Technical Report AC-TR-16-007
first version: Sep 2016
updated: Nov 2017
updated: Apr 2018

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 and Günther Raidl

Institute of Logic and Computation, TU Wien, Vienna, Austria

April 11, 2018

Todays 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.

1. Introduction

The dial-a-ride problem (DARP) considers the design of vehicle routes for a set of customers who specify transportation requests from origin (pick-up) to destination (drop-off) points. Users typically impose time-windows with respect to 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. [30], Cordeau [10], 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 request corresponds to the opposite case, i.e., a customer who wants to return to his/her starting location. According to the survey presented in [31] 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, see [11, 13, 32]. 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. 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 have to be served using a single vehicle and in the latter 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 have 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 number 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 have to be taken into account to provide reasonable service conditions.

We consider solution algorithms based on logic-based Benders decomposition (LBBD) (see Hooker and Ottosson [27]) and branch-and-check (BaC) (see Thorsteinsson [42]). The Benders master problem focuses on the selection of requests and their assignment to vehicles. It is modeled as integer linear program and enhanced by different subproblem relaxations to speed up convergence. For solving the Benders subproblems, which correspond to the route planning tasks, we consider mixed integer linear programming (MILP) as well as constraint programming (CP) approaches. In particular, we also present a hybrid approach that combines MILP and CP. Several strategies for constructing Benders cuts are studied. We consider cuts derived from greedily obtained minimal infeasible request subsets, the full set of all minimal infeasible request subsets, as well as the set of all minimum cardinality infeasible request subsets and compare them to the unrefined cuts that are directly obtained from the subproblem assignments. Moreover, we consider heuristic boosting techniques to possibly speed up the solution process. To this end we terminate the master problem prematurely according to a specific termination criterion and use the suboptimal solution to derive Benders cuts. As soon as no further cuts can be obtained this way, we fall back to solving the master problem to optimality and continue with regular Benders iterations. This is necessary to obtain a provably optimal solution. As termination criterion we consider a decreasing sequence of thresholds for the optimality gap and an increasing sequence of time limits. Employing an adaptive approach we start at the first element of the sequence and move to the next one whenever no further cuts can be found with the current termination condition. A more flexible approach allows traversing the sequence in both directions, depending on whether cuts could be obtained or not. The suggested algorithms are tested extensively on a novel set of benchmark instances as well as on instances from the literature.

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 variant, including a complexity discussion. In terms of the formal specification we provide a compact reference model that is a straightforward extension of the MILP from [10] for the tour-length-minimization DARP. In the main part we present the details of our decomposition approaches; including important implementation details. Finally, we discuss computational results on various test instances and 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 [33] that deals with the static single-vehicle variant. Sexton and Bodin [40, 41] 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 [6] 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, neither method can guarantee optimal solutions. Later on, this approach for the multi-vehicle problem has been refined by using so-called “mini-clusters”, see [15, 14]. The most recent contribution by Ioachim et al. [28] 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 [45] 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.

Berbeglia et al. [5] and Häme and Hakula [24] focus on feasibility checking of DARP instances. Similarly, also the large neighborhood search by Jain and Van Hentenryck [29] has been tested as feasibility checking algorithm. 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 [11, 13] and Parragh et al. [32].

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 in terms of exact solution approaches; see Ropke and Cordeau [38] and Baldacci et al. [2], respectively. For further details consider the survey conducted in [32].

Recently, also revenue maximizing variants of the PDPTW have been considered. In Qiu and Feuerriegel [34] and Qiu et al. [35] each transportation request is assigned a profit. The goal is then to identify a subset of requests to be served with a given heterogeneous vehicle fleet that maximizes the revenue, i.e., sum of profits minus transportation cost. The problem is solved using a graph search algorithm as well as a maximum set partitioning formulation for the case of a homogeneous vehicle fleet. A similar scenario is also considered in Gansterer et al. [17] and solved with different metaheuristic approaches.

Somewhat related are also certain variants of the team orienteering problem. A contribution in this respect is from Baklagis et al. [1] who solve a variant considering pick-up and delivery with a branch-and-price approach.

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 [8] 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 such that 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 then brought to the customers. This kind of VRP arises for example in newspaper distribution. Raidl et al. [36, 37] 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. [36, 37] 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 as subproblems. 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. [36] 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. [37] 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 proposed by Hooker and Ottosson [27]. It builds upon the classical Benders decomposition (BD) [4] that was originally described to solve large linear programming (LP) problems having variables that can be partitioned into two subsets (x, y) such that the problem separates into one or more easier solvable subproblems on the x variables after fixing the y variables. Information regarding the solution to the subproblems is then incorporated into the master problem by means of Benders cuts. These cuts can be either *feasibility* or *optimality cuts*. The former reflect that the subproblems revealed the master assignment to be infeasible and prevent this assignment from occurring again. The latter provide a bound on the part of the objective function considered by the subproblems in case the assignment of the master problem is feasible. Infeasibility can be represented by an infinite bound. Master and subproblems are then repeatedly solved until optimality is proven by finding a feasible master solution that matches the bound of the subproblems.

In the original BD the subproblems are restricted to be LPs. Geoffrion showed in [22] 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.

LBBD extends this approach by allowing also integer variables and possibly nonlinearities in the subproblems. The Benders cuts are obtained by solving the *inference dual* of the subproblem, see [27]. The inference dual finds the tightest dual bound on the objective function that is implied by the constraints. Solving the inference dual yields a bounding function on the master problem's objective value that is tight for the current assignment. In contrast to the traditional BD there exists no (single) general systematic way to identify a strong bounding function for the Benders cuts. Thus, tailored cuts have to be identified with respect to the encountered subproblems.

LBBD has been applied effectively in several areas including planning and scheduling (Hooker [26], Hamdi and Loukil [23]), location problems (Fazel-Zarandi and Beck [16], Wheatley et al. [44]), survivable network design (Garg and Smith [20]), and vehicle routing (Cire and Hooker [8], Raidl et al. [36, 37]).

2.2. Branch-and-Check

The idea behind classical BD and LBBD is to iteratively (re)solve the master problem to optimality. Each obtained solution is used as basis for the subproblems, whose solving potentially gives rise to Benders cuts that are added to the master problem. However, repeatedly solving the master problem to optimality might not be necessary. Suboptimal solutions can be sufficient to derive relevant cuts for further progress. Ultimately, it seems reasonable to generate all Benders cuts within a single branch-and-cut (B&C) tree search for identified intermediate solutions.

This idea was first introduced in Hooker [25] and further examined in [42] and is also closely related to the concept of *Combinatorial Benders cuts* considered by Codato and Fischetti [9]. Thorsteinsson [42] 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 (integral) solution is identified within the B&C tree, the corresponding subproblems are derived and solved. Dependent on their solutions Benders cuts are added, possibly cutting off the current solution. Thus, the main difference to LBBD is that the master problem is solved only once and that Benders cuts are typically generated with respect to suboptimal assignments of the y variables inside a single B&C tree search.

When dealing solely with feasibility cuts, i.e., the solution of the subproblems has no influence on the objective value 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 (found in its last iteration) 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 nodes 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 $(i, n + i)$ such that $i \in P$ 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 $[e_i, l_i]$, $e_i < l_i$.

The set of arcs is defined as $A = \{(i, j) \mid (i = 0 \wedge j \in P) \vee (i, j \in P \cup D \wedge i \neq j \wedge i \neq n + j) \vee (i \in D \wedge j = 2n + 1)\}$. The non-negative travel time of arc (i, j) is t_{ij} and the maximum user ride time is denoted by $L > 0$. We are given a set of vehicles K and every vehicle $k \in K$ has a maximum capacity $Q^k > 0$ and a maximum route duration $T^k > 0$. 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. [3]) and we will show that the problem still remains $\mathcal{NP}$-hard under the modified scenario. Our proof is based on the traveling salesman problem (TSP) and the decision problem variant of the selective DARP (S-DARP-D). Both are provided in the following. The TSP is well-known to be $\mathcal{NP}$-hard, see Garey and Johnson [18].

Definition 1. *TSP [18]*
INSTANCE: Set C of m cities, distance $c_{ij} \in \mathbb{Z}_{>0}$ for each pair of cities $i, j \in C$, positive integer B.
QUESTION: Is there a Hamiltonian tour of C having length B or less?

Definition 2. *S-DARP-D*
INSTANCE: Selective DARP instance, positive integer n'.
QUESTION: Is there a feasible solution to the selective DARP serving at least n' requests?

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

Proof. We show $\mathcal{NP}$-hardness of the selective DARP via a reduction from the TSP to S-DARP-D. First, we create a request for each city in C setting $d_i = q_i = 0$ for the pick-up and drop-off locations. Moreover, a single vehicle with $Q^1 = T^1 = \infty$ is considered. For the pick-up nodes we set the time windows to $[0, B]$ and for the drop-off nodes we set the time windows to $[B+1, \infty]$. The maximum user ride time is assumed to be unrestricted, i.e., $L = \infty$. For i and j, both pick-up nodes, we set $t_{ij} = c_{ij}$. The remaining travel times are set to zero. There exists a Hamiltonian tour of C with length B or less iff the constructed S-DARP-D instance allows serving at least $|C|$ requests. $\square$

Corollary 1. *The selective DARP remains $\mathcal{NP}$-hard when the triangle inequality holds for the travel times t_{ij}; including the even more specific cases of the L_1 (rectilinear) metric and the L_2 (Euclidean) metric.*

Proof. Use the transformation stated above, but start from a TSP instance with the respective properties. The TSP is known to be also $\mathcal{NP}$-hard under these conditions, see Garey et al. [19]. $\square$

3.2. Compact Model

The following MILP model is a slightly modified variant of the one introduced in [10]. We are going to refer to it as compact model (CM). The difference is that we are maximizing the number of requests served, instead of minimizing travel costs.

We use binary variables x_{ij}^k for each arc $(i, j) \in A$ per vehicle $k \in K$. Moreover, variables B_i^k and Q_i^k 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_i^k to model the ride time of each request identified by its pick-up location $i \in P$ on vehicle $k \in K$.

$$\max \sum_{k \in K} \sum_{\forall (i,j) \in A : j \in P} x_{ij}^k \tag{1}$$

$$\sum_{k \in K} \sum_{(i,j) \in A} x_{ij}^k \leq 1 \qquad \forall i \in P, \tag{2}$$

$$\sum_{(i,j) \in A} x_{ij}^k - \sum_{(n+i,j) \in A} x_{n+i,j}^k = 0 \qquad \forall i \in P, \forall k \in K, \tag{3}$$

$$\sum_{j \in P} x_{0j}^k = 1 \qquad \forall k \in K, \tag{4}$$

$$\sum_{(j,i) \in A} x_{ji}^k - \sum_{(i,j) \in A} x_{ij}^k = 0 \qquad \forall i \in P \cup D, \forall k \in K, \tag{5}$$

$$\sum_{i \in D} x_{i,2n+1}^k = 1 \qquad \forall k \in K, \tag{6}$$

$$(B_i^k + d_i + t_{ij}) x_{ij}^k \leq B_j^k \qquad \forall (i,j) \in A, \forall k \in K, \tag{7}$$

$$(Q_i^k + q_j) x_{ij}^k \leq Q_j^k \qquad \forall (i,j) \in A, \forall k \in K, \tag{8}$$

$$
\begin{aligned}
&B_{n+i}^k - (B_i^k + d_i) = L_i^k && \forall i \in P, \forall k \in K, && (9)\\
&B_{2n+1}^k - B_0^k \leq T^k && \forall k \in K, && (10)\\
&e_i \leq B_i^k \leq l_i && \forall i \in N, \forall k \in K, && (11)\\
&t_{i,n+i} \leq L_i^k \leq L && \forall i \in P, \forall k \in K, && (12)\\
&\max\{0, q_i\} \leq Q_i^k \leq \min\{Q^k, Q^k + q_i\} && \forall i \in N, \forall k \in K, && (13)\\
&x_{ij}^k \in \{0,1\} && \forall (i,j) \in A, \forall k \in K. && (14)
\end{aligned}
$$

The objective function (1) determines the number of served requests by counting the selected arcs leading to pick-up nodes. Constraints (2) ensure that each request is served by at most one vehicle. These are the only differences to the original model. Constraints (3) guarantee that pick-up and drop-off of each request are served by the same vehicle. Equalities (4) to (6) ensure that each vehicle leaves the depot as well as each node it visits and that it finally returns to the depot. Constraints (7) and (8) enforce that the 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 (9) calculate the ride time for each request and inequalities (10) limit the route duration for each vehicle. The remaining inequalities ensure that the used variables stay within their respective domains.

The quadratic constraints (7) and (8) can be linearized as follows:

$$
\begin{aligned}
B_i^k + d_i + t_{ij} - M_{ij}^k(1 - x_{ij}^k) &\leq B_j^k && \forall (i,j) \in A, \forall k \in K, && (15)\\
Q_i^k + q_j - W_{ij}^k(1 - x_{ij}^k) &\leq Q_j^k && \forall (i,j) \in A, \forall k \in K, && (16)
\end{aligned}
$$

with the Big-M constants set to $M_{ij}^k = \max\{0, l_i + d_i + t_{ij} - e_j\}$ and $W_{ij}^k = \min\{Q^k, Q^k + q_i\}$, 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 a feasible tour exists.

$$
\begin{aligned}
&(master) \quad \max \sum_{k \in K} \sum_{i \in P} y_i^k && && (17)\\
&\sum_{k \in K} y_i^k \leq 1 && \forall i \in P, && (18)\\
&\text{Benders cuts} && \forall k \in K, && (19)\\
&y_i^k \in \{0,1\} && \forall k \in K, \forall i \in P. && (20)
\end{aligned}
$$

The master problem maximizes the number of requests that are served. Constraints (18) ensure that each request is assigned to at most one vehicle. The Benders cuts (19) will be provided by the subproblems. They are responsible for preventing infeasible assignments of requests. Furthermore, we will later augment this basic master problem by initially provided valid inequalities originating from a relaxation of the subproblem.

