Technical Report AC-TR-16-007
Sep 2016

Solving a selective Dial-a-Ride problem with Logic-based Benders Decomposition

Martin Riedler and Günther Raidl
Solving a selective Dial-a-Ride problem with Logic-based Benders Decomposition

Martin Riedler∗, Günther Raidl

Institute of Computer Graphics and Algorithms, TU Wien, Vienna, Austria

Abstract

Today’s society is facing an ever-growing demand for mobility. Large amounts of these needs can be fulfilled by individual transport and public transport. People that do not have access to the former and cannot use the latter require additional means of transportation. This is where dial-a-ride services come into play. The dial-a-ride problem considers transportation requests of people from pick-up to drop-off locations. Users specify time windows w.r.t. these points. Requests are served by a given vehicle fleet with limited capacity and tour duration per vehicle. Moreover, user inconvenience considerations are taken into account by limiting the travel time between origin and destination for each request.

Previous research on the dial-a-ride problem primarily focused on serving a given set of requests with a fixed-size vehicle fleet at minimal traveling costs. It is assumed that the request set is sufficiently small to be served by the available vehicles. We consider a different scenario in which a maximal number of requests shall be served under the given constraints, i.e., it is no longer guaranteed that all requests can be accepted. For this new problem variant we propose a compact mixed integer linear programming model as well as algorithms based on Benders decomposition. In particular we employ logic-based Benders decomposition and Branch-and-Check using mixed integer linear programming and constraint programming algorithms. We consider different variants on how to generate Benders cuts as well as heuristic boosting techniques and different types of valid inequalities. Computational experiments illustrate the effectiveness of the suggested algorithms.

Keywords: Transportation, Dial-a-Ride Problem, Logic-based Benders Decomposition, Combinatorial Benders Cuts, Branch-and-Check

1. Introduction

The dial-a-ride problem (DARP) considers the design of vehicle routes for a set of costumers specifying requests in the form of wanting to be transported
from an origin location (pick-up) to a destination (drop-off). Users typically impose time-windows w.r.t. these locations. To reduce user inconvenience the time required to go from the pick-up to the drop-off location (ride time) is limited. The available requests shall be served by a fleet of vehicles. Each vehicle has a limited capacity corresponding to the number of customers that can be transported and a maximum total travel time. The restriction on the tour duration is important in order to deal with restrictions regarding driver shifts.

As done by Jaw et al. [26], Cordeau [8], and others we distinguish between outbound and inbound requests. An outbound request considers the case that a customer wants to go from some starting location to a destination. An inbound requests corresponds to the opposite case, i.e., a customer that wants to return to his/her starting location. According to the survey presented in [27] customers have different priorities w.r.t. the adherence to time-windows. For outbound requests it is critical to stay within the time-window at the drop-off location and for inbound requests the priority is to keep the time-window at the pick-up location.

In the literature several variants of the DARP have been investigated. The two main variants are the static and the dynamic case. In the former it is assumed that all requests are known in advance whereas in the latter requests become known gradually over time and routes need to be adjusted accordingly. Of course, there are also mixed variants for which some requests are known in advance and some are revealed dynamically. Moreover, there is a distinction between the single- and the multi-vehicle case. In the former variant the requests need to be served using a single vehicle and in the latter variant multiple vehicles are available. In the following we deal with the static multi-vehicle DARP.

1.1. Our Contribution and Structure of the Article

In many DARP applications it is assumed that all requests can be served and that the total travel expenses together with the user inconvenience need to be minimized. In contrast, we consider the scenario that in general not all customers can be handled with the given fixed-size vehicle fleet and aim at maximizing the amount of served requests. This is intended to deal with situations in which dial-a-ride systems are overallocated. In these cases serving as many customers as possible appears to be more relevant than savings due to shorter tour lengths. Of course user inconvenience considerations still need to be taken into account to provide reasonable service conditions.

The remainder of the article is organized as follows. We first provide an overview of previous work in the area and give details on the used methodological concepts. Then we provide a formal definition of the specific problem case we consider, together with a compact reference model that is a straightforward extension of the usual tour-length-minimization DARP. Starting from there we develop approaches based on Benders decomposition (BD). We consider solution algorithms based on logic-based Benders decomposition (LBB) (see Hooker and Ottosson [24]) and Branch-and-Check (BaC) (see Thorsteinsson [35]). The Benders master-problem is modeled as integer program. For solving
the Benders subproblems we consider mixed integer linear programming (MILP) as well as constraint programming (CP) algorithms. Several strategies for the generation and strengthening of Benders cuts are suggested. Moreover, we consider heuristic boosting techniques to possibly speed up the solution process. We present computational results on various test instances and finally conclude with an outlook on future research directions.

1.2. Previous Work

The DARP has a rather long research history. Among the first was the work by Psaraftis [29] that deals with the static single-vehicle variant. Sexton and Bodin [33, 34] solve the problem by splitting it into a routing and a scheduling phase which they formally describe in the context of Benders decomposition. The routing is done by an insertion heuristic. In [4] the same authors use this approach to tackle the multi-vehicle case by first forming clusters of requests and then solving the single-vehicle problem for each cluster. Since they construct the clusters (grouping close customers) as well as the routes heuristically both methods cannot guarantee optimal solutions. Later on, this approach to the multi-vehicle problem has been refined by using so-called “mini-clusters”, see [14, 13]. The most recent contribution by Ioachim et al. [25] relying on this technique shows the positive influence of using mathematical optimization methods to globally define the set of “mini-clusters”. The authors argue that more sophisticated techniques provide a significant advantage over simpler heuristic approaches. However, all of these algorithms are still heuristics.

Only few contributions so far do not minimize traveling costs. Wolfler Calvo and Colorni [37] maximize the number of served customers and consider a penalty term regarding user inconvenience. This term considers the relative ratio between the direct and the actual travel time. The authors consider a fast heuristic construction approach based on an auxiliary graph.

Also relevant to our work are the contributions by Berbeglia et al. [3], and Häme and Hakula [21] that focus on feasibility checking of DARP instances. Although we consider an optimization problem here, we are still concerned with feasibility checking when it comes to the Benders subproblems.

For a broader overview on the DARP we refer to the surveys by Cordeau and Laporte [9, 11] and Parragh et al. [28].

An optimization problem closely related to the DARP is the pickup and delivery problem with time windows (PDPTW). The main difference between the two problems is that the PDPTW primarily deals with the transportation of goods rather than persons. As a consequence, it does not consider user inconvenience and related concerns. In this area branch-price-and-cut approaches have been shown to be able to provide state-of-the-art results, see Ropke and Cordeau [32] and Baldacci et al. [1], respectively. For further details consider the survey conducted in [28].

Finally, we want to review contributions that are relevant to our work from the methodological point of view, i.e., works that apply (logic-based) Benders decomposition in the context of vehicle routing problems. Cire and Hooker [6]
consider the Home Health Care problem in which medical services need to be provided to patients. Each service is represented as a job and requires a certain minimal qualification level. The services are provided by nurses that travel to the patients. The aim is to design routes and shift plans s.t. all required services can be provided while minimizing the costs for the nurses' working hours. The problem is solved using LBBD. In the master-problem the jobs are assigned to the nurses and the subproblems determine the actual shift plan and route per nurse. After solving a subproblem a cut is introduced into the master-problem reflecting the cost of the assignment or prohibiting an infeasible allocation. In case of an infeasible subproblem it is often possible to strengthen the obtained cut by identifying a subset of assigned jobs that is the cause of the infeasibility. Moreover, a local search procedure is employed that tries to repair infeasible solutions by reassigning jobs to other nurses. The authors solve the master-problem only heuristically and therefore optimal solutions cannot be guaranteed. In the computational study the LBBD approach is compared to a CP model which it outperforms clearly.

The bi-level vehicle routing problem (VRP) considers the distribution of goods in two stages. The goods are first transported from the main depot to satellite depots. Starting at each satellite depot the goods are transported to the customers. This kind of VRP arises for example in newspaper distribution. Raidl et al. [30, 31] consider a bi-level VRP with a global restriction on the time until which all customers need to receive their goods. The assignment of customers to the satellite depots is pre-specified. Deliveries are carried out with a homogeneous fleet of vehicles with restricted capacity. The goal is to perform all deliveries within the time limit at minimal routing cost. Due to the structure of the problem routing costs at the first level as well as for every satellite depot can be considered independently. However, the levels are still interlinked via the global time limit. These properties provide a promising basis for the application of LBBD. Raidl et al. [30, 31] consider a decomposition approach in which the master-problem determines the route from the main depot to the satellite depots. With the now fixed starting times at the satellite depots the corresponding routes can be computed independently. Infeasibilities (due to the global time limit) are prevented by computing a minimal starting time for each satellite depot that guarantees the existence of a feasible route. Hence, only Benders optimality cuts are required. These cuts turn out to be quite strong here since routing costs can only be reduced given a smaller starting time at the respective depot. Raidl et al. [30] consider an exact variant of this decomposition, as well as a hybrid approach with either the master or the subproblems solved via metaheuristics, and a completely heuristic approach. In Raidl et al. [31] the hybrid approach is further refined by verifying and, if needed, correcting the heuristically added Benders cuts in a second phase. With this approach the obtained solution is guaranteed to be provably optimal but the solution process is much faster than the purely exact one.
2. Methodology

In this section we introduce the decomposition techniques that build the basis for the algorithms presented in the remainder of this work.

