomous mobility-on-demand systems with ride-sharing. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS, IEEE, pp. 3583–3590.
Ascheuer, N., Fischetti, M., Grtschel, M., 2000. A polyhedral study of the asymmetric traveling salesman problem with time windows. Networks 36 (2), 69–79.
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.
Bernardino, R., Paias, A., 2022. The family traveling salesman problem with incompatibility constraints. Networks 79 (1), 47–82.
Bongiovanni, C., 2020. The electric autonomous dial-a-ride problem (Dissertation). École Polytechnique Fédérale de Lausanne.
Bongiovanni, C., Kaspi, M., Cordeau, J.-F., Geroliminis, N., 2019a. A learning large neighborhood search for the dynamic electric autonomous dial-a-ride problem. In: hEART 2019 - 8th Symposium of the European Association for Research in Transportation. September 4–6, 2019, Budapest, Hungary.
Bongiovanni, C., Kaspi, M., Geroliminis, N., 2019b. The electric autonomous dial-a-ride problem. Transp. Res. B 122, 436–456.
Cheng, X., Fu, S., Sun, J., Zuo, M., Meng, X., 2023. Trust in online ride-sharing transactions: Impacts of heterogeneous order features. J. Manage. Inf. Syst. 40 (1), 183–207.
Cheng, X., Su, L., Yang, B., 2020. An investigation into sharing economy enabled ridesharing drivers' trust: A qualitative study. Electron. Commer. Res. Appl. 40, 100956.

Christiaens, J., Çalik, H., Wauters, T., Chirayil Chandrasekharan, R., Vanden Berghe, G., 2020. The prisoner transportation problem. European J. Oper. Res. 284 (3), 1058–1073.
Colombi, M., Corberán, Á., Mansini, R., Plana, I., Sanchis, J.M., 2017. The directed profitable rural postman problem with incompatibility constraints. European J. Oper. Res. 261 (2), 549–562.
Cordeau, J.-F., 2006. A branch-and-cut algorithm for the dial-a-ride problem. Oper. Res. 54 (3), 573–586.
Correia, G., Viegas, J.M., 2011. Carpooling and carpool clubs: Clarifying concepts and assessing value enhancement possibilities through a Stated Preference web survey in Lisbon, Portugal. Transp. Res. A 45 (2), 81–90.
Dolins, S., Strömberg, H., Wong, Y.Z., Karlsson, M., 2021. Sharing anxiety is in the driver's seat: Analyzing user acceptance of dynamic ridepooling and its implications for shared autonomous mobility. Sustainability 13 (14), 7828.
Factorovich, P., Méndez-Díaz, I., Zabala, P., 2020. Pickup and delivery problem with incompatibility constraints. Comput. Oper. Res. 113, 104805.
Ferone, D., Festa, P., Pastore, T., Resende, M.G., 2023. Efficient GRASP solution approach for the Prisoner Transportation Problem. Comput. Oper. Res. 153, 106161.
Gendreau, M., Manerba, D., Mansini, R., 2016. The multi-vehicle traveling purchaser problem with pairwise incompatibility constraints and unitary demands: A branch-and-price approach. European J. Oper. Res. 248 (1), 59–71.
Gschwind, T., Irnich, S., 2015. Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. Transp. Sci. 49 (2), 335–354.
Hasan, M.H., van Hentenryck, P., 2021. The benefits of autonomous vehicles for community-based trip sharing. Transp. Res. C 124, 102929.
Ho, S.C., Szeto, W.Y., Kuo, Y.-H., Leung, J.M.Y., Petering, M., Tou, T.W.H., 2018. A survey of dial-a-ride problems: Literature review and recent developments. Transp. Res. B 111, 395–421.
Hsieh, F.-S., Zhan, F.-M., 2018. A discrete differential evolution algorithm for carpooling. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference. COMPSAC, IEEE, pp. 577–582.
Huang, S.-C., Jiau, M.-K., Lin, C.-H., 2015. A genetic-algorithm-based approach to solve carpool service problems in cloud computing. IEEE Trans. Intell. Transp. Syst. 16 (1), 352–364.
Hyland, M., Mahmassani, H.S., 2018. Dynamic autonomous vehicle fleet operations: Optimization-based strategies to assign AVs to immediate traveler demand requests. Transp. Res. C 92, 278–297.
Hyland, M., Mahmassani, H.S., 2020. Operational benefits and challenges of shared-ride automated mobility-on-demand services. Transp. Res. A 134, 251–270.
Johnsen, L.C., Meisel, F., 2022. Interrelated trips in the rural dial-a-ride problem with autonomous vehicles. European J. Oper. Res. 303 (1), 201–219.
Kostorz, N., Fraedrich, E., Kagerbauer, M., 2021. Usage and user characteristics — Insights from MOIA, Europe's largest ridepooling service. Sustainability 13 (2), 958.
Li, M., Zheng, N., Wu, X., Li, W., Wu, J., 2020. An efficient solving method to vehicle and passenger matching problem for sharing autonomous vehicle system. J. Adv. Transp. 2020.
Liang, X., de Almeida Correia, G.H., An, K., van Arem, B., 2020. Automated taxis' dial-a-ride problem with ride-sharing considering congestion-based dynamic travel times. Transp. Res. C 112, 260–281.