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 $\bar{\mathbf{y}}$ to the master problem we identify for each vehicle $k \in K$ the set $I^k = \{i \in P \mid \bar{y}_i^k = 1\}$ of assigned requests. Each of these sets results in an independently solvable subproblem $sub(k, I^k)$. The subproblems can be stated similarly 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 nodes $N^I = \{0, 2n + 1\} \cup P^I \cup D^I$. According to N^I we define the reduced arc set $A^I = A \setminus \{(i, j) \mid i \notin N^I \vee j \notin N^I\}$. The subproblems can now be modeled as follows:

$$(sub(k, I)) \quad \min 0 \tag{21}$$

$$\sum_{(i,j)\in A^I} x_{ij} = 1 \qquad \forall i \in P^I \cup D^I, \tag{22}$$

$$\sum_{j\in P^I} x_{0j} = 1, \tag{23}$$

$$\sum_{(j,i)\in A^I} x_{ji} - \sum_{(i,j)\in A^I} x_{ij} = 0 \qquad \forall i \in P^I \cup D^I, \tag{24}$$

$$\sum_{i\in D^I} x_{i,2n+1} = 1, \tag{25}$$

$$B_i + d_i + t_{ij} - M_{ij}^k (1 - x_{ij}) \le B_j \qquad \forall (i, j) \in A^I, \tag{26}$$

$$Q_k + q_j - W_{ij}^k (1 - x_{ij}) \le Q_j \qquad \forall (i, j) \in A^I, \tag{27}$$

$$B_{n+i} - (B_i + d_i) = L_i \qquad \forall i \in P^I, \tag{28}$$

$$B_{2n+1} - B_0 \le T^k, \tag{29}$$

$$e_i \le B_i \le l_i \qquad \forall i \in N^I, \tag{30}$$

$$t_{i,n+i} \le L_i \le L \qquad \forall i \in P^I, \tag{31}$$

$$\max\{0, q_i\} \le Q_i \le \min\{Q^k, Q^k + q_i\} \qquad \forall i \in N^I, \tag{32}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A^I. \tag{33}$$

The objective function (21) 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 locations are visited by the same vehicle since we only consider one vehicle. It is sufficient to use constraints (22) for ensuring that the pick-up and drop-off locations of all assigned requests are visited. The remaining parts stay the same.

In addition to the MILP formulation we also provide a CP model similar to the one introduced in [5] but restricted to the single vehicle case and slightly adjusted. To formulate element constraints we define a bijective function $\pi \colon N^I \to \{0, \ldots, 2 \cdot |I| + 1\}$

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 $\{|I|+1, \ldots, 2 \cdot |I|\}$ such that $\pi(i) = j$ iff $\pi(i+n) = j + |I|$, $\forall i \in D^I$. Accordingly, we define sets $\widetilde{P}^I = \{\pi(i) \mid i \in P^I\}$, $\widetilde{D}^I = \{\pi(i) \mid i \in D^I\}$, and $\widetilde{N}^I = \widetilde{P}^I \cup \widetilde{D}^I \cup \{0, 2 \cdot |I| + 1\}$. Additionally, we specify an appropriately reduced travel time matrix $\widetilde{t}_{ij} = t_{\pi^{-1}(i)\pi^{-1}(j)}$, $\forall (i,j) \in \widetilde{N}^I \times \widetilde{N}^I$ and load vector $\widetilde{q}_i = q_{\pi^{-1}(i)}$, $\forall i \in \widetilde{N}^I$. 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 $s[i]$, $\forall i \in \widetilde{N}^I \setminus \{2 \cdot |I| + 1\}$, load variables $q[i]$, $\forall i \in \widetilde{N}^I$, and beginning-of-service time variables $b[i]$, $\forall i \in \widetilde{N}^I$. The model reads as follows:

$$
\begin{aligned}
&(sub(k, I)) && \\
&\text{allDifferent}(s), && && (34)\\
&b[i] + \widetilde{t}_{i,|I|+i} + d_{\pi^{-1}(i)} \leq b[|I| + i] && \forall i \in \widetilde{P}^I, && (35)\\
&b[i] + \widetilde{t}_{i,s[i]} + d_{\pi^{-1}(i)} \leq b[s[i]] && \forall i \in \widetilde{N}^I \setminus \{2 \cdot |I| + 1\}, && (36)\\
&b[i+n] - (b[i] + d_{\pi^{-1}(i)}) \leq L && \forall i \in \widetilde{D}^I, && (37)\\
&b[2 \cdot |I| + 1] - b[0] \leq T^k, && && (38)\\
&q[i] + \widetilde{q}_{s[i]} = q[s[i]] && \forall i \in \widetilde{N}^I, && (39)\\
&s[i] \in \{j \mid (i,j) \in A^I\} && \forall i \in \widetilde{N}^I \setminus \{2 \cdot |I| + 1\}, && (40)\\
&\text{domain}(b[i], e_{\pi^{-1}(i)}, l_{\pi^{-1}(i)}) && \forall i \in \widetilde{N}^I, && (41)\\
&\text{domain}(q[i], \widetilde{q}_i, Q^k) && \forall i \in \widetilde{P}^I, && (42)\\
&\text{domain}(q[i], 0, Q^k + \widetilde{q}_i) && \forall i \in \widetilde{D}^I, && (43)\\
&q[0] = 0, && && (44)\\
&q[2 \cdot |I| + 1] = 0. && && (45)
\end{aligned}
$$

Constraints (34) 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 (35). Constraints (36) model the time needed to travel from a node to its immediate successor. Inequalities (37) and (38) restrict the ride time and tour duration appropriately. The load restrictions are considered by constraints (39). 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_j^k the requests assigned to vehicle $k \in K$ and by $\overline{K}_j = \{k \in K \mid sub(k, I_j^k) \text{ is infeasible}\}$ the set of vehicles for which the subproblem turns out to be infeasible. The corresponding Benders cuts are:

$$\sum_{i \in I_j^k} y_i^k \leq |I_j^k| - 1 \qquad \forall k \in \overline{K}_j. \tag{46}$$

These basic cuts, however, frequently can be strengthened as it is likely that the infeasibility is caused by a proper subset of the assigned requests. Similar to the notation used in [9] we classify sets I_j^k as follows:

Definition 3. *We call a set of requests I_j^k* infeasible *iff subproblem $sub(k, I_j^k)$ is infeasible and* feasible *otherwise.*

Definition 4. *We call an infeasible set of requests I_j^k* irreducible infeasible set (IIS) *iff the removal of any request turns it into a feasible set. Otherwise, we call I_j^k* reducible infeasible set*.*

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_j^k. All such sets are by definition minimal and, thus, lead to non-dominated cuts within this class of cuts. Note that the IISs with respect to a given base set can have different cardinality. For practical reasons it makes sense to prefer smaller sets when the number of IISs gets large. Moreover, each Benders cut prevents assignments that are supersets with respect to 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 (similar to what is done in [26]) 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 the resulting set is still infeasible. If this is the case, we keep the smaller set, otherwise we proceed with the next request.

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

Input: Set I of requests and vehicle $k \in K$ such that $sub(k, I)$ is infeasible.

foreach $i \in I$ **do**
  if $sub(k, I \setminus \{i\})$ *is infeasible* **then**
    $I = I \setminus \{i\}$;
  end
end
return I; `// I is now an IIS`

Algorithm 3.1: Greedy Set Reduction

times by just keeping the natural order of the requests. However, to decrease the chances of ending up with bad results we apply the greedy reduction twice, 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 5. *We define a partial order on the vehicles denoted by* $\leq^{\mathrm{k}}$*:*

$$k_1 \leq^{\mathrm{k}} k_2 \Leftrightarrow (Q^{k_1} \leq Q^{k_2}) \wedge (T^{k_1} \leq T^{k_2}) \qquad k_1 \in K, k_2 \in K.$$

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

$$\sum_{i \in I_j^k} y_i^{k'} \leq |I_j^k| - 1 \qquad \forall k \in \overline{K}_j, \forall k' \in K : k' \leq^{\mathrm{k}} k. \tag{47}$$

4. Algorithmic Framework

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

4.1. Preprocessing

In this section we describe the used preprocessing techniques. They are based on the concepts introduced in [10]. We point out our modifications.

4.1.1. Time-Window Tightening

In [10] 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 requests to $e_{n+i} \leftarrow \max\{e_{n+i}, e_i + d_i + t_{i,n+i}\}$ and to $l_{n+i} \leftarrow \min\{l_i + d_i + L, l_{n+i}\}$.

The time windows on depot copies 0 and $(2n+1)$ are set to $e_0 = e_{2n+1} \leftarrow \min_{i \in P \cup D}\{e_i - t_{0i}\}$ and $l_0 = l_{2n+1} \leftarrow \max_{i \in P \cup D}\{l_i + d_i + t_{i,2n+1}\}$.

We modify this slightly since this might lead to unwanted effects when requests are too close to the depot, i.e., $t_{0i} > e_i$ 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 $e_i \leftarrow \max\{e_i, t_{0i}\}$ 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 [10] 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 $(i, n+j)$ with $i, j \in P$ is infeasible if path $(j, i, n+j, n+i)$ is infeasible as there is no other feasible path using that arc while serving i and j
- symmetric to the previous condition arc $(n+i, j)$ with $i, j \in P$ is infeasible if path $(i, n+i, j, n+j)$ is infeasible
- arc (i, j) with $i, j \in P$ is infeasible if paths $(i, j, n+i, n+j)$ and $(i, j, n+j, n+i)$ are both infeasible as the path can only be infeasible due to the arc itself or a time window violation when reaching either of the drop-off locations; visiting further nodes may only increase the degree of violation
- symmetric to the previous condition arc $(n+i, n+j)$ with $i, j \in P$ is infeasible if paths $(i, j, n+i, n+j)$ and $(j, i, n+i, n+j)$ are both infeasible

When checking the feasibility of paths, we also need to compute the forward time slack. In [12] the forward time slack F_i at node i in a path from i to q is computed as follows:

$$F_i = \min_{i \le j \le q} \left\{ \sum_{i < p \le j} W_p + \max\{0, \min\{l_j - B_j, L - P_j\}\} \right\}, \tag{48}$$

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 prevent any requests from exceeding the maximum user ride time.

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 node. Similarly, it makes sense to add depot 0 as first node 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 detecting some infeasibilities. This can 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 [10] two requests $(i, n + i)$ and $(j, n + j)$ cannot be served by the same vehicle if all possible paths serving the two requests turn out to be infeasible. According to the precedence constraints the following paths have to be considered:

$$
\begin{aligned}
&(i, j, i + n, j + n),\\
&(i, j, j + n, i + n),\\
&(i, i + n, j, j + n),\\
&(j, i, i + n, j + n),\\
&(j, i, j + n, i + n),\\
&(j, j + n, i, i + n).
\end{aligned}
$$

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 per se infeasible requests. Thus, it is sufficient to check if at least one of the following options is available:

$$
\begin{array}{lclcl}
(i, j) & \wedge & (j, i + n) & \wedge & (i + n, j + n),\\
(i, j) & \wedge & (j + n, i + n), & & \\
(i + n, j), & & & & \\
(j, i) & \wedge & (i + n, j + n), & & \\
(j, i) & \wedge & (i, j + n) & \wedge & (j + n, i + n),\\
(j + n, i). & & & &
\end{array}
$$

If none of them is possible, these two requests cannot be served by the same vehicle. As a consequence, this allows the removal of all arcs between the nodes associated with the pick-up and drop-off locations of requests i and j.

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:

$$y_i^k + y_j^k \leq 1 \qquad \forall k \in K, \forall (i, j) \in C. \tag{49}$$

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 [10] 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.

4.2. Subproblem Relaxations

In this section we describe the used subproblem relaxations which are incorporated into the master problem in terms of valid inequalities. The purpose of these constraints is to add subproblem knowledge to the master problem to avoid poor assignments in earlier iterations where only few Benders cuts are present.

4.2.1. Capacities

We consider pairs of requests that are guaranteed to be together on the vehicle if served within the same tour. Based on these "overlapping requests" we construct a conflict graph to derive clique inequalities.

Definition 6. *Request* $(i, n+i)$ overlaps *with request* $(j, n+j)$ *for* $i, j \in P$ *if there exists a feasible path serving the two requests but paths* $(i, i+n, j, j+n)$ *and* $(j, j+n, i, i+n)$ *are both infeasible.*

Informally this means that two requests overlap if they can be served by a single vehicle but not in strict succession.

We then define graph $G^{\mathrm{C}} = (V^{\mathrm{C}}, E^{\mathrm{C}})$ with $V^{\mathrm{C}} = P$ and $E^{\mathrm{C}} = \{\{i, j\} \mid \text{request } (i, i + j) \text{ overlaps with request } (j, n + j), i \in P, j \in P\}$. In this graph we identify all maximal cliques. This can be done by the Bron-Kerbosch algorithm, see [7]. 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 number 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_C^k be the maximum number of requests in C that fit into vehicle $k \in K$. Then, we can add the following inequalities to the master problem:

$$\sum_{i \in C} y_i^k \leq k_C^k \qquad \forall C \in \mathcal{C} : k_C^k < |C|. \tag{50}$$

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. Since the number of these cuts is usually not that large, 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.

4.2.2. Computing a Lower Bound on the Tour Duration

We compute for each node $i \in N$ the minimal time required to reach the next node, 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:

$$t_R^{\min} = t_0^{\min} + \sum_{i\in I} (t_i^{\min} + d_i + t_{n+i}^{\min} + d_{n+i}). \tag{51}$$

This relaxation gives us a (frequently 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:

$$t_0^{\min} + \sum_{i\in P} y_i^k (t_i^{\min} + d_i + t_{n+i}^{\min} + d_{n+i}) \leq T^k \qquad \forall k \in K. \tag{52}$$

This bound can be improved in certain cases. If $t_i^{\min}$ and $t_{n+i}^{\min}$ refer to the same target node v' (i.e., $t_i^{\min} = t_{iv'}$ 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 $t^{\min}$ values accordingly. If neither i nor $(n+i)$ has an outgoing arc to a node 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.

4.3. 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 12 to 15 for the moment). Remember that our decomposition approach uses only 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 13) to derive feasible solutions from intermediate infeasible master assignments, possibly even closing the optimality

```
j ← 0;                                                   // Iteration counter
repeat
    j ← j + 1;
    feasible ← true;
    solve master problem;
    foreach k ∈ K do
        if sub(k, I_j^k) is infeasible then
            add Benders cuts to the master problem;
            feasible ← false;
        end
    end
    if feasible = false then
        repair();            // construct feasible solution heuristically
        // check whether optimality gap could be closed
        if obj(master problem) = obj(repair) then  feasible ← true ;
    end
until feasible = true ∨ time limit reached;
// if feasible=true then optimal solution found
// else potentially suboptimal, repaired solution
```

Algorithm 4.1: Logic-based Benders Algorithm

gap allowing premature termination. Details on the used repair heuristic will be given in Section 4.3.2.