2.1. Logic-based Benders Decomposition

The so-called logic-based Benders decomposition (LBBD) has been introduced by Hooker and Ottosson [24]. It extends the well known classical Benders decomposition (BD), by allowing also integer variables and possibly nonlinearities in the subproblem. Standard BD was originally proposed to solve large linear programming (LP) problems having variables that can be partitioned into two subsets \((x, y)\) s.t. the problem separates into one or more easier solvable subproblems on the \(x\)-variables after fixing the \(y\)-variables.

The general shape of these problems is the following, where \(S\) is the set of feasible solutions and \(D_x, D_y\) are the domains of \(x\) and \(y\):

\[
\min f(x, y) \quad (1)
\]

subject to

\[
(x, y) \in S \quad (2)
\]
\[
x \in D_x \quad (3)
\]
\[
y \in D_y \quad (4)
\]

The idea is to decompose the problem into a master-problem only using the \(y\)-variables and depending subproblems expressed on the \(x\)-variables. The master-problem is obtained by removing all elements containing \(x\)-variables. Instead, their contributions are modeled by additional inequalities (6), called Benders cuts:

\[
\min z \quad (5)
\]

subject to

\[
z \geq \beta_k(y) \quad \forall k \in K \quad (6)
\]
\[
y \in D_y \quad (7)
\]

In the above model the new variable \(z\) corresponds to the original objective function \(f(x, y)\) and is now determined by the Benders cuts. To solve this model, one starts with a reduced master problem containing no (or only a small initial set of) Benders cuts, yielding a solution \(\bar{y}\). When considering again the original problem (1–4) and fixing the \(y\)-variables to \(\bar{y}\), we get the following subproblem, solely defined on the \(x\)-variables and supposed to be in some sense much easier to solve than the original problem:

\[
\min f(x, \bar{y}) \quad (8)
\]
Considering the dual of the subproblem we want to find the best possible lower bound $\beta^*$ on the optimal solution value, when $y$ is fixed to $\bar{y}$ that can be inferred from the constraints. The main challenge is to identify a bounding function $\beta_{\bar{y}}(y)$ providing a valid lower bound on the optimal objective value of (1) given any fixed value $y$.

If all subproblems are feasible, they yield a valid lower bound $\beta_{\bar{y}}(y)$ w.r.t. the current assignment of the $y$-variables. So-called Benders optimality cuts are derived from the dual solution and are added to incorporate this information into the master-problem. If one or more subproblems turn out to be infeasible, this means that the considered solution on the $y$-variables is not acceptable. This information is also communicated to the master-problem by means of Benders cuts which are then called feasibility cuts. In particular, since infeasible subproblems have unbounded dual we obtain $\beta_{\bar{y}}(\bar{y}) = \infty$. With this additional information we continue and search for a better assignment of the $y$-variables by solving the augmented master-problem and then again solve the corresponding subproblem(s). This procedure is iterated until it is no longer possible to find an assignment of the $y$-variables that improves the objective and no infeasible subproblems remain. Then, an optimal solution to the initial problem has been found. We denote by $\beta_{\bar{y}}(\bar{y})$ the bounding function inferred from the $k$th trial value $\bar{y}^k$. According to Hooker and Ottosson [24] it is known that if in every iteration $k$ of the Benders algorithm

- the Benders cut $z \geq \beta_{\bar{y}}(y)$ is valid, i.e., any feasible solution $(x, y)$ to (1) satisfies $f(x, y) \geq \beta_{\bar{y}}(y)$ and
- $\beta_{\bar{y}}(\bar{y}^k) = \beta$ where $\beta$ is the optimal solution to the dual of (8).

then given finite domain $D_y$ for the $y$-variables and if the subproblem dual is solved to optimality, the generic Benders algorithm terminates with the correct result.

It is often the case that not all $x$-variables are linked to each other after the $y$-variables have been fixed. In these cases it is possible to split the subproblem into smaller independent problems that can be solved separately. A prominent example for this case are problems whose constraint matrix has dual block angular structure, see Figure 1. This structure is the dual to the one arising and exploited in Dantzig-Wolfe Decomposition [12]. Having independently solvable, smaller and/or easier subproblems is one of the advantages of BD as it often helps to deal with much larger problem instances than is possible when solving the problem as a whole. We are going to exploit this feature in our algorithms.

In the original BD the subproblems are restricted to be LPs. Geoffrion showed in [19] how to extend the method to other convex optimization methods using nonlinear convex duality theory. This allows for a systematic generation
of the bounding function by means of duality theory. Unfortunately, this also limits the applicability of the approach.

LBBBD allows for more general subproblems. However, in contrast to the traditional BD there exists no (single) systematic way to identify a strong bounding function for the Benders cuts. The concept of LP duality is generalized using so-called inference duality, and tailored cuts have to be identified w.r.t. the encountered subproblems.

LBBBD has been applied effectively in several areas including planning and scheduling (Hooker [23], Hamdi and Loukil [20]), location problems (Fazel-Zarandi and Beck [15], Wheatley et al. [36]), survivable network design (Garg and Smith [17]), and vehicle routing (Cire and Hooker [6], Raidl et al. [30, 31]).

2.2. Branch-and-Check

The idea behind classical BD and LBBBD is to always (re)solve the master-problem to optimality. This solution is then used as basis for the subproblems. However, this might not always be necessary. Suboptimal solutions can be sufficient to derive relevant cuts for the master-problem, as it seems reasonable to generate all Benders cuts within a single branch-and-cut (B&C) tree for identified intermediate solutions.

This idea was first introduced in Hooker [22] and further examined in [35] and is also closely related to the concept of Combinatorial Benders cuts considered by Codato and Fischetti [7]. Thorsteinsson [35] referred to this strategy as Branch-and-Check (BaC). We will adopt this term in the following.
Using the terminology introduced for BD, BaC basically specifies a single problem defined only on the \( y \)-variables together with their constraints. This problem is then solved. Whenever a feasible solution is identified within the B&C tree the corresponding subproblems are derived and solved. Dependant on their solutions Benders cuts are added, possibly cutting off the current solution. The main difference to LBBD is that the master-problem is solved only once and that Benders cuts are typically generated w.r.t. suboptimal assignments of the \( y \)-variables inside the B&C tree.

We want to emphasize that technically there is no difference between traditional B&C and BaC. The difference between them is only conceptual. B&C typically deals with constraints that represent single aspects of the problem (e.g., subtour elimination constraints in routing) whereas BaC defers the main properties of the considered problem into the subproblems.

When dealing solely with feasibility cuts, i.e., the solution of the subproblems has no influence on the objective of the master-problem, BaC has one advantage over LBBD that really stands out. Since BaC operates on the branch-and-bound tree and iteratively approaches the feasible area it is usually capable of finding feasible solutions quite fast. LBBD, on the other hand, either terminates with an optimal solution or no feasible solution at all. Often it is possible to derive a feasible solution from the trial values of the intermediate Benders iterations, however, this requires additional computational effort which is not incurred when using BaC.

3. Formulations

Our variant of the DARP is defined on a directed graph \( G = (N, A) \). Given \( n \) requests, the set of vertices \( N \) consists of two copies of the depot \( \{0, 2n + 1\} \), the set of pick-up locations \( P = \{1, \ldots, n\} \), and the set of drop-off locations \( D = \{n + 1, \ldots, 2n\} \). A request corresponds to a pair \((i, n+i)\) s.t. \( i \in P \) and \((n+i) \in D \). In the following we occasionally identify requests by their corresponding pick-up locations. The load (e.g., the number of persons to be transported) at each pick-up location \( i \in P \) is given by \( q_i \geq 0 \) and the same amount is to be unloaded at the drop-off location, i.e., \( q_{n+i} = -q_i \). The service duration at each node \( i \in N \) is given by \( d_i \geq 0 \). For the depot \( q_0 = q_{2n+1} = d_0 = d_{2n+1} = 0 \) holds. In addition, each node \( i \) has an associated time window \([e_i, l_i]\), \( e_i \leq l_i \).