Lokhandwala, M., Cai, H., 2020. Understanding the impact of heterogeneous rider preferences on a shared autonomous vehicle system. Transp. Res. F 75, 120–133.
Lu, Y., Benlic, U., Wu, Q., 2018. A memetic algorithm for the orienteering problem with mandatory visits and exclusionary constraints. European J. Oper. Res. 268 (1), 54–69.
Lysgaard, J., 2006. Reachability cuts for the vehicle routing problem with time windows. European J. Oper. Res. 175 (1), 210–223.
Ma, N., Zeng, Z., Wang, Y., Xu, J., 2021. Balanced strategy based on environment and user benefit-oriented carpooling service mode for commuting trips. Transportation 48 (3), 1241–1266.
Manerba, D., Mansini, R., 2015. A branch-and-cut algorithm for the multi-vehicle traveling purchaser problem with pairwise incompatibility constraints. Networks 65 (2), 139–154.
MOIA, 2022. Hamburg, welcome to your autonomous driving future! https://www.moia.io/en/innovation. (Accessed 28 July 2023).
Palomo-Martínez, P.J., Salazar-Aguilar, M.A., Albornoz, V.M., 2017. Formulations for the orienteering problem with additional constraints. Ann. Oper. Res. 258 (2), 503–545.
Parragh, S.N., Doerner, K.F., Hartl, R.F., 2008. A survey on pickup and delivery models: Part II: Transportation between pickup and delivery locations. J. Betriebswirtschaft 58 (2), 81–117.
Parragh, S.N., Pinho de Sousa, J., Almada-Lobo, B., 2015. The dial-a-ride problem with split requests and profits. Transp. Sci. 49 (2), 311–334.
Pfeiffer, C., Schulz, A., 2022a. An ALNS algorithm for the static dial-a-ride problem with ride and waiting time minimization. OR Spectrum 44, 87–119.
Pfeiffer, C., Schulz, A., 2022b. A new lower bound for the static dial-a-ride problem with ride and waiting time minimization. In: International Conference on Dynamics in Logistics. Springer, pp. 231–243.
Pimenta, V., Quilliot, A., Toussaint, H., Vigo, D., 2017. Models and algorithms for reliability-oriented dial-a-ride with autonomous electric vehicles. European J. Oper. Res. 257 (2), 601–613.
Psaraftis, H.N., 1980. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. Transp. Sci. 14 (2), 130–154.
Psaraftis, H.N., 1983. An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. Transp. Sci. 17 (3), 351–357.
Riedler, M., Raidl, G., 2018. Solving a selective dial-a-ride problem with logic-based Benders decomposition. Comput. Oper. Res. 96, 30–54.
Rist, Y., Forbes, M.A., 2021. A new formulation for the dial-a-ride problem. Transp. Sci. 55 (5), 1113–1135.
Rist, Y., Forbes, M., 2022. A column generation and combinatorial Benders decomposition algorithm for the selective dial-a-ride-problem. Comput. Oper. Res. 140, 105649.
Ropke, S., Cordeau, J.-F., 2009. Branch and cut and price for the pickup and delivery problem with time windows. Transp. Sci. 43 (3), 267–286.
Ropke, S., Cordeau, J.-F., Laporte, G., 2007. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks 49 (4), 258–272.
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.
Sanguinetti, A., Kurani, K., Ferguson, B., 2019. Is it OK to get in a car with a stranger? Risks and benefits of ride-pooling in shared automated vehicles. https://escholarship.org/uc/item/1cb6n6r9. (Accessed 28 July 2023).
Schulz, A., Pfeiffer, C., 2024. Using fixed paths to improve branch-and-cut algorithms for precedence-constrained routing problems. European J. Oper. Res. 312 (2), 456–472.
Su, Y., Dupin, N., Puchinger, J., 2022. A column-generation-based heuristic for the electric autonomous dial-a-ride problem. In: 23ÈME CONGRÈS de la ROADEF.
Su, Y., Dupin, N., Puchinger, J., 2023. A deterministic annealing local search for the electric autonomous dial-a-ride problem. European J. Oper. Res. 309 (3), 1091–1111.
von Mörner, M., 2019. Demand-oriented mobility solutions for rural areas using autonomous vehicles. In: Coppola, P., Esztergár-Kiss, D. (Eds.), Autonomous Vehicles and Future Mobility. Elsevier, pp. 43–56.
Yan, S., Chen, C.-Y., 2011. An optimization model and a solution algorithm for the many-to-many car pooling problem. Ann. Oper. Res. 191 (1), 37–71.
Yan, S., Chen, C.-Y., Lin, Y.-F., 2011. A model with a heuristic algorithm for solving the long-term many-to-many car pooling problem. IEEE Trans. Intell. Transp. Syst. 12 (4), 1362–1373.

Zhang, L., Liu, Z., Yu, L., Fang, K., Yao, B., Yu, B., 2022. Routing optimization of shared autonomous electric vehicles under uncertain travel time and uncertain service time. Transp. Res. E 157, 102548.
Zwick, F., Kuehnel, N., Moeckel, R., Axhausen, K.W., 2021. Agent-based simulation of city-wide autonomous ride-pooling and the impact on traffic noise. Transp. Res. D 90, 102673.