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

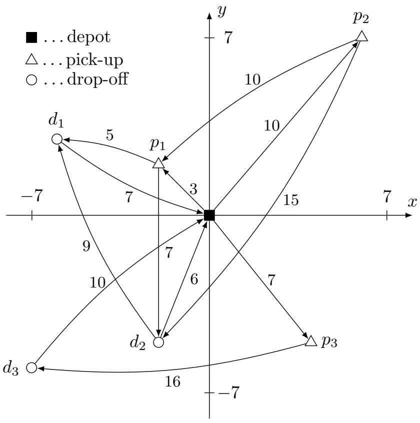
4.3.1. Benders Cuts

In our experiments in Section 5 we will consider four strategies for constructing Benders infeasibility cuts: variant *simple* uses the unmodified master assignment, *aIIS* uses all IISs that can be obtained from the initial assignment, *mIIS* uses all IISs of minimal cardinality, and *gIIS* uses heuristically computed IISs. The IISs for variants aIIS and

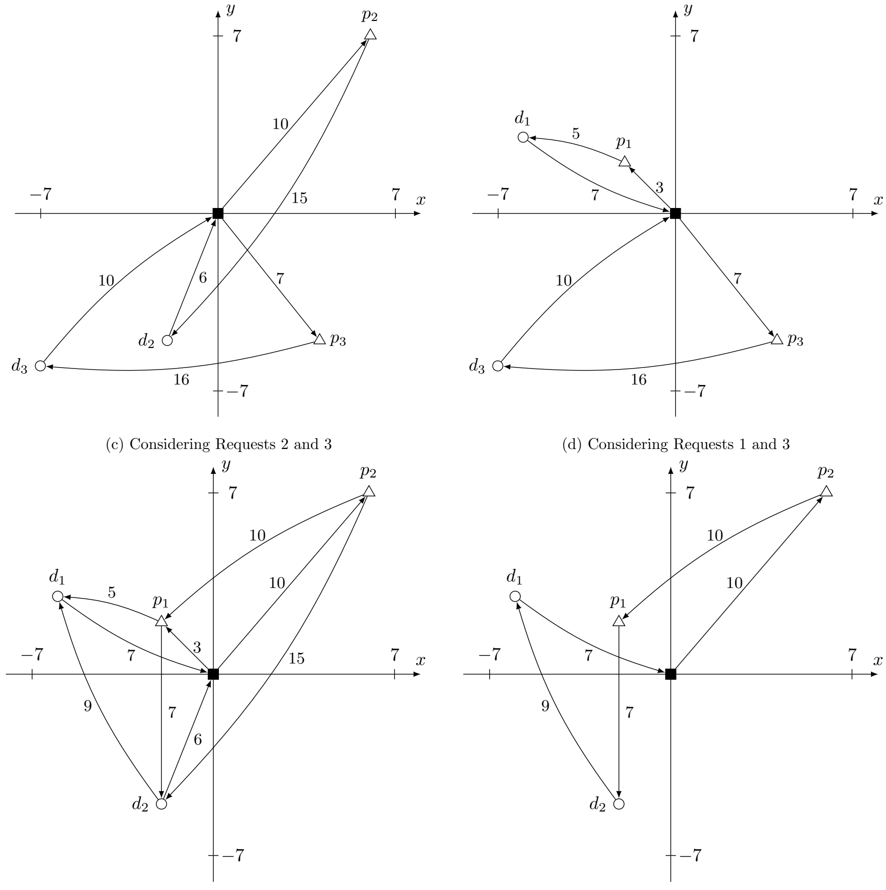
	Coordinates		Time Windows			
Node	x	y	e	l	q	d
depot	0	0	0	60	0	0
p_1	-2	2	17	30	1	3
d_1	-6	3	40	55	-1	3
p_2	6	7	12	25	1	3
d_2	-2	-5	35	50	-1	3
p_3	4	-5	32	41	1	3
d_3	-7	-6	45	50	-1	3

$|K| = 1$, $T^1 = 60$, $Q^1 = 3$, $L = 20$

(a) Input Data



(b) Preprocessed Graph



(c) Considering Requests 2 and 3

(d) Considering Requests 1 and 3

(e) Considering Requests 1 and 2

(f) Optimal Solution

Figure 1: A simple Benders iteration without valid 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 unique optimal solution.

mIIS are computed using bottom-up enumeration by extending an initially empty set with the assigned requests until it becomes infeasible (including appropriate pruning for the minimum cardinality variant). Variant gIIS applies Algorithm 3.1 once in ascending and once in descending natural order of the request indices to obtain two IISs. If both turn out to be equivalent, the second set is discarded. As there is no connection between the order of request indices and their properties, this means that there is no strategic decision involved.

4.3.2. Repair Heuristic

Similarly as done in [8] we use a repair heuristic to construct feasible solutions based on infeasible assignments obtained from the Benders master problem. To this end we consider the input sets I^k, $k \in K$, with some of them possibly being 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. This step simplifies if the related subproblem turned out to be feasible because we can directly add all requests in this case. 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 profitable due to freed-up requests.

4.3.3. 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 by Berbeglia et al. [5]. 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 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.

4.4. Heuristic Boosting

Empirical tests have shown that the master problem of the LBBD approach frequently finds good or even optimal solutions fast. Afterwards, a significant amount of time is typically spent to close the relative gap between lower (LB) and upper (UB) bound, i.e.,

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

$F \leftarrow P \setminus \bigcup_{k \in K} I^k$; `// set of unassigned requests`
foreach $k \in K$ **do**
    if $sub(k, I^k)$ *is feasible* **then**
        $\tilde{I}^k \leftarrow I^k$;
        $I^k \leftarrow \emptyset$;
    else
        $\tilde{I}^k \leftarrow \emptyset$;
        foreach $i \in I^k$ **do**
            if $sub(k, \tilde{I}^k \cup \{i\})$ *is feasible* **then**
                $I^k \leftarrow I^k \setminus \{i\}$;
                $\tilde{I}^k \leftarrow \tilde{I}^k \cup \{i\}$;
            end
        end
    end
    foreach $i \in F$ **do**
        if $sub(k, \tilde{I}^k \cup \{i\})$ *is feasible* **then**
            $F \leftarrow F \setminus \{i\}$;
            $\tilde{I}^k \leftarrow \tilde{I}^k \cup \{i\}$;
        end
    end
    $F \leftarrow F \cup I^k$; `// unused requests might be assigned to the other vehicles`
    $I^k \leftarrow \tilde{I}^k$;
end

Algorithm 4.2: Repair Heuristic

the optimality gap (UB − LB)/LB. However, closing the gap might not be required to obtain an intermediate solution yielding high quality Benders cuts. Note the similarity to BaC which also derives Benders cuts from potentially suboptimal solutions encountered during the B&C search. The following sections describe our approaches exploiting this observation.

4.4.1. 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, i.e., using a threshold of zero, until no additional Benders cuts can be identified. Thus, we still obtain an optimal solution but might save time that is “wasted” on closing the optimality gap.

The difficulty is to choose a suitable threshold for premature termination, especially in earlier iterations. Using a large threshold has higher potential for speedup but can also lead to significantly worse intermediate solutions. Correspondingly, we might obtain weaker Benders cuts, implying a larger number of master iterations.

To overcome the limitations of using a single threshold, we consider a more sophisticated adaptive approach based on a decreasing sequence of thresholds. Initially we start with the largest threshold and then switch to the subsequent smaller one every time no further cuts can be identified. Apart from this iterative variant we consider an up-and-down approach that allows the gap threshold to adjust in both directions. As before, we switch to the next smaller threshold whenever no further cuts can be identified. In addition, we now also switch back to the previous larger threshold if cuts could be added, for details see Algorithm 4.3. To preserve optimality the smallest threshold needs to be zero (or a sufficiently small numerical constant). For the used threshold values see Section 5.2.2. Preliminary experiments have shown that using a small sequence of specific values is superior to a more fine-grained approach, e.g., based on a geometric/arithmetic series.

Input: Decreasing sequence of gap thresholds $g = (g_m, g_{m-1}, \dots, g_1)$ with $g_i > g_{i-1}$ for $1 < i \leq m$

$i \leftarrow m$;
while *termination criteria not met* **do**
  solve Benders master problem until a relative optimality gap $\leq g_i$ is reached;
  derive and add Benders cuts;
  if *no Benders cuts found* $\wedge\, i > 1$ **then** $i \leftarrow i - 1$;
  else if *no Benders cuts found* $\wedge\, i = 1$ **then** terminate ; `// optimal`
  else if *Benders cuts found* $\wedge\, i < m$ **then** $i \leftarrow i + 1$;
end

Algorithm 4.3: Adaptive Gap Boosting (up-and-down)

A similar—yet different—approach is considered in Tran and Beck [43]. There the authors also terminate the master problem prematurely according to a threshold on the optimality gap. However, their motivation is to complete at least a single master iteration from which a heuristic solution is derived. In contrast, our approach is designed in an optimality preserving way while speeding up the overall computation.

Observe that the considered objective function is integral. Thus, it is also possible to specify a threshold for the absolute optimality gap (UB − LB) instead of the relative one. Preliminary experiments have shown that the behavior is roughly the same when choosing comparable thresholds.

4.4.2. Time Limit Boosting

Early termination based on the optimality gap helps to reduce time spent in the Benders master problem. However, the 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 each master problem instance. 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 a more flexible adaptive approach. In the beginning we use the smallest value of an increasing sequence of time limits and switch to the next larger one whenever no additional cuts can be found. Again, we consider a variant in which the time limit is adjusted in both directions. Optimality is preserved by using the total remaining time as final value of the sequence. For the used time limits see Section 5.2.2.

4.4.3. Solving the Subproblems Heuristically

We further tried to improve the solving of the subproblems by first using heuristics as done in [36]. 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 the 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 valid route can be computed, we still have to check with an exact approach whether this result is correct. Preliminary tests have shown that the employed heuristic required such exact reevaluations quite frequently, outweighing the provided speedup from the positive cases.

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

The most commonly used benchmark instances for the classical DARP are those by Cordeau and Laporte [12] and Cordeau [10]. The first set is mainly interesting for testing with heuristics due to the large number of requests considered. Therefore, we decided to use the latter. The mentioned work considers instances of up to 48 requests and 4 vehicles. In Ropke et al. [39] this set got extended with instances of up to 96 requests and 8 vehicles. The properties of this instance set are shown in Table 1.

Berbeglia et al. [5] consider variants of these instances with modified maximum user ride times of $L = 30$ and $L = 22$, respectively, and a variant with 75% of the originally available vehicles. We consider the original instances and two of the modified variants excluding the one with $L = 22$ because it turned out that this modification makes some requests infeasible, i.e., not even a tour containing no other requests is feasible[1]. The unmodified instances are guaranteed to be feasible, i.e., it is known in advance that all requests can be served. Therefore, they are only of minor relevance for testing our algorithms. As shown in Berbeglia et al. [5] and Häme and Hakula [24] also most of the modified instances are feasible.

Under these premises we decided to generate further, for our scenario more relevant, instances. We aim at two aspects: First, we require instances that are more challenging from the "packing" perspective. This means that it is not guaranteed that all requests can be served. We accomplish this by choosing the number of requests large in relation to the length of the time horizon. Second, the existing instance set is too diverse to precisely measure the impact of certain instance properties. In particular, we are interested in instances with different degrees of utilization, i.e., the number of requests compared to the number of available vehicles. To this end, we consider scenarios with four and five vehicles and a (small) fixed time horizon while only varying the number of available requests.

The new instance are generated according to the procedure mentioned in [10]. We first place nodes randomly on a 20×20 grid; the depot is located in the center of this grid at coordinates $(0, 0)$. Travel times between the nodes are set to the Euclidean distances between the corresponding points. For each instance with n requests the first $n/2$ requests are considered to be outbound requests and the remaining ones 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 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.

We consider a time horizon of $T = 240$ which corresponds to a half working day (assuming minutes as unit of time). For outbound requests we set the time window at the drop-off location $(n + i)$ by first choosing e_{n+i} uniformly at random from the interval $[t_{0i} + d_i + t_{i,n+i}, T - t_{n+i,2n+1} - d_{n+i} - 15]$ 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 can always return feasibly to the depot. Similarly, we choose for inbound requests e_i of pick-up node i from the interval $[t_{0i}, T - t_{n+i,2n+1} - d_{n+i} - t_{i,n+i} - d_i - 15]$ and set $l_i = e_i + 15$. The remaining time windows are then tightened as described

[1]For an example see request 21 in instance a6-60.

Instance	n	$\lvert K\rvert$	L	Q	T	Instance	n	$\lvert K\rvert$	L	Q	T
a2-16	16	2	30	3	480	b2-16	16	2	45	6	480
a2-20	20	2	30	3	600	b2-20	20	2	45	6	600
a2-24	24	2	30	3	720	b2-24	24	2	45	6	720
a3-18	18	3	30	3	360	b3-18	18	3	45	6	360
a3-24	24	3	30	3	480	b3-24	24	3	45	6	480
a3-36	36	3	30	3	720	b3-36	36	3	45	6	720
a4-16	16	4	30	3	240	b4-16	16	4	45	6	240
a4-24	24	4	30	3	360	b4-24	24	4	45	6	360
a4-32	32	4	30	3	480	b4-32	32	4	45	6	480
a4-40	40	4	30	3	600	b4-40	40	4	45	6	600
a4-48	48	4	30	3	720	b4-48	48	4	45	6	720
a5-40	40	5	30	3	480	b5-40	40	5	45	6	480
a5-50	50	5	30	3	600	b5-50	50	5	45	6	600
a5-60	60	5	30	3	720	b5-60	60	5	45	6	720
a6-48	48	6	30	3	480	b6-48	48	6	45	6	480
a6-60	60	6	30	3	600	b6-60	60	6	45	6	600
a6-72	72	6	30	3	720	b6-72	72	6	45	6	720
a7-56	56	7	30	3	480	b7-56	56	7	45	6	480
a7-70	70	7	30	3	600	b7-70	70	7	45	6	600
a7-84	84	7	30	3	720	b7-84	84	7	45	6	720
a8-64	64	8	30	3	480	b8-64	64	8	45	6	480
a8-80	80	8	30	3	600	b8-80	80	8	45	6	600
a8-96	96	8	30	3	720	b8-96	96	8	45	6	720

Table 1: Properties of the instances by Ropke et al. [39]. Per instance vehicle capacities as well as the maximum route durations are the same for all vehicles. The maximum route durations are the same as the time horizon. In group “a” all requests have a load of $q_i = 1$ and a service time of $d_i = 3$. For group “b” the loads are chosen uniformly at random from $\{1, \dots, 6\}$ with proportional service times $d_i = q_i$.