The set of arcs is defined as \( A = \{(i, j) \mid (i = 0 \land j \in P) \lor (i, j \in P \cup D \land i \neq j \land i \neq n + j) \lor (i \in D \land j = 2n + 1)\} \). The non-negative travel time of arc \((i, j)\) is \( t_{ij}\) and the maximal ride time is denoted by \( L \). We are given a set of vehicles \( K \) and every vehicle \( k \in K \) has a maximum capacity \( Q^k \) and a maximum route duration \( T^k \). Moreover, we assume that a time horizon limited by \( T \) is given, i.e., all requests have to be served in the time window \([0, T]\).

The goal is to serve as many requests as possible respecting all time windows, precedence constraints, capacity restrictions, maximum route durations, and the maximum ride times.
3.1. Complexity

The original DARP has been shown to be \( \mathcal{NP} \)-hard (see Baugh et al. [2]) and it can easily be seen that the problem still remains \( \mathcal{NP} \)-hard under the modified scenario. After requests have been assigned to the vehicles we still need to find a Hamiltonian path per vehicle which is known to be \( \mathcal{NP} \)-hard on its own, see Garey and Johnson [16].

**Theorem 1.** The selective DARP is \( \mathcal{NP} \)-hard.

**Proof.** An instance of the Hamiltonian path problem for graph \( G = (V, E) \) can be reduced to an instance of the selective DARP as follows. Create a request for each vertex of \( G \) and consider a single vehicle. We set \( d_i = q_i = 0 \) for all nodes and \( Q_1 = T_1 = L = \infty \). For the pick-up nodes we set time windows to \([0, |E|]\) and for drop-off nodes we set time windows to \([|E| + 1, \infty]\). For \( i \) and \( j \), both pick-up nodes, we set \( t_{ij} = 1 \) if the corresponding vertices are connected in \( G \), otherwise \( t_{ij} = \infty \). The remaining travel times are set to zero. Then, there exists a Hamiltonian path w.r.t. \( G \) iff the objective to the selective DARP is equal to \(|V|\).

3.2. Compact Model

The following MILP model is a slightly modified variant of the one introduced in [8]. We are going to refer to it as compact model (CM). The difference is that we are maximizing the amount of requests served, instead of minimizing travel costs.

We use binary variables \( x^k_{ij} \) for each arc \((i, j) \in A\) per vehicle \( k \in K \). Moreover, variables \( B^k_i \) and \( Q^k_i \) are used to track for each vehicle \( k \in K \) the beginning-of-service time and the load at node \( i \in N \) after serving \( i \), respectively. Finally, we use variables \( L^k_i \) to model the ride time of each request identified by its pick-up location \( i \in P \) on vehicle \( k \in K \).

\[
\begin{align*}
\text{max } & \sum_{k \in K} \sum_{(i,j) \in A} x^k_{ij} \\
\text{subject to } & \\
& \sum_{k \in K} \sum_{(i,j) \in A} x^k_{ij} \leq 1 \quad \forall i \in P \\
& \sum_{(i,j) \in A} x^k_{ij} - \sum_{(n+i,j) \in A} x^k_{n+i,j} = 0 \quad \forall i \in P, \forall k \in K \\
& \sum_{j \in P} x^k_{i0} = 1 \quad \forall k \in K \\
& \sum_{(j,i) \in A} x^k_{ji} - \sum_{(i,j) \in A} x^k_{ij} = 0 \quad \forall i \in P \cup D, \forall k \in K \\
& \sum_{i \in D} x^k_{i,2n+1} = 1 \quad \forall k \in K 
\end{align*}
\]
The objective function \((11)\) determines the number of served requests by counting the selected arcs leading to pick-up nodes. Constraints \((12)\) ensure that each request is served by at most one vehicle. These are the only differences to the original model. Constraints \((13)\) guarantee that pick-up and drop-off of each request are served by the same vehicle. Equalities \((14)\) to \((16)\) ensure that each vehicle leaves the depot as well as each node it visits and that it finally returns to the depot. Constraints \((17)\) and \((18)\) enforce that \(B\) and \(Q\) variables are set correctly. Note that in addition to tracking the beginning-of-service times these constraints also serve as a variant of Miller-Tucker-Zemlin constraints to prevent subtours. Equalities \((19)\) calculate the ride time for each request and inequalities \((20)\) limit the route duration for each vehicle. The remaining inequalities ensure that the used variables stay within their respective domains.

The quadratic constraints \((17)\) and \((18)\) can be linearized as follows:

\[
(B^k_i + d_i + t_{ij})x^k_{ij} \leq B^k_j \quad \forall (i, j) \in A, \forall k \in K \quad (17)
\]
\[
(Q^k_i + q_j)x^k_{ij} \leq Q^k_j \quad \forall (i, j) \in A, \forall k \in K \quad (18)
\]

with the Big-M constants set to \(M^k_{ij} = \max\{0, l_i + d_i + t_{ij} - e_j\}\) and \(W^k_{ij} = \min\{Q^k_i, Q^k + q_j\}\), respectively.

### 3.3. Decomposition Approach

For the decomposition approach we split the problem into a master-problem and several subproblems. The master-problem is responsible for assigning the requests to the vehicles. When an assignment has been identified we generate one subproblem per vehicle to check if it is possible to find a feasible tour satisfying the DARP constraints.

\[
(master) \quad \max \sum_{k \in K} \sum_{i \in P} y^k_i \quad (27)
\]

subject to

\[
\sum_{k \in K} y^k_i \leq 1 \quad \forall i \in P \quad (28)
\]
Benders cuts \( \forall k \in K \)  
\( \forall k \in K, \forall i \in P \)  

\( y^k_i \in \{0,1\} \)

The master-problem maximizes the number of requests that are served. Constraints (28) ensure that each request is assigned to at most one vehicle. The Benders cuts (29) will be provided by the subproblems. They are responsible for preventing infeasible assignments of requests to one of the vehicles. Furthermore, we will later augment this base master-problem by initially provided strengthening inequalities.

We formulate the subproblems \( sub(k, I) \) based on a vehicle \( k \in K \) and a subset \( I \subseteq P \) of the requests. Dependent on a solution \( \bar{y} \) to the master-problem we identify for each vehicle \( k \in K \) the set \( I^k = \{i \in P \mid \bar{y}^k_i = 1\} \) of assigned requests. Each of these sets results in an independently solvable subproblem \( sub(k, I^k) \). The subproblems can be stated similar to the compact formulation introduced in the previous section and essentially constitute feasibility-based single-vehicle DARPs.

For subproblem \( sub(k, I) \) let \( P^I = I \) and \( D^I = \{n+i \mid i \in I\} \) be the pick-up and drop-off locations corresponding to set \( I \), resulting in a restricted set of vertices \( N^I = \{0, 2n+1\} \cup P^I \cup D^I \). According to \( N^I \) we define reduced arc set \( A^I = A \setminus \{(i,j) \mid i \notin N^I \lor j \notin N^I\} \). Then, the subproblems can be modeled as follows:

\[
(sub(k, I)) \quad \min 0 \quad \text{subject to} \\
\sum_{(i,j) \in A^I} x_{ij} = 1 \quad \forall i \in P^I \cup D^I \quad (32) \\
\sum_{j \in P^I} x_{0j} = 1 \\
\sum_{(j,i) \in A^I} x_{ji} = \sum_{(i,j) \in A^I} x_{ij} = 0 \quad \forall i \in P^I \cup D^I \quad (34) \\
\sum_{i \in D^I} x_{i,2n+1} = 1 \\
B_i + d_i + t_{ij} - M^k_{ij}(1-x_{ij}) \leq B_j \quad \forall (i,j) \in A^I \quad (36) \\
Q_k + q_j - W^k_{ij}(1-x_{ij}) \leq Q_j \quad \forall (i,j) \in A^I \quad (37) \\
B_{n+i} - (B_i + d_i) = L_i \\
B_{2n+1} - B_0 \leq T^k \\
e_i \leq B_i \leq l_i \\
t_i,n+i \leq L_i \leq L \\
\max\{0,q_i\} \leq Q_i \leq \min\{Q^k,Q^k + q_i\} \quad \forall i \in N^I \quad (42)
\]
Note that the objective function (31) is constant since we are only interested whether there exists a feasible tour or not, i.e., this is actually a decision problem. In each subproblem all assigned requests \( I \) have to be served. We no longer need to enforce that pick-up and drop-off location are visited by the same vehicle since we only consider one vehicle. It is sufficient to use constraints (32) for ensuring that both pick-up and drop-off locations are visited for each assigned request. The remaining parts stay basically the same.