Instance	n	$\lvert K\rvert$	L	Q	T	Instance	n	$\lvert K\rvert$	L	Q	T
30N_4K_A	30	4	30	3	240	44N_5K_A	44	5	30	3	239
30N_4K_B	30	4	30	3	240	44N_5K_B	44	5	30	3	240
30N_4K_C	30	4	30	3	240	44N_5K_C	44	5	30	3	240
30N_5K_A	30	5	30	3	240	50N_4K_A	50	4	30	3	240
30N_5K_B	30	5	30	3	240	50N_4K_B	50	4	30	3	240
30N_5K_C	30	5	30	3	240	50N_4K_C	50	4	30	3	240
40N_4K_A	40	4	30	3	240	50N_5K_A	50	5	30	3	240
40N_4K_B	40	4	30	3	240	50N_5K_B	50	5	30	3	240
40N_4K_C	40	4	30	3	232	50N_5K_C	50	5	30	3	240
40N_5K_A	40	5	30	3	240	60N_4K_A	60	4	30	3	240
40N_5K_B	40	5	30	3	240	60N_4K_B	60	4	30	3	240
40N_5K_C	40	5	30	3	240	60N_4K_C	60	4	30	3	240
44N_4K_A	44	4	30	3	240	60N_5K_A	60	5	30	3	240
44N_4K_B	44	4	30	3	240	60N_5K_B	60	5	30	3	240
44N_4K_C	44	4	30	3	240	60N_5K_C	60	5	30	3	240

Table 2: Properties of the newly generated *SDARP* instance set. Per instance vehicle capacities as well as the maximum route durations are the same for all vehicles. The maximum route durations are the same as the time horizon. All requests have a load of $q_i = 1$ and a service time of $d_i = 3$.

in Section 4.1.1. For each request we assume a 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 numbers of homogeneous vehicles with capacity $Q^k = 3$ and maximum route duration $T^k = T$. Table 2 provides an overview of the properties of the generated test instances. In the following we are going to refer to this instance set as “SDARP” instances. The SDARP instances are available at `https://www.ac.tuwien.ac.at/research/problem-instances/#Dial-a-Ride_Problem`.

To deal consistently with the Euclidean distances in the MILP and CP algorithms we restrict the precision to two fractional digits for both instance sets.

5.2. Computational Experiments

In this section we are going to present the computational results of our algorithms obtained on the introduced benchmark 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 up to 8GB. Test runs have been executed using CPLEX 12.7.1 with a single thread using dual simplex and traditional B&C. The CP part has been implemented using Gecode 5.1.0 [21], also utilizing only a single thread for each test run. For the Bron-Kerbosch algorithm we used the implementation from

BaC	branch-and-check
LBBD	logic-based Benders decomposition
simple	unrefined Benders cuts
aIIS	Benders cut per irreducible infeasible set (IIS)
mIIS	Benders cut per minimum cardinality IIS
gIIS	Benders cuts for up to two greedily computed IISs
MIP	MILP subproblems
CPMIP	combined CP-MILP subproblems
Hgaprel_it	heuristic boosting with thresholds for the relative optimality gap (iterative)
Hgaprel_ud	heuristic boosting with thresholds for the relative optimality gap (up-and-down)
Htime_it	heuristic boosting with time limits (iterative)
Htime_ud	heuristic boosting with time limits (up-and-down)
Compact	compact MILP model

Table 3: Summary of the abbreviations used to identify the tested algorithms.

Boost 1.63.0. Since objective values are known to be integral, runs terminate once the absolute optimality gap falls below a threshold of $1 - n \cdot \varepsilon$, where ε is the reduced-cost optimality tolerance of the MILP solver.

We start by investigating the different approaches for solving the subproblems and generating Benders cuts. Then, we evaluate the heuristic boosting techniques before providing further insights regarding specific properties of our algorithms. For these parts we rely on our newly generated SDARP instances. Afterwards, we test our algorithms on the instances used in the previous literature and provide a comparison to the compact MILP reference model.

We try to condense the presented results as much as possible to highlight the core results. For more details we provide additional tables in the appendix. Table 3 summarizes the abbreviations used to identify the algorithm variants.

5.2.1. Evaluation of Subproblem Algorithms and Cut Generation Strategies

In several of the upcoming figures we provide sums of the total number of served requests per algorithm. Due to the integrality of the objective function value, it is sometimes difficult to distinguish the marks when the time limit is reached. To resolve this issue we provide the respective numbers in Table 4. In addition, this table also provides results for algorithms that have been omitted from the figures to improve readability.

Figure 2 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.

A first observation is that the relative performance of the variants for computing Benders cuts is quite similar for both decomposition approaches. As expected, the naïve strategy performs quite poorly. It solves the fewest instances to optimality and also takes much longer to find results comparable to the other variants. Also, the strategy adding all IISs does not work well. Regarding the final number of served requests, it is

Algorithm	$\sum$ LB	#	Algorithm	$\sum$ LB	#
Compact	773	24	BD-gIIS-CP	1189	30
BaC-simple-CP	808	30	BD-mIIS-CP	1192	30
BD-aIIS-CP	938	27	BD-gIIS-MIP	1193	30
BD-aIIS-CPMIP-Htime_ud	947	26	BD-gIIS-CPMIP	1193	30
BD-aIIS-CPMIP-Htime_it	952	26	BD-mIIS-MIP	1200	30
BD-aIIS-CPMIP-Hgaprel_ud	953	26	BD-mIIS-CPMIP	1201	30
BD-aIIS-CPMIP-Hgaprel_it	954	26	BD-mIIS-CPMIP-Hgaprel_it	1209	30
BD-aIIS-CPMIP	972	27	BD-mIIS-CPMIP-Hgaprel_ud	1209	30
BaC-simple-CPMIP	978	30	BaC-gIIS-CP	1212	30
BaC-simple-MIP	983	30	BaC-mIIS-CP	1212	30
BD-aIIS-MIP	1011	29	BD-gIIS-CPMIP-Hgaprel_it	1213	30
BaC-aIIS-MIP	1092	28	BD-gIIS-CPMIP-Hgaprel_ud	1213	30
BaC-aIIS-CPMIP	1096	28	BaC-mIIS-MIP	1221	30
BaC-aIIS-CP	1097	28	BaC-gIIS-MIP	1223	30
BD-simple-CP	1150	30	BaC-gIIS-CPMIP	1224	30
BD-simple-CPMIP-Htime_it	1155	30	BD-gIIS-CPMIP-Htime_it	1224	30
BD-simple-CPMIP-Htime_ud	1155	30	BaC-mIIS-CPMIP	1225	30
BD-simple-MIP	1156	30	BD-gIIS-CPMIP-Htime_ud	1226	30
BD-simple-CPMIP	1157	30	BD-mIIS-CPMIP-Htime_it	1232	30
BD-simple-CPMIP-Hgaprel_it	1170	30	BD-mIIS-CPMIP-Htime_ud	**1233**	30
BD-simple-CPMIP-Hgaprel_ud	1170	30			

Table 4: Summary of the total number of served requests ($\sum$ LB) across all tested algorithms for the SDARP instances. Column # denotes the number of instances for which the respective algorithm computed a feasible solution. Algorithms that could not solve all 30 instances terminated prematurely due to the memory limit.

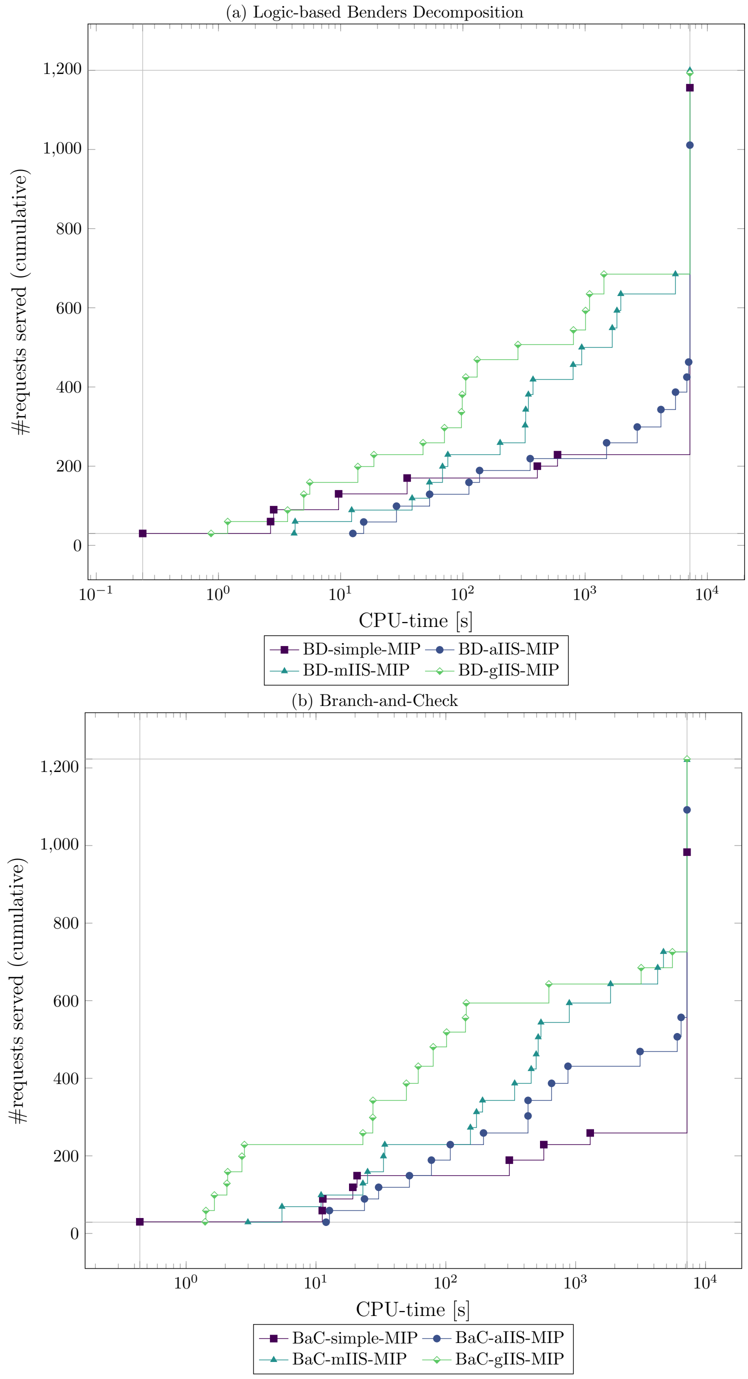


Figure 2: Comparison of different strategies for generating Benders cuts for LBBD and BaC with MILP subproblems in terms of served requests on the SDARP instances. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality or due the time limit of two hours. Both charts use a logarithmic x-axis.

even dominated by the simple approach for the LBBD. This is primarily due to several instances hitting the memory limit resulting in missing lower bounds (cf. Table 4). The bad performance of strategy aIIS compared to the more successful variants has two reasons: First, it takes a large amount of time to compute all the IISs. Second, the number of added cuts is typically rather large which increases the size of the master problem quite fast. Strategies gIIS and mIIS work much better in this respect. Of these two, the greedy variant exhibits slightly better results since the cuts it provides turned out to perform reasonably well but can be computed much faster than the minimum cardinality IISs. Nevertheless, our results show that the mIIS variant has large potential. Given the computational overhead for computing the minimum cardinality IISs, it is quite impressive that the approach is still competitive. This shows that the stronger cuts provide indeed a considerable improvement on the cuts obtained from the greedy approach. Having identified gIIS and mIIS as superior strategies for computing Benders cuts we focus on those two in the following.

Comparing the decomposition approaches we observe a slight advantage for BaC. It turned out that solutions are found faster by evaluating all integral solution candidates—instead of only optimal ones like LBBD does. For both algorithms finding good or even optimal solutions becomes harder the scarcer the available resources become compared to the demand. This relation can be expected as the number of combinatorial possibilities from which the algorithm needs to find an optimal one increases. In particular, the Benders algorithms are required to exclude a much larger number of infeasible assignments until only feasible options remain. Relaxations that bound the tour size (see Sections 4.2.1 and 4.2.2) help to reduce this effect.

In addition to solving the subproblems via an MILP solver we also investigated the CP approach from Section 3.3. In general, computation times are superior but for some instances severe outliers occurred, including situations in which single subproblems required more time than half an hour. Some of these difficult subproblems appear at the very beginning of the solution process due to unbalanced assignments. However, they also continue to occur later on for request subsets with cardinalities similar to those in optimal solutions. To still profit from the mostly faster CP variant we further investigated a combination working as follows. We start with the CP solver using a time limit of half a second. If no result is obtained, we apply the MILP approach 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 variable domains of the CP model. Thus, we can take advantage of the outcome of constraint propagation at the root node of CP search, which possibly yields a smaller model. The results for the combined CP-MILP subproblems are shown in Figure 3.

It can be seen that adding CP for solving the subproblems helps to find solutions faster than when using the pure MILP approach. In general the relation among the decomposition approaches and the cut generation techniques stays the same. However, the mIIS algorithms profit more since they spend more time on solving subproblems, see also Figure 4. As the addition of CP provides a clear improvement we selected the combined variant as subproblem algorithm for the remaining experiments.

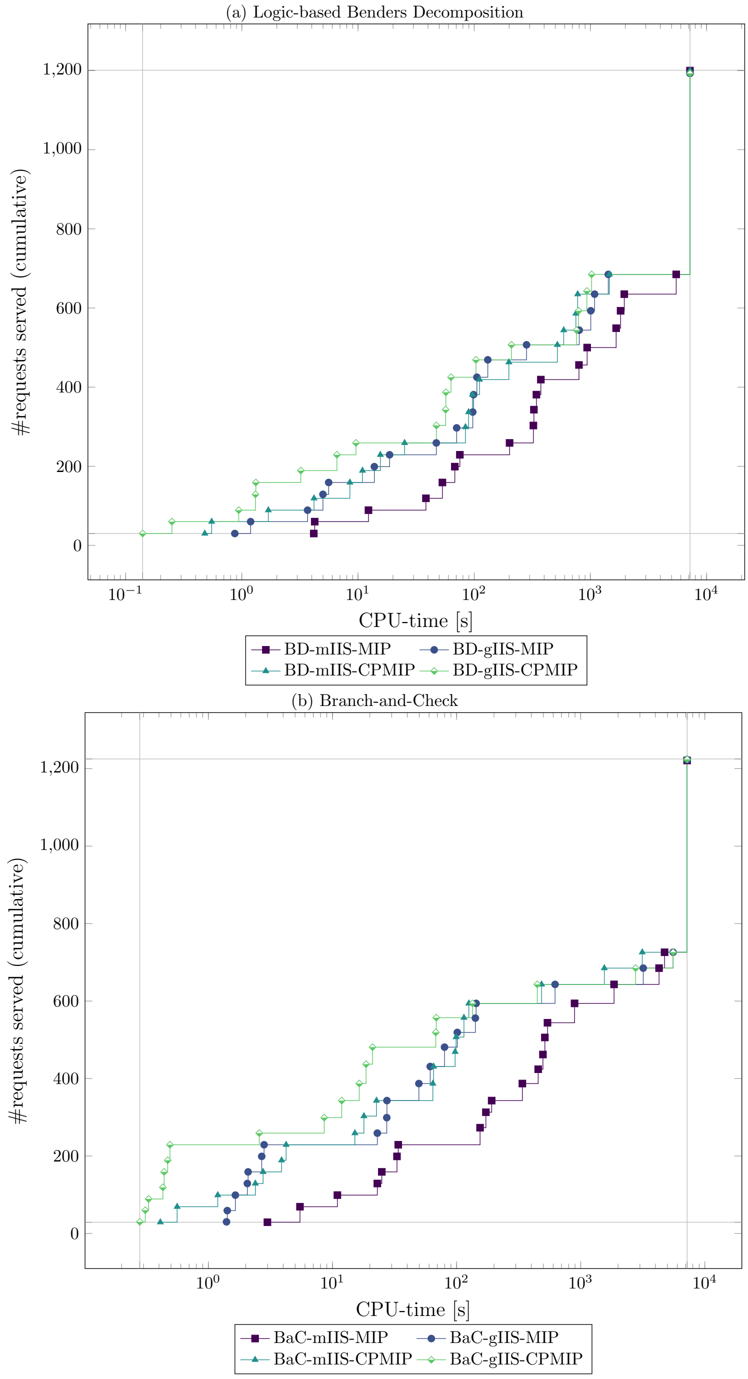


Figure 3: Comparison of MILP and combined CP-MILP subproblems for LBBD and BaC in terms of served requests on the SDARP instances. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality or due the time limit of two hours. Both charts use a logarithmic x-axis.

5.2.2. Evaluation of the Heuristic Boosting Techniques

In the following we compare the heuristically boosted LBBD algorithms to their basic counterparts. We consider the adaptive boosting in the pure iterative variant (it) and also the variant with adjustments in both directions (ud). As criteria for early termination we use a set of thresholds with respect to the relative optimality gap and a set of time limits, see Section 4.4. For the time limit variant we consider time limits of 5, 10, and 30 seconds in ascending order. As final value we use the total remaining time according to the overall time limit. The gap variant uses relative optimality gaps of 0.1, 0.05, 0.025, and 0 in descending order.

Table 5 shows the comparison for the heuristic boosting techniques. With both boosting techniques our algorithms could solve additional instances to optimality and also serve more requests in total. In general, we can observe that the time limit boosting technique works better than the gap boosting. It is mostly faster and also serves more requests overall. Compared to the un-boosted algorithms the time limit boosting is always at least as good in terms of the number of served requests. Except for one instance it provides improvements in all cases where the basic algorithm does not prove optimality. The highest improvements could be achieved by the up-and-down variants for which 33 (gIIS) and 32 (mIIS) additional requests could be served in total. In several cases also the computation times decreased, however, if already the basic variant works well, we sometimes observe a slowdown. To some extent this is related to the potentially worse cuts. The other reason are the required re-solves for proving optimality, which are not needed for the un-boosted algorithms. The gap boosting approaches feature a few outliers at which they serve fewer requests than the reference algorithm. Since the boosting is only a heuristic technique, such outliers are not unexpected. In certain cases it pays off to solve the master problem to optimality to obtain better cuts. However, considering the number of served requests in total, we still observe a reasonable improvement for the gap boosting technique.

In addition to the adaptive approaches presented above, we conducted preliminary tests using a single value as gap threshold or time limit. However, these variants turned out to be much less robust. For some instances they work exceptionally well but this is paid for exceedingly on the remaining ones.

5.2.3. Discussion of the Algorithm Properties

In Figure 4 we display the amount of time spent in the master problem, the subproblems, and the repair algorithm for the different decomposition approaches. The first thing to note is that the time spent in the subproblems decreases significantly when switching from the pure MILP algorithm to the combination with CP. This is most notable for the mIIS algorithms where the reduction is the largest due to the high number of solved subproblems.

The idea of the heuristic boosting techniques is to reduce the time spent in the master problem. Most of this time is really saved. However, part of it also shifts into the subproblems or the repair algorithm. In particular for the time limit boosting we observe

		ΔLB								Δcomputation time [s]							
		mIIS				gIIS				mIIS				gIIS			
		gap		time		gap		time		gap		time		gap		time	
Instance	LB*	it	ud	it	ud	it	ud	it	ud	it	ud	it	ud	it	ud	it	ud
30N_4K_A	**30**	-	-	-	-	-	-	-	-	**-3**	**-3**	-	-	-	-	-	-
30N_4K_B	**29**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
30N_4K_C	**30**	-	-	-	-	-	-	-	-	11	17	-	-	9	12	-	-
30N_5K_A	**30**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
30N_5K_B	**30**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
30N_5K_C	**30**	-	-	-	-	-	-	-	-	**-3**	3	-	-	3	1	-	-
40N_4K_A	**38**	-	-	-	-	-	-	-	-	34	122	**2**	**2**	18	44	6	**2**
40N_4K_B	**38**	-	-	-	-	-	-	-	-	18	18	1	2	**-139**	-131	-64	-75
40N_4K_C	**37**	-	-	-	-	-	-	-	-	-379	-341	-237	-237	**-587**	-495	-89	-301
40N_5K_A	**40**	-	-	-	-	-	-	-	-	17	17	-	-	-	-	-	-
40N_5K_B	**40**	-	-	-	-	-	-	-	-	28	228	4	3	26	242	**1**	2
40N_5K_C	**40**	-	-	-	-	-	-	-	-	**-1**	**-1**	-	-	**-1**	**-1**	-	-
44N_4K_A	40	1	1	1	1	**2**	**2**	**2**	**2**	-	-	-	-	-	-	-	-
44N_4K_B	**42**	-	-	-	-	-	-	-	-	923	2099	504	452	**-95**	1075	199	639
44N_4K_C	**41**	-	1	1	1	1	**2**	**2**	**2**	-	-2772	-2761	-666	-	-4261	**-4479**	-3640
44N_5K_A	**44**	-	-	-	-	-	-	-	-	169	897	**-2**	1	124	431	1	3
44N_5K_B	**44**	-	-	-	-	-	-1	-	-	**-151**	-59	-2	2	-2	208	1	2
44N_5K_C	**44**	-	-	-	-	-	-	-	-	190	1136	**-4**	6	121	1297	1	2
50N_4K_A	41	1	1	**3**	**3**	-	-	1	2	-	-	-	-	-	-	-	-
50N_4K_B	43	1	1	**4**	**4**	1	1	3	3	-	-	-	-	-	-	-	-
50N_4K_C	44	1	1	3	3	3	3	**4**	**4**	-	-	-	-	-	-	-	-
50N_5K_A	48	**3**	2	2	**3**	1	2	1	1	-	-	-	-	-	-	-	-
50N_5K_B	**49**	-	-	-	-	-	-	-	-	95	2007	302	104	**86**	1442	127	345
50N_5K_C	**50**	-	-	-	-	-	-1	-	-	212	135	**-492**	-471	-240	6267	517	-288
60N_4K_A	44	1	1	**5**	**5**	3	3	4	4	-	-	-	-	-	-	-	-
60N_4K_B	45	-2	-2	1	1	**3**	**3**	**3**	**3**	-	-	-	-	-	-	-	-
60N_4K_C	44	-	-	**3**	**3**	-1	-1	2	2	-	-	-	-	-	-	-	-
60N_5K_A	56	-1	-1	-	-	-	-	-	**1**	-	-	-	-	-	-	-	-
60N_5K_B	50	2	2	**4**	**4**	3	3	**4**	**4**	-	-	-	-	-	-	-	-
60N_5K_C	53	1	1	4	4	4	4	**5**	**5**	-	-	-	-	-	-	-	-
Total		8	8	31	32	20	20	31	**33**	1161	3504	-2685	-799	-677	6132	**-3779**	-3310

Table 5: Results of the heuristic boosting techniques. Column LB* denotes the best lower bound, provably optimal values marked bold. Columns ΔLB and "Δcolumn computation time" report the difference of the lower bounds and computation times, respectively, to the un-boosted algorithm variants. For the lower bound, positive values indicate that additional requests could be served and negative values indicate the contrary. Negative values for the computation times indicate a speedup and positive values a slowdown. Cells that contain "-" indicate that the respective value did not change. The largest improvements per column and instance are marked bold.

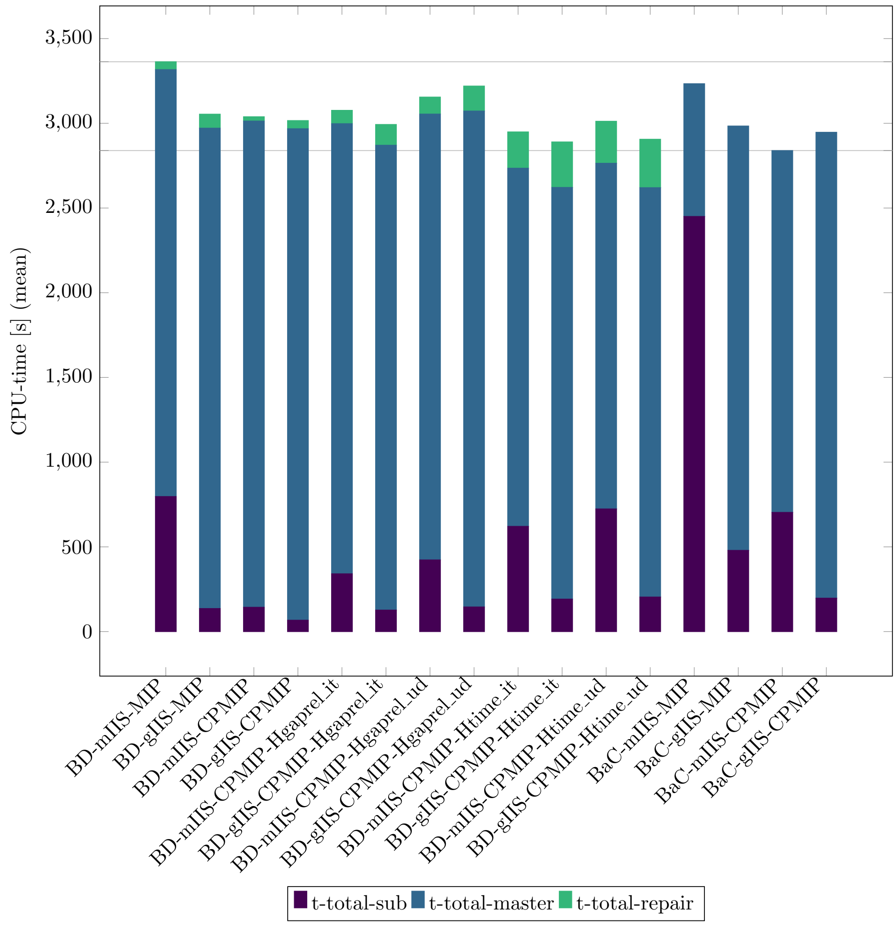


Figure 4: Average computation time spent in the master problem, the subproblem, and the repair algorithm for different LBBD and BaC approaches on the SDARP instances.

a significant increase regarding the time spent for repairing solutions. The advantage of the time limit boosting is that it consistently reduces the time spent per master iteration. However, for more challenging master iterations this might lead to worse solutions that leave more work for the repair algorithm.

Figure 5 provides details on the gaps with respect to the best known bounds. We compute lower bound gaps by $100 \cdot (\mathrm{LB}^* - \mathrm{LB})/\mathrm{LB}^*$ and upper bound gaps by $100 \cdot (\mathrm{UB} - \mathrm{UB}^*)/\mathrm{UB}^*$ 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.

Observe that the heuristically boosted LBBD as well as the BaC algorithms perform particularly well with respect to the lower bound gaps. However, they mostly do not perform so well when it comes to finding good upper bounds. In general, the heuristically boosted LBBD provides the better balance, featuring an acceptable performance for both parts. The un-boosted LBBD works not so well for the lower bounds but in general provides the best upper bounds. According to the design of the algorithms this is exactly what one would expect. BaC as well as the heuristically boosted LBBD both derive cuts from potentially suboptimal master solutions. On the one hand, this helps to reduce the time spent for solving the master problem and to derive feasible solutions earlier. On the other hand, this typically slows down progress with respect to the upper bound. In contrast, the un-boosted LBBD solves the master problem always to optimality which helps to find tight upper bounds and strong Benders cuts while taking longer to find good feasible solutions—even with the repair heuristic.