In addition to the MILP formulation we also provide a CP model similar to the one introduced in [3] but restricted to the single vehicle case and slightly adjusted. To formulate element constraints we define a bijective function \( \pi : N^I \mapsto \{0, \ldots, 2 \cdot |I| + 1\} \) mapping the nodes required in the subproblem to a consecutive range as follows. Depot node 0 is mapped to itself and depot copy \( 2n + 1 \) is mapped to \( 2 \cdot |I| + 1 \). Nodes in \( P^I \) are mapped to \( \{1, \ldots, |I|\} \) and those in \( D^I \) to \( \{|I| + 1, \ldots, 2 \cdot |I|\} \) s.t. it holds that \( \pi(i) = j \) iff \( \pi(i + n) = j + |I| \), \( \forall i \in D^I \). Accordingly, we define sets \( \tilde{P}^I = \{\pi(i) | i \in P^I\} \), \( \tilde{D}^I = \{\pi(i) | i \in D^I\} \), and \( \tilde{N}^I = \tilde{P}^I \cup \tilde{D}^I \cup \{0, 2 \cdot |I| + 1\} \). Additionally, we specify an appropriately reduced travel time matrix \( \tilde{t}_{ij} = t_{\pi^{-1}(i)\pi^{-1}(j)} \), \( \forall (i,j) \in \tilde{N}^I \times \tilde{N}^I \) and load values \( \tilde{q}_i = q_{\pi^{-1}(i)} \), \( \forall i \in \tilde{N}^I \). Observe that the remaining input (service duration, time windows) is not part of element constraints and therefore does not need transformed data structures.

To provide the model we use three sets of variables which are successor variables \( s[i], \forall i \in \tilde{N}^I \setminus \{2 \cdot |I| + 1\} \), load variables \( q[i], \forall i \in \tilde{N}^I \), and beginning-of-service time variables \( b[i], \forall i \in \tilde{N}^I \). The model reads as follows:

\[
\text{(sub}(k, I))
\]

subject to

\[
\text{allDifferent}(s) \quad (44)
\]

\[
b[i] + \tilde{t}_{i,|I|+i} + d_{\pi^{-1}(i)} \leq b[|I|+i] \quad \forall i \in \tilde{P}^I \quad (45)
\]

\[
b[i] + \tilde{t}_{i,s[i]} + d_{\pi^{-1}(i)} \leq b[s[i]] \quad \forall i \in \tilde{N}^I \setminus \{2 \cdot |I| + 1\} \quad (46)
\]

\[
b[i + n] - (b[i] + d_{\pi^{-1}(i)}) \leq L \quad \forall i \in \tilde{D}^I \quad (47)
\]

\[
b[2 \cdot |I| + 1] - b[0] \leq T^k
\]

\[
q[i] + \tilde{q}_{d[i]} = q[s[i]] \quad \forall i \in \tilde{N}^I \quad (48)
\]

\[
s[i] \in \{j | (i,j) \in A^I\} \quad \forall i \in \tilde{N}^I \setminus \{2 \cdot |I| + 1\} \quad (49)
\]

\[
domain(b[i], e_{\pi^{-1}(i)}, l_{\pi^{-1}(i)}) \quad \forall i \in \tilde{N}^I \quad (50)
\]

\[
domain(q[i], \tilde{q}_i, Q^k) \quad \forall i \in \tilde{P}^I \quad (51)
\]

\[
domain(q[i], 0, Q^k + \tilde{q}_i) \quad \forall i \in \tilde{D}^I \quad (52)
\]

\[
q[0] = 0
\]
\[ q[2n + 1] = 0 \quad (56) \]

Constraints (45) ensure that each node has a unique successor. Assuming that the triangle inequality holds, we know that the beginning-of-service times at the pick-up location and the corresponding drop-off location differ at least by the direct travel time (46). Constraints (47) model the time needed to travel from a node to its immediate successor. Inequalities (48) and (49) restrict the ride time and tour duration appropriately. The remaining constraints specify the variable domains.

3.3.1. Benders Cuts

If a subproblem turns out to be infeasible, we need to add a cut preventing that the requests that caused the infeasibility are again assigned to the same route in subsequent iterations. The easiest way to do this is to add a Benders cut preventing the exact same assignment and any superset of it.

In iteration \( j \) we denote by \( I^k_j \) the requests assigned to vehicle \( k \in K \) and by \( \mathcal{K}_j = \{ k \in K \mid \text{sub}(k, I^k_j) \text{ is infeasible} \} \) the set of vehicles for which the subproblem turns out to be infeasible. The corresponding Benders cuts are:

\[
\sum_{i \in I^k_j} y^k_i \leq |I^k_j| - 1 \quad \forall k \in \mathcal{K}_j \quad (57)
\]

This basic cut, however, frequently can be strengthened as it is likely that the infeasibility is caused by a proper subset of the assigned requests only. Similar to the notation used in [7] we classify sets \( I^k_j \) as follows:

**Definition 1.** We call a set of requests \( I^k_j \) infeasible iff subproblem \( \text{sub}(I^k_j) \) is infeasible and feasible otherwise.

**Definition 2.** We call an infeasible set of requests \( I^k_j \) irreducible infeasible set (IIS) iff the removal of any request turns it into a feasible set and reducible infeasible set otherwise.

Reducible infeasible sets lead to unnecessarily weak Benders cuts. Therefore, we never want to add cuts that are based on reducible infeasible sets. In general, there exist several IISs of smaller cardinality for each reducible infeasible set \( I^k_j \). All such sets are by definition minimal and, thus, lead to non-dominated cuts within this class of cuts. Note that the IISs w.r.t. a given base set can have different cardinality. For practical reasons it makes sense to prefer smaller sets when the amount of IISs gets large. Moreover, each Benders cut prevents assignments that are supersets w.r.t. its underlying IIS. Hence, IISs of minimum cardinality are in general able to cut-off larger parts of the search space.

Unfortunately, there is neither an efficient way to compute all IISs nor those of minimum cardinality. However, by means of a greedy strategy we are at least able to reduce a given base set to an IIS efficiently, see Algorithm 3.1. The algorithm tries to remove requests one after another and checks each time if
algorithm 3.1: greedy set reduction

\begin{algorithm}
\begin{algorithmic}
  \STATE Input: Set $I$ of requests and vehicle $k \in K$ s.t. $\text{sub}(k, I)$ is infeasible.
  \STATE \hfill // $I$ is now an IIS
  \FOR {\textit{i} \in I}
    \STATE if $\text{sub}(k, I \setminus \{i\})$ is infeasible then
    \STATE $I = I \setminus \{i\}$;
  \END\FOR
  \RETURN $I$;
\end{algorithmic}
\end{algorithm}

the resulting set is still infeasible. If this is the case, we keep the smaller set, otherwise we proceed with the next request.

At line 2 we need to check if there exists a feasible route for a given set of requests w.r.t. a specific vehicle. This can be done by trying to solve the according instance of the subproblem of the decomposition approach introduced in the previous section. Note that the order in which the requests are considered has in general a strong influence on the outcome of the algorithm. As mentioned before, smaller IISs are usually preferable as they cut off larger parts of the search space. The greedy strategy cannot guarantee to compute a set of minimum cardinality. Consequently, we should attempt to order the requests heuristically to increase the chances of ending up with a small set. Unfortunately, it is not trivial to find an appropriate ordering that can be computed quickly. One strategy would be to prioritize the removal of requests that are unlikely to be the cause of the infeasibility. Unfortunately, identifying these requests is again difficult. Following preliminary experiments, we finally decided in favor of low running times to just keep the natural order of the requests. However, to decrease the chances of ending up with bad results we iterate twice through the set, the second time in reverse order and add both obtained cuts if they are distinct.

To analyze the impact of heuristically computed sets, we also consider adding cuts for all IISs as well as only for those of minimum cardinality.

Cuts obtained for one vehicle can also be added to the master-problem for other vehicles with equally or more restrictive characteristics:

Definition 3. We define a partial order denoted by $\leq^k$ on the vehicles:

$$k_1 \leq^k k_2 \Leftrightarrow (Q^{k_1} \leq Q^{k_2}) \land (T^{k_1} \leq T^{k_2}) \quad k_1, k_2 \in K$$

We can add Benders cuts for all vehicles with at most the capacity and the maximum tour length of the vehicle for which infeasibility has been detected:

$$\sum_{i \in I_j^k} y_{ik}^{k'} \leq |I_j^k| - 1 \quad \forall k \in K, \forall k' \in K : k' \leq^k k$$  \hspace{1cm} (58)
4. Algorithmic Framework

We start with some remarks on preprocessing that help to reduce the size of the problem instances for certain cases. Then, we present details for our algorithms and further techniques that help to speed up the solving process.

4.1. Preprocessing

In this section we describe the used preprocessing techniques. Several of them are based on the concepts introduced in [8]. We point out our modifications.