5.2.4. Comparison to the Literature

In the following we test our algorithms on the instances by Ropke et al. [39] including the modified variants by Berbeglia et al. [5] to establish a connection to the existing literature. Different from the SDARP instances, the instances by Ropke et al. [39] feature a significantly larger time horizon relative to the number of available requests. The unmodified instances are guaranteed to be feasible, i.e., all requests can be served. For most of the modified instances it is also possible to serve all requests. In case not all requests can be served, only few have to be rejected. This means that the master problem is much easier to solve than for the SDARP instances. We illustrate this behavior in Figure 6. To improve readability we omitted the mIIS variants of BaC with quite excessive computation times of 218 (CP-MILP subproblems) and 793 (MILP subproblems) seconds on average since their relative time distribution is similar to the BaC algorithms included in the figure.

Observe that the Benders algorithms spend almost no time on solving the master problem, little time is spent in the repair routine, and most of the time is spent in the subproblems. The behavior on the modified instances is quite similar, except that the overall computation times increase and that a little more time is spent in the repair routine. Due to this distribution the heuristic boosting techniques do not have a noticeable effect. Moreover, solving the subproblems is more challenging since they typically involve a higher number of requests. This leads to advantages for the gIIS approach that still provides a reasonable guidance for the master problem but solves considerably fewer subproblems than the mIIS variant. Therefore, we focus on the gIIS algorithms without heuristic boosting for the upcoming comparison.

Figures 7 and 8 provide an overview regarding the results of the obtained lower bounds and computation times. The investigated algorithms solve all instances to optimality. The performance with respect to the original instances and those with restricted ride time is quite similar, which is not unexpected because half of the instances (group “a”) is not affected by the modification. In contrast, the instances with a reduced number of vehicles are significantly more challenging. This is due to the higher number of requests that have to be rejected. The success of LBBD on the unmodified and the $L = 30$ instances has two reasons. First, the master problem is much easier to solve than for the SDARP instances. Therefore, it is more affordable to always solve it to optimality allowing the LBBD algorithm to converge faster. Second, the repair heuristic turned out to be particularly successful: For the unmodified instances it is often possible to prove

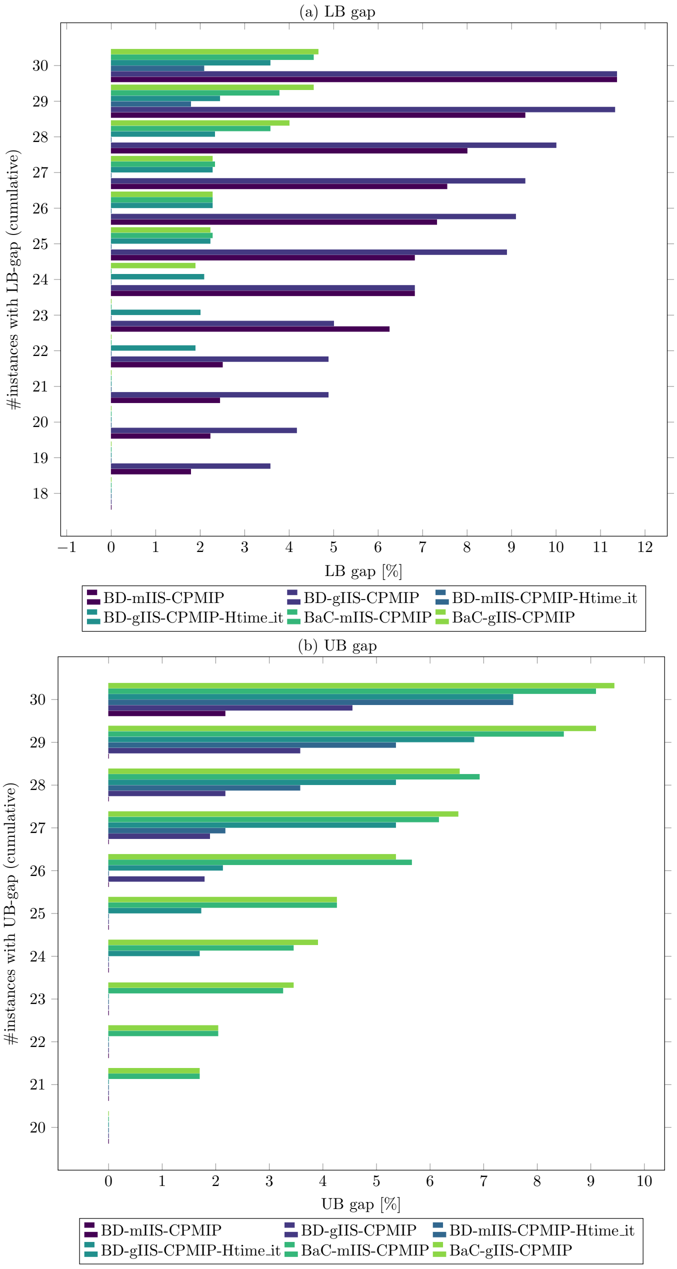


Figure 5: Lower and upper bound gaps for different LBBD and BaC algorithms on the SDARP instances. For each algorithm the length of the bar at coordinate y_i corresponds to the largest gap among the y_i instances with the smallest gaps.

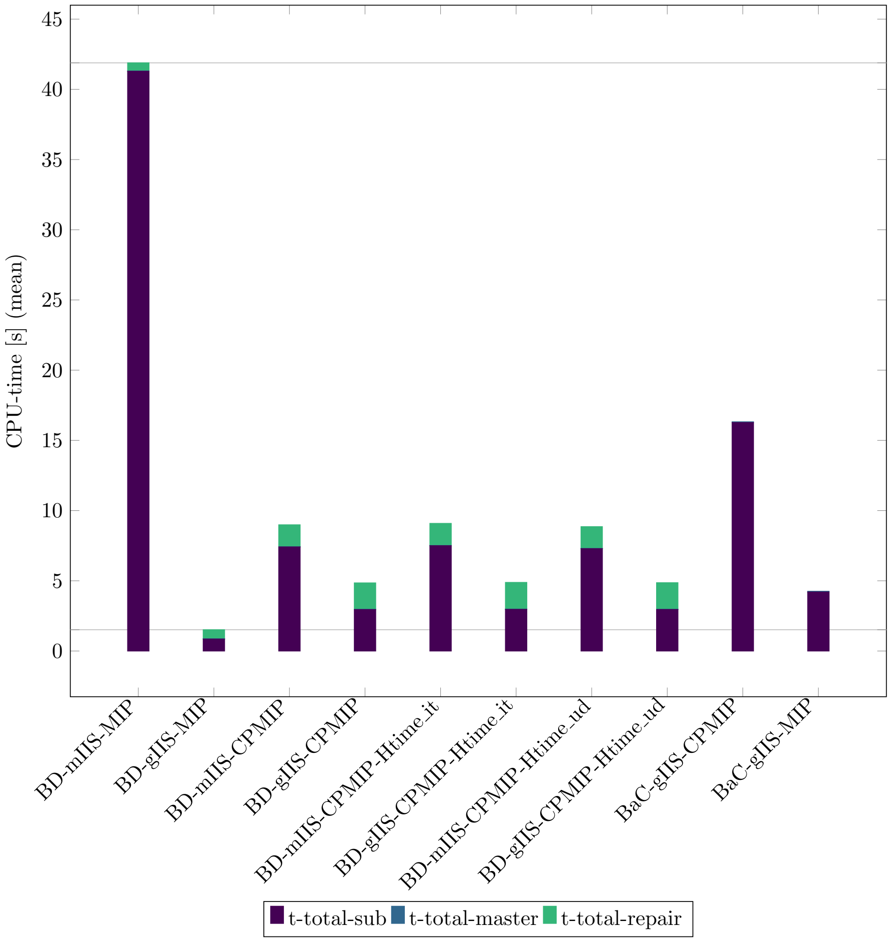


Figure 6: Average computation time spent in the master problem, the subproblem, and the repair algorithm for different LBBD and BaC algorithms on the instances by Ropke et al. [39].

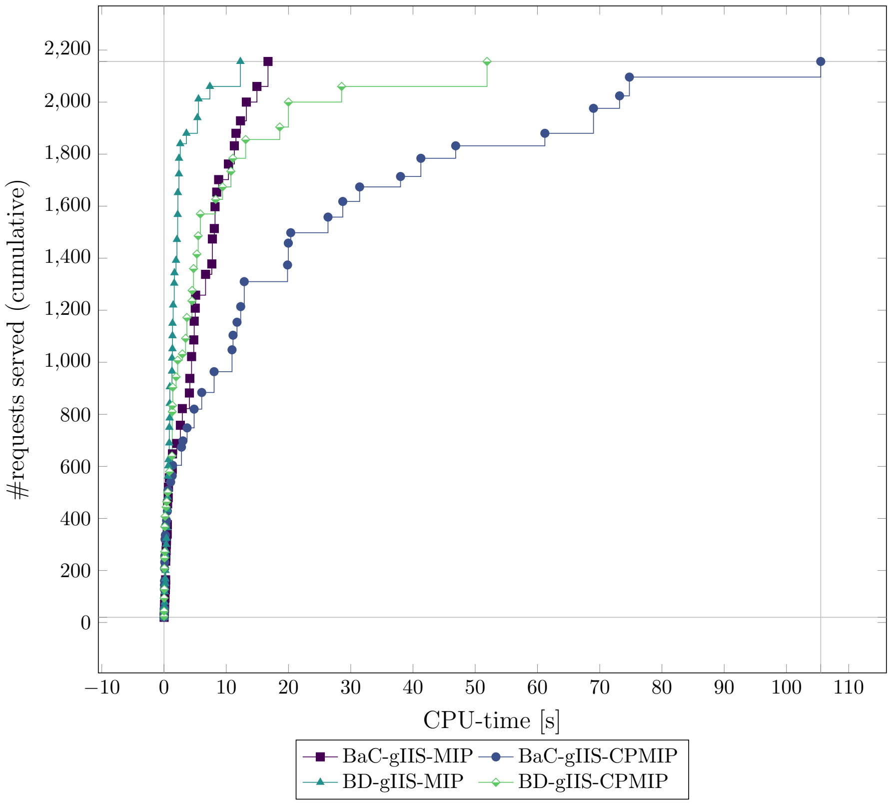
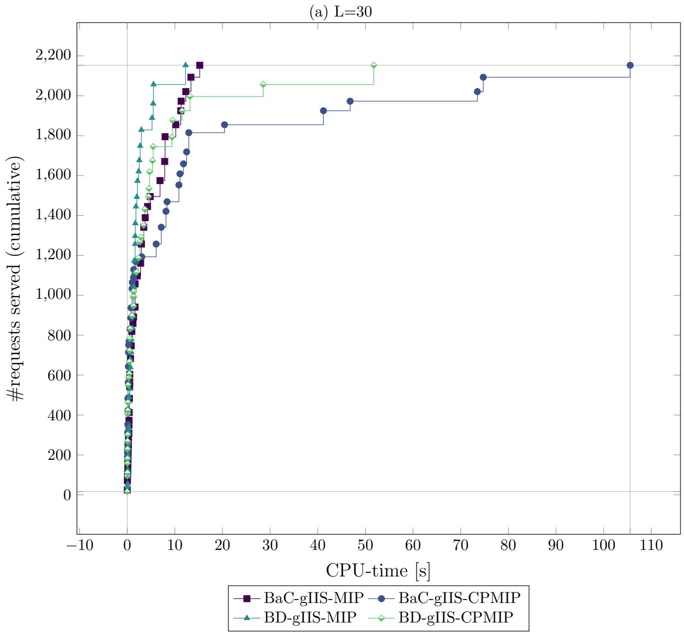


Figure 7: Comparison of LBBD and BaC with different types of MILP and combined CP-MILP subproblems in terms of served requests on the instances by Ropke et al. [39]. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality.

optimality after the repair operation in the very first iteration, i.e., without adding any Benders cuts and resolving the master problem. For the instances with only 75% of the original vehicles this is usually not possible which makes BaC the superior algorithm here.

Another interesting observation is that using pure MILP subproblems turned out to work better here when using the greedy approach for computing Benders cuts. The reason is that the subproblems are typically less constrained than for the SDARP instances. Moreover, the initial subproblems tend to be larger due to the higher number of requests. However, the mIIS algorithm in general works better with the combined CP-MILP approach due to the bottom-up construction of the IISs, which involves solving many small subproblems.

In Tables 6 and 7 we provide the results for the modified instances by Berbeglia et al. [5], restricted to those cases in which not all requests can be served. The feasibility checking algorithm by Häme and Hakula [24] also provides a partial solution in case not all requests can be served. We compare our results to theirs showing that further



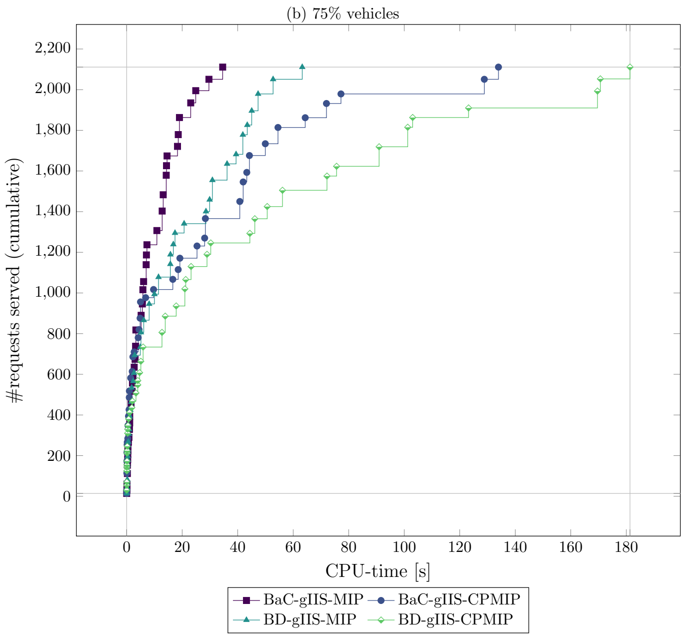


Figure 8: Comparison of LBBD and BaC with different types of MILP and combined CP-MILP subproblems in terms of served requests on the instances by Berbeglia et al. [5]. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality.

Instance	Opt.
b2-16_L30	15
b2-20_L30	19
b3-24_L30	23

Table 6: Summary of the number of served requests for the instances by Ropke et al. [39] with $L = 30$. We only list the instances for which not all requests can be served. Column "Opt." provides the maximal number of requests that can be served.