4.1.1. Time-Window Tightening

In [8] several techniques for time-window tightening are introduced. For outbound requests we can set the time window at the pick-up location to $e_i \leftarrow \max\{e_i, e_{n+i} - L - d_i\}$ and $l_i \leftarrow \min\{l_{n+i} - t_{i,n+i} - d_i, l_i\}$. Similarly, we set the time windows for drop-off nodes of inbound users to $e_n+i \leftarrow \max\{e_{n+i}, e_{n+i} + d_i + t_{i,n+i}\}$ and to $l_{n+i} \leftarrow \min\{l_{n+i} + d_i + L, l_{n+i}\}$. The time windows on depot copies 0 and $(2n+1)$ are set to $e_0 = e_{2n+1} \leftarrow \min\{e_i - t_0, i \in P \cup D\}$ and $l_0 = l_{2n+1} \leftarrow \max\{l_i + d_i + t_{i,2n+1}\}$.

We suggest to modify this slightly since this might lead to unwanted effects when requests are too close to the depot, i.e., $t_0 > e_i$ or $l_i + t_{i,2n+1} > T$. In these cases we might end up with increasing the time horizon $[0, T]$. To avoid this we additionally apply $e_i \leftarrow \max\{e_i, t_0\}$ and $l_{n+i} \leftarrow \min\{l_{n+i}, T - t_{n+i,2n+1}\}$ for $i \in P$. Afterwards it is safe to tighten the time windows at the depot nodes as described. Alternatively, this can be taken into account during the following arc elimination.

4.1.2. Arc Elimination

As done in [8] we also eliminate arcs from $A$ that cannot be part of a feasible solution. The following situations are considered:

- arcs $(0, n+i), (i, 2n+1)$, and $(n+i, i)$ are infeasible for $i \in P$ (this is already considered by the definition of the arc set)
- arc $(i, j)$ is infeasible if $e_i + d_i + t_{ij} > l_j$
- arcs $(i, j)$ and $(j, n+i)$ with $i \in P, j \in N$ are both infeasible if $t_{ij} + d_j + t_{j,n+i} > L$
- arc $(i, n+j)$ is infeasible if path $(j, i, n+j, n+i)$ is infeasible
- arc $(n+i, j)$ is infeasible if path $(i, n+i, j, n+j)$ is infeasible
- arc $(i, j)$ is infeasible if paths $(i, j, n+i, n+j)$ and $(i, j, n+j, n+i)$ are both infeasible
- arc $(n+i, n+j)$ is infeasible if paths $(i, j, n+i, n+j)$ and $(j, i, n+i, n+j)$ are both infeasible

15
When checking the feasibility of paths we also need to compute the forward time slack. In [10] the forward time slack $F_i$ at node $i$ in a path from $i$ to $q$ is computed as follows:

$$F_i = \min_{i \leq j \leq q} \left\{ \sum_{i < p \leq j} W_p + \max \{0, \min \{l_j - B_j, L - P_j\}\} \right\}, \quad (59)$$

where $W_i$ denotes the waiting time at node $i$ and $P_i$ represents the ride time for the request with destination node $i \in D$. For the remaining $i$ we define $P_i = -\infty$. The second term of the inner minimum-function, i.e., $L - P_j$, is required to avoid that for some requests the maximum user ride time is exceeded.

If time windows of the requests do not prevent vehicles from returning too late to the depot, i.e., $l_i + t_{i,2n+1} > T$ for $i \in P$, we augment the paths considered above by including the depot $(2n+1)$ as final vertex. Similarly, it makes sense to add depot 0 as first vertex if $t_{0i} > e_i$ can be the case for some pick-up locations $i \in P$. If this is not done, we might miss to detect some infeasibilities. This might happen due to a too early beginning-of-service time in the former case and due to a too high forward time slack in the latter case.

4.1.3. Infeasible Request Pairs

As stated in [8] two requests $(i,n+i)$ and $(j,n+j)$ cannot be served by the same vehicle if none of the following connections is possible:

$$(i,j,i+n,j+n)$$
$$(i,j,j+n,i+n)$$
$$(i,i+n,j,j+n)$$
$$(j,i,i+n,j+n)$$
$$(j,i+j+n,i+n)$$
$$(j,j+n,i,i+n)$$

Observe that a request is only feasible (assuming that the triangle inequality holds) if the direct connection between pick-up and drop-off is feasible. Therefore, we assume that both $(i,i+n)$ and $(j,j+n)$ are feasible since it makes no sense to consider infeasible requests. Thus, it is sufficient to check if at least one of the following options is available:

$$(i,j) \land (j,i+n) \land (i+n,j+n)$$
$$(i,j) \land (j+n,i+n)$$
$$(i+n,j)$$
$$(j,i) \land (i+n,j+n)$$
$$(j,i) \land (i,j+n) \land (j+n,i+n)$$
$$(j+n,i)$$

If none of them is possible, these two requests cannot be served by the same vehicle. As a consequence this also allows to remove all arcs between those
requests. Let $C$ be the set of all incompatible request pairs identified by their pick-up locations, i.e., $C \subseteq \{(i,j) \mid (i,j) \in P \times P, i < j\}$. Then, we can add the following constraints to the master-problem:

$$y_k^i + y_k^j \leq 1 \quad \forall k \in K, \forall (i,j) \in C$$  (60)

These constraints are essentially instances of Benders cuts for which the set of infeasible requests has cardinality two. Therefore, these are the strongest cuts of this type. They are enumerated exhaustively and added to the initial formulation. We also add this type of constraints to the compact model using the sum of outgoing arcs for nodes $i$ and $j$ instead of the assignment variables.

In [8] the incompatible request pairs are used to fix certain requests to vehicles. This cannot be done here since it is unknown which requests will be served and which will be rejected.

### 4.1.4. Capacities

For each request $i \in P$ we have a minimal time window for which the corresponding customer has to be on board of a vehicle if the request is accepted, which is $[l_i, e_{n+i}]$. Of course the actual time window in which the user is on board of the vehicle is usually larger but we only know things for sure w.r.t. this window.

Now we define sets of requests with overlapping time windows unless some requests have been identified to be incompatible by other techniques.

**Definition 4.** Request $(i, n+i)$ overlaps with request $(j, n+j)$ for $i, j \in P$ if one of the following holds:

$$l_i \leq l_j \leq e_{n+i} \quad (61)$$

$$l_i \leq l_{n+j} \leq e_{n+i} \quad (62)$$

$$l_j < l_i \land (e_{n+i} < e_{n+j}) \quad (63)$$

We then define graph $G^C = (V^C, E^C)$ as follows. We set $V^C = P$ and $E^C$ contains an edge for each $i, j \in P : i \neq j$ if the respective requests overlap but are not incompatible.

In this graph we identify all maximal cliques. This can be done by the Bron-Kerbosch algorithm, see [5]. The cliques in $G^C$ define sets of requests that have to be on board together when served by the same vehicle. We now need to determine whether all of them fit in the vehicle simultaneously. For each maximal clique and each vehicle $k \in K$ we sum up the loads of the corresponding requests, starting with the smallest one until we exceed the vehicle capacity. Then, we know the maximum amount of requests in the clique that can be served by this vehicle.

Let $\mathcal{C}$ be the set of all maximal cliques in $G^C$. For each $C \in \mathcal{C}$ let $k^m_C$ be the maximum amount of requests in $C$ that fit into vehicle $k \in K$. Then, we can add the following inequalities to the master-problem:

$$\sum_{i \in C} y_k^i \leq k^m_C \quad \forall C \in \mathcal{C} : k^m_C < |C|$$  (64)
Observe that these cuts are similar to the Benders cuts introduced before. However, the difference between $k_C^k$ and $|C|$ can be larger than one and thus these cuts are in general distinct.

Note that if there are several vehicles with the same capacity, these constraints need only be computed once per capacity variant. As the graph $G^C$ is typically sparse it is reasonable to search for all maximal cliques. Unfortunately, in most practical cases only few such constraints can be identified. We add all of them to the initial formulation.

Again, we also add this type of constraints to the compact model using the sum of outgoing arcs instead of the assignment variables.

### 4.1.5. Computing a Lower Bound on the Tour Duration

We compute for each node $i \in N$ the minimal time required to reach the next vertex, i.e., $t_{i}^{\min} = \min_{(i,j) \in A} t_{ij}$. If we consider a subset $I \subseteq P$ of the requests (given by the respective pick-up nodes), we can compute a lower bound on the time required to serve all requests as follows:

$$t_{R}^{\min} = t_{0}^{\min} + \sum_{i \in I} (t_{i}^{\min} + d_{i} + t_{n+i}^{\min} + d_{n+i})$$

(65)

This relaxation gives us a (weak) lower bound on the time required to serve the requests in $I$. We use this value to state the following constraints in the Benders master-problem:

$$t_{0}^{\min} + \sum_{i \in P} y_{i}^k (t_{i}^{\min} + d_{i} + t_{n+i}^{\min} + d_{n+i}) \leq T^k \quad \forall k \in K$$

(66)

This bound can be improved in certain cases. If $t_{i}^{\min}$ and $t_{n+i}^{\min}$ refer to the same target vertex $v'$ (i.e., $t_{i}^{\min} = t_{i,v'}$ and $t_{n+i}^{\min} = t_{n+i,v'}$), we consider the closest successors for $i$ and $(n+i)$ excluding $v'$. We then choose the successor nodes resulting in the combined shorter distance $t_{i}^{\min} + t_{n+i}^{\min}$ and update the $t_{i}^{\min}$ values accordingly. If neither $i$ nor $(n+i)$ has an outgoing arc to a vertex different from $v'$, then the request is infeasible. This type of constraints is not considered for the compact model since tour duration restrictions are already explicit there.

### 4.2. Implementation of the Decomposition Approaches

The decomposition approach introduced in Section 3.3 can be implemented using logic-based Benders decomposition (LBBD) or Branch-and-Check (BaC). Algorithm 4.1 shows the basic functionality of the LBBD algorithm (ignoring Lines 13 to 18 for the moment). Remember that our decomposition approach only uses feasibility cuts, i.e., the subproblems do not directly contribute to the master-problem’s objective function. As already discussed, this means that LBBD either terminates with an optimal solution or no solution at all. BaC, on the other hand, relies on regular B&C which means that it computes lower
and upper bounds and tries to close the gap between them. Therefore, it usually provides a feasible solution prior to proving optimality. To make this also possible for LBBD we employ a repair heuristic (Line 14) to derive feasible solutions from intermediate infeasible master assignments, possibly even closing the optimality gap allowing premature termination. Details on the used repair heuristic will be given in Section 4.2.2.

1 feasible ← true;
2 \( j \leftarrow 0; \) // Iteration counter
3 repeat
4 \( j \leftarrow j + 1; \)
5 feasible ← true;
6 solve master-problem;
7 foreach \( k \in K \) do
8 if \( \text{sub}(k, I_k^j) \) is infeasible then
9 add Benders cuts to the master-problem;
10 feasible ← false;
11 end
12 end
13 if feasible = false then
14 repair(); // construct feasible solution heuristically
15 // check whether optimality gap could be closed
16 if \( \text{obj(master-problem)} = \text{obj(repair)} \) then
17 feasible ← true;
18 end
19 until feasible = true \( \lor \) time limit reached;
20 if feasible = true then
21 // optimal solution found
22 else
23 // potentially suboptimal, repaired solution
24 end

\textbf{Algorithm 4.1:} Logic-based Benders algorithm

In Figure 2 we illustrate a simple iteration of the Benders algorithm. We consider three requests and one vehicle. The instance properties are shown in Figure 2a. To keep the example simple without terminating immediately we do not consider strengthening inequalities for the master-problem here. Time window tightening and arc elimination have been applied to obtain a smaller graph, see Figure 2b. Initially the master-problem assigns all requests to the single vehicle. This turns out to be infeasible. When we try to reduce the identified infeasible assignment \{1, 2, 3\} we find out that subsets \{2, 3\} (Figure 2c) and \{1, 3\} (Figure 2d) are IISs of minimum cardinality. However, subset \{1, 2\} (Figure 2e) is feasible. Therefore, we add Benders cuts that prevent requests 2 and 3 as well as 1 and 3 to be in the same tour, respectively. In the second
master iteration requests 1 and 2 are assigned to the vehicle. Now we are able to identify a feasible tour for the subproblem (Figure 2f) and the algorithm terminates with an optimal solution serving requests 1 and 2 but rejecting request 3.

4.2.1. Benders Cuts

In our experiments in Section 5 we will consider four types of sets upon which we build Benders infeasibility cuts: variant simple uses the unmodified set, aIIS uses all IISs that can be obtained from the initial set, mIIS uses all IISs of minimal cardinality, and gIIS uses heuristically computed IISs. The IISs for variants aIIS and mIIS are computed using bottom-up enumeration by extending an initially empty set, using the assigned requests, until it becomes infeasible (including appropriate pruning for the minimum cardinality variant). Variant gIIS uses no special ordering of the requests. According to our implementation this means that we apply Algorithm 3.1 once in ascending and once in descending order of the request indices to obtain two IISs. If both turn out to be equivalent, only one Benders cut is added. As there is no connection between the order of request indices and their properties this means that there is no strategic decision involved.

4.2.2. Repair Heuristic

As mentioned before we are using a repair heuristic to construct feasible solutions based on infeasible assignments obtained from the Benders master-problem. To this end we consider input sets $I^k$, $k \in K$, some of which might be infeasible. We construct a solution for each vehicle by assigning requests to it one at a time. If a request can be served by the vehicle, it is assigned to that vehicle, otherwise skipped. We first try to insert the requests selected by the Benders master-problem. Afterwards we consider the unassigned requests. Requests that could not be served are added to the pool of unassigned requests and might be used by the remaining vehicles. Algorithm 4.2 provides details.

Note that the order in which the requests are considered has a significant impact on the outcome of the algorithm. However, the Benders master-problem already makes a selection which provides (especially in later iterations) a reasonable starting assignment from which typically only few requests need to be removed. Hence, we avoid sorting the requests to save computation time since repair operations need to be performed rather frequently and thus execution speed is critical. For the same reasons we avoid a second pass over the vehicles that might be necessary due to freed up requests.

4.2.3. Subproblem

In Section 3.3 we introduced the MILP and CP formulations for the Benders subproblems. The former is a compact model and can be implemented in a straightforward way. For the CP model we additionally incorporated the custom branching heuristic presented in [3]. Their approach branches on the successor variables $s[i]$, prioritizing variables with minimum cardinality domains. Ties are broken by counting the number of appearances of each value within all minimum
Figure 2: A simple Benders iteration without strengthening inequalities for the master-problem. There exists no feasible tour visiting all three requests. The combination of request 3 with either of the remaining two turns out to be infeasible. Requests 1 and 2 can be served together which also constitutes the single optimal solution.
Input: Set $P$ of requests, identified by the pick-up locations
Input: Sets $I^k$, $k \in K$ of potentially infeasible assignments
Output: Pairwise disjoint feasible sets $I^k$, $k \in K$ of requests assigned to the vehicles

1. $F \leftarrow P \setminus \bigcup_{k \in K} I^k$; // set of unassigned requests
2. foreach $k \in K$ do
3.     $\tilde{I}^k \leftarrow \emptyset$
4.     foreach $i \in I^k$ do
5.         if $\text{sub}(k, \tilde{I}^k \cup \{i\})$ is feasible then
6.             $I^k \leftarrow I^k \setminus \{i\}$;
7.             $\tilde{I}^k \leftarrow \tilde{I}^k \cup \{i\}$;
8.         end
9.     end
10.    foreach $i \in F$ do
11.        if $\text{sub}(k, \tilde{I}^k \cup \{i\})$ is feasible then
12.            $F \leftarrow F \setminus \{i\}$;
13.            $\tilde{I}^k \leftarrow \tilde{I}^k \cup \{i\}$;
14.        end
15.    end
16.    $F \leftarrow F \cup I^k$; // unused requests might be assigned to the other vehicles
17.    $I^k \leftarrow \tilde{I}^k$
18. end

Algorithm 4.2: Repair heuristic
cardinality domains, choosing the variable for which the sum of appearance counts of the values of its domain is maximal. We use no custom value selection heuristic and always pick the minimum value of the domain of the variable on which is branched.

4.3. Heuristic Boosting

Empirical tests have shown that the master-problem of the LBBD approach frequently finds good or even optimal solutions quite fast. Afterwards a significant amount of time is typically spent to close the relative gap between lower (LB) and upper (UB) bound, i.e., the optimality gap \( \frac{UB-LB}{LB} \). However, closing the gap might not be required to obtain an intermediate solution yielding high quality Benders cuts. The following sections describe our approaches exploiting this observation.

4.3.1. Gap Boosting

To reduce the time spent on closing the optimality gap of the Benders master-problem we terminate the solving process when the optimality gap falls below a certain threshold. This is done until no further Benders cuts can be found with this strategy. Then, we proceed with regular Benders iterations without premature termination, again until no additional Benders cuts can be identified. Thus, we still obtain an optimal solution but might save some time that is “wasted” on closing the optimality gap.

The difficulty is to choose an ideal threshold for early termination, especially in earlier iterations. Allowing a large initial gap has higher potential for speedup but can also lead to significantly worse solutions. On the other hand, a smaller gap is more likely to provide a good solution but usually requires more time.

To overcome the limitations of using a single value we consider an adaptive approach: Initially we start with a large gap and then iteratively decrease it every time no further cuts can be identified. Going even a step further we allow the gap threshold to iterate in both directions. If no cuts can be identified using the current value, we decrease the threshold, otherwise we increase it.