Instance	Opt.	Prev. Best	Instance	Opt.	Prev. Best
a2-16_K75	**12**	-	b4-24_K75	**22**	-
a2-20_K75	**18**	-	b4-40_K75	**39**	38
a2-24_K75	**21**	-	b4-48_K75	**47**	**47**
a3-36_K75	**32**	-	b5-40_K75	**39**	**39**
b2-16_K75	**12**	-	b5-50_K75	**47**	44
b2-20_K75	**14**	-	b5-60_K75	**58**	56
b2-24_K75	**20**	-	b6-48_K75	**46**	**46**
b3-24_K75	**23**	-	b7-84_K75	**83**	82
b3-36_K75	**32**	-			

Table 7: Summary of the number of served requests for the instances by Ropke et al. [39] with only 75% of the original vehicles. We only list the instances for which not all requests can be served. Column "Opt." provides the maximal number of requests that can be served according to our experiments. Column "Prev. Best" reports the results by Häme and Hakula [24]; instances not considered by them are marked with "-". Provably optimal solution values are marked bold.

requests can be served in an optimal solution for some instances. Considering the pure feasibility checking task, our algorithms are not as fast in terms of computation times. The maximum cluster algorithm by Häme and Hakula [24] computes the feasibility status in less than a second for most instances, except for a few outliers for the modification with the reduced fleet size taking up to 47 seconds. Our algorithms are able to solve all instances in up to 35 seconds (BaC-gIIS-MIP). Given that our main goal is to determine the maximum number of requests that can be served—the feasibility status is only obtained as a side result—the computational performance appears to be reasonable.

5.2.5. Comparison to the Compact MILP Model

In the following we compare our decomposition approaches to the compact MILP model provided in Section 3.2. Since ease of implementation is often a major concern, we selected algorithm variants as competitors that are most comparable in this respect.

Therefore, we choose BaC with unrefined Benders cuts and Benders cuts computed by the greedy approach. Note that, using, e.g., CPLEX, BaC can be implemented in terms of a model with a lazy constraint-callback that solves a compact MILP model as subproblem. The subproblem is solved exactly once for the naïve approach and multiple times according to Algorithm 3.1 when using the greedy strategy.

Figures 9 and 10 compare the BaC algorithms and the compact MILP model. Already the naïve strategy for generating Benders cuts turns out to perform better in terms of optimally solved instances. However, considering the finally obtained lower bounds it is only superior on the SDARP instances. The greedy Benders cut generation approach, on the other hand, dominates the compact model in all aspects. It is not only much faster but its final solutions in general also serve significantly more requests. Across all instances BaC-gIIS-MIP solves 45 more instances to optimality than the compact model: 14 of the SDARP instances, 11 of the original Cordeau instances, 7 of the Cordeau instances with reduced user ride time, and 13 of the Cordeau instances with reduced fleet size.

6. Conclusion

In this work we considered a variant of the dial-a-ride problem (DARP) that aims at serving a maximal number of requests rather than minimizing routing costs. We proposed a simple compact reference model and a decomposition approach. The master problem was formulated as mixed integer linear programming (MILP) model and the subproblems were stated as MILP model and also as constraint programming (CP) model. We reviewed preprocessing techniques from the literature and suggested improvements. The master problem of the decomposition approach is supplemented by inequalities representing subproblem relaxations.

In the computational study we solved the decomposition model using logic-based Benders decomposition (LBBD) and branch-and-check (BaC). Subproblems have been solved using MILP and a combination with CP. The latter hybrid turned out to be most successful. We considered four strategies to construct Benders feasibility cuts. Experiments have shown that a fast greedy approach and the enumeration of all minimum cardinality irreducible infeasible sets (IISs) work best. It is most crucial to base the Benders cuts on IISs to avoid unnecessarily weak cuts. Interestingly, it turned out that the rather time-consuming approach for constructing the minimum cardinality IISs is still competitive. This shows that the mIIS approach has significant potential. The Benders cuts obtained this way are the best we can hope for and our experiments show that computing them is worthwhile. We think that using them might also be interesting for other applications of LBBD.

To speed up solving the Benders master problems we considered heuristic boosting techniques: Instead of always solving the master problem to optimality, we stop once a certain time limit or threshold with respect to the relative optimality gap has been reached. In particular the time limit boosting helped to improve the Benders algorithm, making it possible to find better solutions and to reduce computation times. The sug-

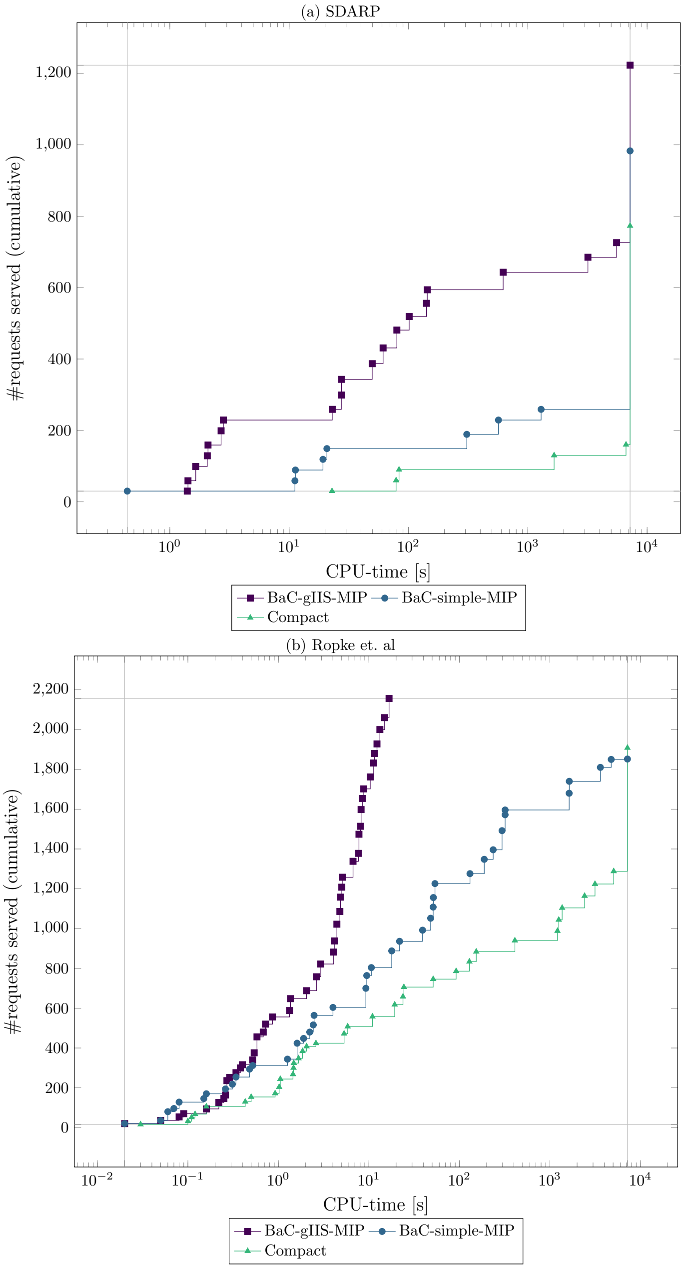
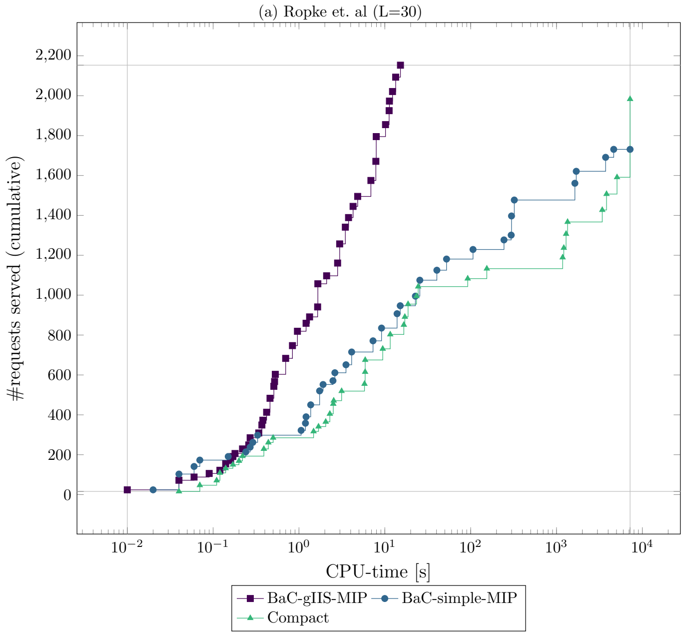


Figure 9: Comparison of BaC with the compact reference MILP model in terms of served requests on the SDARP instances and the instances by Ropke et al. [39]. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality or reaching the time limit. Both charts use a logarithmic x-axis.



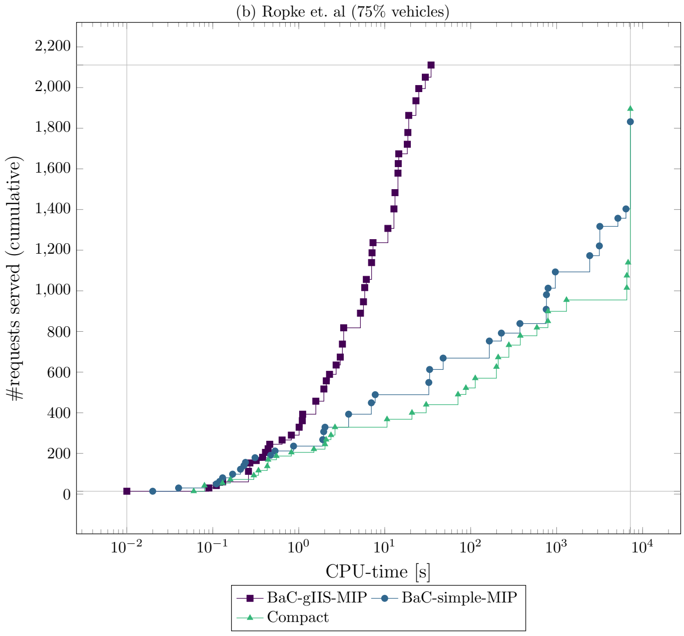


Figure 10: Comparison of BaC with the compact reference MILP model in terms of served requests on the instances by Ropke et al. [39] with the modifications by Berbeglia et al. [5]. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality or reaching the time limit. Both charts use a logarithmic x-axis.

gested boosting techniques are conceptually simple and more generally promising also in the context of LBBD approaches for other applications.

Comparing LBBD and BaC we observed that in general the former excels at computing good dual bounds whereas the latter is superior for computing primal bounds. This effect can be reduced by including an algorithm for repairing infeasible master solutions obtained by LBBD. In situations where solving the master problem is time-consuming the boosting techniques may provide a middle way. For the SDARP instances, they slightly decrease progress with respect to the dual bound but provide a significant speedup for finding good primal bounds.

In general, we can draw the conclusion that the decomposition approach works best if the proportion of requests that can be accepted is not too low.

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.

In our experiments heuristic boosting techniques turned out to be beneficial for solving the master problem. Applying a similar strategy for the subproblems did not work that well. However, by replacing the basic heuristic we considered with a more sophisticated approach it might be possible to also speed up solving the subproblems.

We considered four strategies for constructing Benders feasibility cuts. Our algorithms are based on enumeration and a greedy approach. In this respect it would be interesting to design problem specific approaches that are able to find structures close to the minimum cardinality IISs requiring less time than enumeration. The work by Häme and Hakula [24] could serve as a starting point for research in this direction.

References

[1] D. G. Baklagis, G. Dikas, and I. Minis. The team orienteering pick-up and delivery problem with time windows and its applications in fleet sizing. *RAIRO-Operations Research*, 50(3):503–517, 2016.

[2] R. Baldacci, E. Bartolini, and A. Mingozzi. An exact algorithm for the pickup and delivery problem with time windows. *Operations Research*, 59(2):414–426, 2011.

[3] J. W. Baugh, jr., G. K. R. Kakivaya, and J. R. Stone. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. *Engineering Optimization*, 30(2):91–123, 1998.

[4] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[5] G. Berbeglia, G. Pesant, and L.-M. Rousseau. Checking the feasibility of dial-a-ride instances using constraint programming. *Transportation Science*, 45(3):399–412, 2011.

[6] L. D. Bodin and T. Sexton. The multi-vehicle subscriber dial-a-ride problem. *TIMS studies in Management Science*, 2:73–86, 1986.

[7] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[8] A. A. Cire and J. N. Hooker. A heuristic logic-based Benders method for the home health care problem. Technical report, Tepper School of Business, Carnegie Mellon University, 2012.

[9] G. Codato and M. Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

[10] J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[11] J.-F. Cordeau and G. Laporte. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2):89–101, 2003.

[12] J.-F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003.

[13] J.-F. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

[14] J. Desrosiers, Y. Dumas, F. Soumis, S. Taillefer, and D. Villeneuve. An algorithm for mini-clustering in handicapped transport. Technical Report G-91-02, GERAD, HEC Montréal, 1991.

[15] Y. Dumas, J. Desrosiers, and F. Soumis. Large scale multi-vehicle dial-a-ride problems. Technical Report G-89-30, GERAD, HEC Montréal, 1989.

[16] M. M. Fazel-Zarandi and J. C. Beck. Using logic-based Benders decomposition to solve the capacity- and distance-constrained plant location problem. *INFORMS Journal on Computing*, 24(3):387–398, 2012.

[17] M. Gansterer, M. Küçüktepe, and R. F. Hartl. The multi-vehicle profitable pickup and delivery problem. *OR Spectrum*, 39(1):303–319, 2017.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[19] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, pages 10–22, New York, NY, USA, 1976. ACM.

[20] M. Garg and J. C. Smith. Models and algorithms for the design of survivable multicommodity flow networks with general failure scenarios. *Omega*, 36(6):1057–1071, 2008.

[21] Gecode Team. Gecode: Generic constraint development environment, 2017. Available from `http://www.gecode.org`.

[22] A. M. Geoffrion. Generalized Benders decomposition. *Journal of optimization theory and applications*, 10(4):237–260, 1972.

[23] I. Hamdi and T. Loukil. Logic-based Benders decomposition to solve the permutation flowshop scheduling problem with time lags. In *Modeling, Simulation and Applied Optimization (ICMSAO), 2013 5th International Conference on*, pages 1–7, 2013.