Observe that the considered objective function is integral. Thus, it is also possible to use the absolute optimality gap \( UB - LB \) instead of the relative one. Preliminary computational tests have shown that the behavior is roughly the same when choosing comparable gap sizes.

4.3.2. Time Limit Boosting

Early termination based on the optimality gap helps to reduce time spent in the Benders master-problem. However, the exact amount of time that is used still might vary substantially. As an alternative we may directly limit the time allowed to be spent on finding a solution to the master-problem. However, a fixed time limit might not accommodate for the increasing size of the master-problem due to the Benders cuts. To deal with this we again consider an adaptive approach. Starting with a small time limit we switch to the next larger
one whenever no additional cuts can be found until we finally allow the master-problem to use the total remaining time. Again we can extend this procedure by allowing the time limit to be not only increased but also decreased.

4.3.3. Boosting the Subproblems

We further tried to improve the subproblems by using heuristics. To this end we employed a simple iterative algorithm that attempts to find a feasible route for the requests assigned to a vehicle during the Benders iterations. The algorithm constructs a route by inserting nodes sequentially, prioritizing those with smallest amount of time left in their service window or the least remaining ride time. Due to the heuristic nature of the algorithm we can accept the result if a feasible route has been found. However, if no route can be computed, we still need to check with an exact approach whether the result is correct. Preliminary tests have shown that the employed heuristic required reevaluations quite frequently, outweighing the provided speedup from the positive cases. Nevertheless, it might be possible to identify a heuristic that provides a good balance between execution speed and quality of results in future work.

5. Computational Study

In this section we are going to present the computational results for the considered algorithms with their variants. We start by giving details on the used test instances and the motivation for their selection. Then we provide details on the actually used configuration. Finally, we present the obtained results.

5.1. Test Instances

We first had a look at instances used in other works considering exact solution approaches for the DARP. Unfortunately they only consider rather low numbers of vehicles. In Cordeau [8], for example, instances range from two to four vehicles with which all requests can be served. To obtain an instance that is suitable for our scenario one would need to remove vehicles, which would lead to trivial or rather simple instances. Under these premises we decided to generate new instances but stick to the generation procedure mentioned in [8]. We first place vertices randomly on a $20 \times 20$ grid; the depot is located in the center of this grid at coordinate $(0,0)$. Travel times between the vertices are set to the rounded-up euclidean distance between the corresponding points. For each instance with $n$ requests the first $n/2$ requests are considered to be outbound requests and the remaining are inbound requests. For the former we fix the time window at the drop-off location and derive the time window at the pick-up location as described in Section 4.1.1 and for the inbound requests we fix the time window at the pick-up location and derive the time window at the drop-off location.

For outbound requests we set the time window at the drop-off location $(n+i)$. To this end we first choose $e_{n+i}$ uniformly at random in the interval $[t_{0i} + d_i +$
and then set \( l_{n+i} = e_{n+i} + 15 \). This guarantees that the time window has a fixed length of 15. Furthermore, it ensures that we cannot arrive before the earliest possible service time and it is always feasible to return to the depot. Similarly, we choose for inbound requests \( e_i \) of pick-up node \( i \) in the interval \([t_0, T - t_{n+i}, 2n+1 - d_{n+i} - t_i, n+i - d_i - 15]\) and set \( l_i = e_i + 15 \). The remaining time windows are then tightened as described in Section 4.1.1. For each request we assume unit load of \( q_i = -q_{n+i} = 1 \) and the service duration is \( d_i = d_{n+i} = 3 \) for \( i \in P \). The maximum user ride time is set to \( L = 30 \). We consider different amounts of homogeneous vehicles with capacity \( Q^k = 3 \) and route durations \( T^k = T \). Table 1 provides an overview of the properties of the generated test instances.

5.2. Computational Results

In this section we are going to present the computational results of our algorithms obtained on the introduced instances. The test runs have been executed on an Intel Xeon E5540 with 2.53 GHz. The execution time limit has been set to 7200 seconds and the memory limit to 4GB RAM. Test runs have been executed using CPLEX 12.6.3 with a single thread using dual simplex and traditional B&C. The CP part has been implemented using Gecode 4.4.0 [18], also utilizing only a single thread for each test run. For the Bron-Kerbosch algorithm we used the implementation from Boost 1.60.0. Since objective values are known to be integral runs use an absolute optimality gap of \( 1 - n \cdot \varepsilon \), where \( \varepsilon \) is the reduced-cost optimality tolerance of the MILP solver.

Table 1 gives an overview of the computation times and lower bounds of the algorithms without the use of heuristic boosting techniques. The subproblems are solved using the MILP model. If an instance could not be solved to optimality within the time limit, the entry is marked with “TL”. Similarly, test runs that terminated due to the memory limit are marked with “ML”. Column \( LB^* \) reports the best lower bound obtained across all algorithm variants (including also the heuristically boosted ones). Entries in column \( LB^* \) are marked bold if the corresponding solution has been shown to be optimal.

The compact model shows a rather bad performance both w.r.t. computation time and obtained solutions. Among the decomposition approaches it can be seen that results differ greatly depending on the refinement of the Benders cuts. As one would expect unrefined Benders cuts deliver the worst results. Using all IISs gives slightly better results but is typically outperformed by the variant using only minimum cardinality IISs. This is due to the high amount of IISs obtained from the initial set. The high number of resulting Benders cuts also leads to memory problems when dealing with large and/or difficult instances. Fortunately, the number of minimum cardinality IISs is considerably smaller while providing a similar restriction of the search space. The greedy construction of IISs (gIIS) also works quite well. Having no guarantee on the quality of the obtained Benders cuts is compensated by the much faster computation times for finding Benders cuts (see also Table 4). Comparing the LBBD and BaC approaches both provide similar results, with the latter being faster in most cases.
<table>
<thead>
<tr>
<th>Instance</th>
<th>K</th>
<th>T</th>
<th>Q</th>
<th>L</th>
<th>LB *</th>
<th>CM</th>
<th>LBBD</th>
<th>BaC</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td>26</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>aIIS</td>
<td>26</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>mIIS</td>
<td>26</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>gIIS</td>
<td>26</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
</tr>
</tbody>
</table>

Table 1: Overview of the instance properties and the computation times of the unboosted algorithm variants. Column LB* shows the best known lower bounds. Bounds corresponding to provably optimal solution values are marked bold. Columns CM, LBBD, and BaC show the computation times and lower bounds for the compact model, the LBBD decomposition algorithm, and the BaC decomposition algorithm, respectively. For the decomposition approaches four kinds of sets have been used to obtain Benders cuts: simple uses unrefined cuts, aIIS uses all IISs, mIIS uses all IISs of minimum cardinality, and gIIS uses two heuristically computed IISs. Instances that could not be solved within the time limit of 2 hours are marked with “TL” and test runs that terminated due to the memory limit are marked with “ML.” For each instance the computation times of the fastest algorithm(s) and best bounds obtained are marked bold.
The results indicate that finding good or even optimal solutions becomes harder the scarcer the available resources are compared to the demand, i.e., the higher the number of requests combinations that needs to be tested and excluded the more difficult is the instance. This relation can be expected as it increases number of combinatorial possibilities from which the algorithm needs to find an optimal one. In particular, the Benders algorithms are required to exclude a much higher amount of infeasible assignments until only feasible options remain. Relaxations that bound the tour size (see Sections 4.1.4 and 4.1.5) help to reduce this effect.

In addition to solving the subproblems via an MILP solver we also investigated the CP approach. In general, computation times are superior but for some instances severe outliers occurred where single subproblems required more time than half an hour. To still profit from the mostly faster CP variant we further investigated a combination working as follows. We first allow the CP solver to work on the subproblem for half a second. If afterwards no result is available, we use the MILP model instead which is in general slightly slower but much more consistent featuring no practically noticeable outliers. Of course we do not want to waste the work done by the CP solver. Therefore, we build the MILP model using the variables and domains of the CP model. Thus, we may profit from the results of constraint propagation possibly yielding a smaller model. The results for the combined CP-MILP subproblems are shown in Table 2. It can be seen that the computation times improve upon those from the pure MILP approach for all test instances. Additionally, further instances could be solved to optimality. The BaC approaches are still faster than the LBBD ones and in general respond a bit better to the faster subproblems. This is related to the fact that the BaC approaches typically spend more time on solving the subproblems (see also Table 4).

In the remainder of this section we will restrict results provided in the tables to Benders cut refinement variants mIIS and gIIS which have clearly shown to be superior compared to the other two. The primary motivation behind this step is to keep tables readable. Experiments have still been conducted on all variants to verify that the relation among them stays as expected. For similar reasons we only consider combined CP-MILP subproblems here.

Table 3 provides details on the computation times and lower bounds when using the heuristic boosting techniques introduced in Section 4.3. For both boosting techniques we consider a purely iterative approach (it) and a variant that adjusts in both directions (ud). In the latter variant the time limit (gap) is increased (decreased) whenever no more cuts can be identified and decreased (increased) if cuts have been added. For the time limit variant we consider time limits of 5, 10, and 30 seconds in increasing order. If no more cuts are found using the highest limit, the algorithm is allowed to use the total remaining time in the final stage. The gap variant uses relative optimality gaps of 0.1, 0.05, 0.025, and 0 in decreasing order.

With both boosting techniques our algorithms solve additional instances to optimality or at least provide significantly better lower bounds than the unboosted variants. There is no clear winner among the two boosting techniques,
<table>
<thead>
<tr>
<th>Instance</th>
<th>LBBD</th>
<th>BaC</th>
<th>LBBD</th>
<th>BaC</th>
</tr>
</thead>
<tbody>
<tr>
<td>30N-5K-A</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>30N-5K-B</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>30N-5K-C</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 2: Overview of the computation times of the unboosted algorithm variants using a CP-MIP combination for the subproblems. Column LB* shows the best known lower bounds. Bounds corresponding to provably optimal solution values are marked bold. Columns LBBD, and BaC show the computation times and lower bounds for the LBBD decomposition algorithm and the BaC decomposition algorithm, respectively. For the decomposition approaches four kinds of sets have been used to obtain Benders cuts: simple uses unrefined cuts, aIIS uses all IISs, mIIS uses all IISs of minimum cardinality, and gIIS uses two heuristically computed IISs. Instances that could not be solved within the time limit of 2 hours are marked with “TL” and test runs that terminated due to the memory limit are marked with “ML”. For each instance the computation times of the fastest algorithm(s) and best bounds obtained are marked bold.
however, the time limit boosting outperforms the gap boosting variant on larger instances. Purely iterative adjustments and the variant allowing adaptation in both directions work equally well. We also conducted preliminary tests using a single termination criterion. However, those turned out to be much less robust than the iterative variants. For some instances they work exceptionally well but this is paid for exceedingly on the remaining ones.

Table 4 gives details on the characteristics of the decomposition approaches. In favor of a more readable table size we decided to only consider the time boosting variant here since the behavior of the gap boosting is similar. We provide the number of iterations, the ratio between the computation time spent in the master-problem and those spent in the subproblems (\(\frac{t_{\text{master}}}{t_{\text{sub}}}\)), and the number of added Benders cuts. The minimum number of required cuts is marked bold per row, however, note that only comparing algorithms that completed within the time limit is meaningful. For the LBBD approaches the number of iterations corresponds to the number of times the master-problem has been solved whereas for the BaC approaches it is equal to the number of times the separation routine has been called. The first outstanding observation is that variant mIIS has a much smaller master-sub ratio than gIIS, i.e., more time is spent in the subproblems. This is caused by the costly enumeration procedure to find the minimum cardinality IISs. However, due to the better quality of the cuts the number of necessary master iterations is typically smaller. Also variant mIIS often needs less cuts to find an optimal solution although it might add more cuts than variant gIIS per iteration. These properties can be observed for both decomposition approaches independent of the used boosting technique. Comparing the decomposition algorithms, it becomes evident that the BaC approaches spend in general much more time in the subproblems than the LBBD approaches. This is related to the fact that LBBD only uses optimal master solutions for adding cuts whereas BaC uses all integral solutions found during B&C.

Table 5 provides details on the gaps w.r.t. to the best known bounds. Columns LB* and UB* report the best known lower and upper bounds obtained across all algorithms. We compute LB gap by 100 \(\cdot\) \(\frac{\text{LB} - \text{LB}}{\text{LB}^*}\) and UB gap by 100 \(\cdot\) \(\frac{\text{UB} - \text{UB}}{\text{UB}^*}\), where LB and UB are the lower and upper bound obtained by the considered algorithm and LB* and UB* are the respective best bounds known. Smallest gaps per instance are marked bold.

Observe that the heuristically boosted LBBD as well as the BaC algorithms have small gaps w.r.t. the lower bounds. However, they mostly do not perform so well when it comes to finding good upper bounds. This is an expected consequence of the shift towards prioritizing the search for intermediate feasible solutions.

6. Conclusion

In this work we considered a variant of the DARP aiming at serving a maximal number of requests rather than minimizing routing costs. We proposed a
Table 3: Results of the heuristic boosting techniques. Column computation time reports the time consumed and column LB provides the value of the lower bound. Column standard shows results of the algorithms without boosting, “gap (rel) - it” shows results for boosting with purely iterative adjustments whereas “gap (rel) - ud” adapts the limit in both directions. Similarly, “time - it” and “time - ud” report results for boosting with reduced time limit. Smallest computation times and best bounds per instance are marked bold.
Table 4: Characteristics of the decompositions performed. For the BB algorithm, this corresponds to the number of iterations the algorithm completed. For the BB-Bound algorithm, the number of times the separation routine has been called. For the BB-Bound cuts, it is equal to the relative number of times the separation routine in the master-problem produced new cuts for the subproblems (not in the tableau). The first column (entry) shows the number of iterations the master-problem was solved. The last column (cuts) shows the total number of Benders cuts that have been added.

<table>
<thead>
<tr>
<th>Instance</th>
<th>iterations</th>
<th>cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>master-sub ratio</td>
<td>LBB</td>
</tr>
<tr>
<td>unboosted</td>
<td>time - it</td>
<td>time - wtl</td>
</tr>
<tr>
<td></td>
<td>msIS gIS</td>
<td>msIS gIS</td>
</tr>
<tr>
<td>Instance</td>
<td>LB</td>
<td>UB</td>
</tr>
<tr>
<td>----------</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>30N,4K_A</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>30N,4K_B</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>30N,4K_C</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>30N,5K_A</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>30N,5K_B</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>30N,5K_C</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>40N,4K_A</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>40N,4K_B</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>40N,4K_C</td>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>40N,5K_A</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>40N,5K_B</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>40N,5K_C</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>44N,4K_A</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>44N,4K_B</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>44N,4K_C</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>44N,5K_A</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>50N,4K_A</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>50N,4K_B</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>50N,4K_C</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>50N,5K_A</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>50N,5K_B</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>50N,5K_C</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>60N,4K_A</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>60N,4K_B</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>60N,4K_C</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>60N,5K_A</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>60N,5K_B</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>60N,5K_C</td>
<td>58</td>
<td>58</td>
</tr>
</tbody>
</table>
simple compact reference model and a decomposition approach. The master-
problem has been formulated as MILP model and the subproblems have been 
stated as MILP model and also as CP model. We reviewed preprocessing tech-
niques from the literature and suggested improvements. Additionally, we pro-
vided novel preprocessing methods. In the computational study we solved the 
decomposition model using LBBD and BaC. Subproblems have been solved using 
MILP and CP. The combination of both turned out to be most successful.
We considered four types of sets upon which we build the used Benders feasibility 
cuts. Experiments have shown that a fast greedy approach and the enumeration 
of all minimum cardinality IISs work best. To speed up solving of the Benders 
master-problems we considered heuristic boosting techniques based on the re-
ative optimality gap and the time limit. Both techniques helped to improve 
the Benders algorithm allowing to find further solutions and to improve compu-
tation times. The suggested boosting techniques are more generally promising 
also in the context of LBBD and BaC approaches for other applications.

6.1. Future work

In practical applications not all requests might be equally important. Thus, a 
natural extension of the considered DARP variant would be to consider weights 
for the requests. Due to the focus on request selection we do not consider routing 
costs in the objective. The easiest extension would be to consider cost optimal 
routing for each vehicle separately, keeping the problem complexity more or 
less the same. However, this may lead to globally suboptimal solutions since 
selecting different requests might reduce the routing costs while retaining the 
number of served requests. Considering globally optimal routing costs makes 
the problem much more challenging since the objectives of the subproblems now 
influence the master-problem and thus also Benders optimality cuts are needed.
Moreover, also other second-level objectives might be worth considering like 
additional user-inconvenience considerations, e.g., limiting the direct route to 
actual route ratio. Additionally, investigating further strategies and testing with 
heterogeneous vehicles would be interesting.

We considered four variants for computing infeasible sets that serve as ba-
sis for the Benders feasibility cuts. Our algorithms are based on enumeration 
and greedy strategies. In this respect it would be interesting to design prob-
lem specific approaches that are able to find structures close to the minimum 
cardinality IISs requiring less time than enumeration. The work by Hämé and 
Hakula [21] could serve as a starting point for research in this direction.

References

pickup and delivery problem with time windows. Operations Research, 59 

dial-a-ride problem and a multiobjective solution using simulated anneal-