[24] L. Häme and H. Hakula. A maximum cluster algorithm for checking the feasibility of dial-a-ride instances. *Transportation Science*, 49(2):295–310, 2015.

[25] J. N. Hooker. *Logic-based Branch and Bound*, chapter 9, pages 149–161. John Wiley & Sons, Inc., 2000. ISBN 9781118033036.

[26] J. N. Hooker. Planning and scheduling by logic-based Benders decomposition. *Operations Research*, 55(3):588–602, 2007.

[27] J. N Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[28] I. Ioachim, J. Desrosiers, I. Dumas, M. M. Solomon, and D. Villeneuve. A request clustering algorithm for door-to-door handicapped transportation. *Transportation Science*, 29(1):63–78, 1995.

[29] S. Jain and P. Van Hentenryck. Large Neighborhood Search for Dial-a-Ride Problems. In J. Lee, editor, *International Conference on Principles and Practice of Constraint Programming — CP 2011*, volume 6876 of *Lecture Notes in Computer Science*, pages 400–413. Springer, Berlin, Heidelberg, 2011.

[30] J.-J. Jaw, A. R. Odoni, H. N. Psaraftis, and N. H. M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257, 1986.

[31] J. Paquette, F. Bellavance, J.-F. Cordeau, and G. Laporte. Measuring quality of service in dial-a-ride operations: The case of a Canadian city. *Transportation*, 39 (3):539–564, 2012.

[32] S. N. Parragh, K. F. Dörner, and R. F. Hartl. A survey on pickup and delivery models : Part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58:81–117, 2008.

[33] H. N. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

[34] X. Qiu and S. Feuerriegel. A multi-vehicle profitable pickup and delivery selection problem with time windows. In *Proceedings of the European Conference on Information Systems (ECIS) 2014*, 2014.

[35] X. Qiu, S. Feuerriegel, and D. Neumann. Making the most of fleets: A profit-maximizing multi-vehicle pickup and delivery selection problem. *European Journal of Operational Research*, 259(1):155–168, 2017.

[36] G. R. Raidl, T. Baumhauer, and B. Hu. Boosting an exact logic-based Benders decomposition approach by variable neighborhood search. In *Proceedings of the 3rd International Conference on Variable Neighborhood Search*, volume 47 of *Electronic Notes in Discrete Mathematics*, pages 149–156. Elsevier, 2014.

[37] G. R. Raidl, T. Baumhauer, and B. Hu. Speeding up logic-based Benders' decomposition by a metaheuristic for a bi-level capacitated vehicle routing problem. In M. J. Blesa, C. Blum, and S. Voß, editors, *Hybrid Metaheuristics*, volume 8457 of *Lecture Notes in Computer Science*, pages 183–197. Springer International Publishing, 2014.

[38] S. Ropke and J.-F. Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[39] S. Ropke, J.-F. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

[40] T. R. Sexton and L. D. Bodin. Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling. *Transportation Science*, 19(4):378–410, 1985.

[41] T. R. Sexton and L. D. Bodin. Optimizing single vehicle many-to-many operations with desired delivery times: II. Routing. *Transportation Science*, 19(4):411–435, 1985.

[42] E. S. Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *Principles and Practice of Constraint Programming — CP 2001*, pages 16–30. Springer, 2001.

[43] T. T. Tran and J. C. Beck. Logic-based benders decomposition for alternative resource scheduling with sequence dependent setups. In *Proceedings of the 20th European Conference on Artificial Intelligence*, ECAI'12, pages 774–779, Amsterdam, The Netherlands, The Netherlands, 2012. IOS Press. ISBN 978-1-61499-097-0.

[44] D. Wheatley, F. Gzara, and E. Jewkes. Logic-based Benders decomposition for an inventory-location problem with service constraints. *Omega*, 55:10–23, 2015.

[45] R. Wolfler Calvo and A. Colorni. An effective and fast heuristic for the dial-a-ride problem. *4OR*, 5(1):61–73, 2007.

A. Additional Result Tables

	Instance Properties						Computation Time [s]									LB								
								LBBD				BaC					LBBD				BaC			
Instance	$\|K\|$	n	T	Q	L	LB*	CM	simple	aIIS	mIIS	gIIS	simple	aIIS	mIIS	gIIS	CM	simple	aIIS	mIIS	gIIS	simple	aIIS	mIIS	gIIS
30N_4K_A	4	30	240	3	30	**30**	83	3	53	38	6	11	24	23	**2**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_4K_B	4	30	240	3	30	**29**	ML	595	15	12	4	11	12	3	**1**	ML	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**
30N_4K_C	4	30	240	3	30	**30**	ML	TL	355	202	47	1293	195	192	**23**	ML	29	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_A	5	30	240	3	30	**30**	79	< **1**	112	4	1	21	30	34	2	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_B	5	30	240	3	30	**30**	23	3	13	4	1	< **1**	13	11	1	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_C	5	30	240	3	30	**30**	6655	406	137	75	19	19	52	25	**3**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
40N_4K_A	4	40	240	3	30	**38**	TL	TL	TL	343	**71**	TL	6053	496	101	36	36	**38**	**38**	**38**	36	**38**	**38**	**38**
40N_4K_B	4	40	240	3	30	**38**	ML	TL	6798	375	282	TL	3132	540	**144**	ML	36	**38**	**38**	**38**	29	**38**	**38**	**38**
40N_4K_C	4	40	232	3	30	**37**	TL	TL	TL	798	802	TL	TL	454	**142**	29	34	36	**37**	**37**	28	**37**	**37**	**37**
40N_5K_A	5	40	240	3	30	**40**	TL	35	2669	68	5	568	77	33	**3**	31	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
40N_5K_B	5	40	240	3	30	**40**	TL	TL	1499	327	97	TL	430	172	**27**	39	38	**40**	**40**	**40**	39	**40**	**40**	**40**
40N_5K_C	5	40	240	3	30	**40**	1662	10	29	53	14	308	108	5	**2**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
44N_4K_A	4	44	240	3	30	40	TL	TL	TL	TL	TL	TL	TL	TL	TL	38	36	36	39	38	24	38	**40**	**40**
44N_4K_B	4	44	240	3	30	**42**	TL	TL	TL	1959	**1088**	TL	TL	4279	3191	30	38	40	**42**	**42**	34	41	**42**	**42**
44N_4K_C	4	44	240	3	30	**41**	TL	TL	TL	TL	TL	TL	TL	**4740**	5553	32	37	35	40	39	33	40	**41**	**41**
44N_5K_A	5	44	239	3	30	**44**	ML	TL	TL	323	131	TL	429	339	**61**	ML	42	43	**44**	**44**	30	**44**	**44**	**44**
44N_5K_B	5	44	240	3	30	**44**	TL	TL	4173	1821	99	TL	872	155	**27**	42	43	**44**	**44**	**44**	38	**44**	**44**	**44**
44N_5K_C	5	44	240	3	30	**44**	ML	TL	5491	938	106	TL	652	514	**50**	ML	42	**44**	**44**	**44**	40	**44**	**44**	**44**
50N_4K_A	4	50	240	3	30	41	TL	TL	TL	TL	TL	TL	TL	TL	TL	26	37	35	38	39	26	39	**40**	**40**
50N_4K_B	4	50	240	3	30	43	TL	TL	ML	TL	TL	TL	TL	TL	TL	35	38	ML	39	39	34	35	**42**	41
50N_4K_C	4	50	240	3	30	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	30	39	35	41	40	31	41	**43**	**43**
50N_5K_A	5	50	240	3	30	48	TL	TL	TL	TL	TL	TL	TL	TL	TL	19	44	39	45	46	40	**47**	**47**	**47**
50N_5K_B	5	50	240	3	30	**49**	TL	TL	TL	1667	1008	TL	TL	1856	**621**	36	44	42	**49**	**49**	30	48	**49**	**49**
50N_5K_C	5	50	240	3	30	**50**	TL	TL	TL	5480	1426	TL	6481	892	**80**	36	47	48	**50**	**50**	43	**50**	**50**	**50**
60N_4K_A	4	60	240	3	30	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	29	40	34	39	39	26	39	42	**43**
60N_4K_B	4	60	240	3	30	45	TL	TL	TL	TL	TL	TL	TL	TL	TL	-	42	0	44	41	25	-	**45**	44
60N_4K_C	4	60	240	3	30	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	27	41	38	41	41	34	37	**43**	42
60N_5K_A	5	60	240	3	30	56	TL	TL	TL	TL	TL	TL	TL	TL	TL	37	52	0	54	54	35	-	52	**56**
60N_5K_B	5	60	240	3	30	50	TL	TL	TL	TL	TL	TL	TL	TL	TL	25	45	44	46	45	30	45	**50**	49
60N_5K_C	5	60	240	3	30	53	TL	TL	TL	TL	TL	TL	TL	TL	TL	36	47	43	49	47	39	48	51	**52**

Table 8: 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.

		Computation Time [s]								LB							
		LBBD				BaC				LBBD				BaC			
Instance	LB*	simple	aIIS	mIIS	gIIS	simple	aIIS	mIIS	gIIS	simple	aIIS	mIIS	gIIS	simple	aIIS	mIIS	gIIS
30N_4K_A	**30**	2	10	4	1	5	4	3	**< 1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_4K_B	**29**	467	2	2	1	7	2	**< 1**	**< 1**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**
30N_4K_C	**30**	TL	95	25	10	754	31	15	**3**	29	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_A	**30**	**< 1**	20	**< 1**	**< 1**	8	4	4	**< 1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_B	**30**	1	2	1	**< 1**	**< 1**	2	1	**< 1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_C	**30**	349	24	11	3	6	6	2	**< 1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
40N_4K_A	**38**	TL	5391	111	**63**	TL	5670	99	68	36	**38**	**38**	**38**	36	**38**	**38**	**38**
40N_4K_B	**38**	TL	6181	90	209	TL	2908	97	**69**	36	**38**	**38**	**38**	29	**38**	**38**	**38**
40N_4K_C	**37**	TL	TL	589	760	TL	TL	**125**	135	34	36	**37**	**37**	28	**37**	**37**	**37**
40N_5K_A	**40**	23	2747	16	1	747	14	4	**< 1**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
40N_5K_B	**40**	TL	1015	84	57	TL	129	23	**9**	38	**40**	**40**	**40**	39	**40**	**40**	**40**
40N_5K_C	**40**	5	5	9	7	259	19	1	**< 1**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
44N_4K_A	40	TL	TL	TL	TL	TL	TL	TL	TL	36	36	39	38	24	39	**40**	**40**
44N_4K_B	**42**	TL	TL	**750**	1025	TL	TL	1550	2768	38	40	**42**	**42**	34	41	**42**	**42**
44N_4K_C	**41**	TL	TL	TL	TL	TL	TL	**3134**	5552	37	35	40	39	35	40	**41**	**41**
44N_5K_A	**44**	TL	6252	97	104	TL	126	65	**21**	42	**44**	**44**	**44**	27	**44**	**44**	**44**
44N_5K_B	**44**	TL	3141	520	47	TL	263	18	**12**	43	**44**	**44**	**44**	38	**44**	**44**	**44**
44N_5K_C	**44**	TL	4576	199	57	TL	252	64	**16**	42	**44**	**44**	**44**	40	**44**	**44**	**44**
50N_4K_A	41	TL	TL	TL	TL	TL	TL	TL	TL	37	35	38	39	22	39	**41**	**41**
50N_4K_B	43	TL	ML	TL	TL	TL	TL	TL	TL	38	ML	39	39	34	34	**42**	41
50N_4K_C	44	TL	TL	TL	TL	TL	TL	TL	TL	39	35	41	40	31	41	**43**	**43**
50N_5K_A	48	TL	TL	TL	TL	TL	TL	TL	TL	45	39	45	46	40	47	**48**	**48**
50N_5K_B	**49**	TL	ML	778	791	TL	TL	484	**447**	44	ML	**49**	**49**	30	48	**49**	**49**
50N_5K_C	**50**	TL	TL	1458	933	TL	4343	115	**19**	47	48	**50**	**50**	43	**50**	**50**	**50**
60N_4K_A	44	TL	TL	TL	TL	TL	TL	TL	TL	40	34	39	39	26	40	42	**43**
60N_4K_B	45	TL	TL	TL	TL	TL	TL	TL	TL	42	0	44	41	25	-	**45**	44
60N_4K_C	44	TL	TL	TL	TL	TL	TL	TL	TL	41	38	41	41	34	39	**43**	42
60N_5K_A	56	TL	TL	TL	TL	TL	ML	TL	TL	52	46	55	54	35	ML	54	**56**
60N_5K_B	50	TL	ML	TL	TL	TL	TL	TL	TL	45	ML	46	45	30	46	**50**	48
60N_5K_C	53	TL	TL	TL	TL	TL	TL	TL	TL	47	43	49	47	39	48	51	**52**

Table 9: Overview of the computation times and lower bounds of the un-boosted LBBD and BaC algorithm variants using a CP-MILP combination for the subproblems. Column LB* shows the best known lower bounds (provably optimal solution values are marked bold). 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.

		Computation Time [s]										LB									
		standard		gap (rel) - it		gap (rel) - ud		time - it		time - ud		standard		gap (rel) - it		gap (rel) - ud		time - it		time - ud	
Instance	LB*	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS
30N_4K_A	**30**	4	**1**	**1**	**1**	**1**	**1**	4	**1**	4	**1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_4K_B	**29**	2	**1**	2	**1**	2	**1**	2	**1**	2	**1**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**
30N_4K_C	**30**	25	**10**	36	19	42	22	26	**10**	25	**10**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_A	**30**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_B	**30**	1	**< 1**	1	**< 1**	1	**< 1**	1	**< 1**	1	**< 1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_C	**30**	11	**3**	8	7	14	4	11	**3**	11	**3**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
40N_4K_A	**38**	111	**63**	145	81	233	107	113	69	113	65	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**
40N_4K_B	**38**	90	209	108	**71**	108	79	91	146	92	134	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**
40N_4K_C	**37**	589	760	210	**173**	248	265	352	671	353	460	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**
40N_5K_A	**40**	16	**1**	33	**1**	33	**1**	16	**1**	16	**1**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
40N_5K_B	**40**	84	**57**	112	83	312	298	88	58	87	59	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
40N_5K_C	**40**	9	7	7	**6**	7	**6**	8	7	9	7	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
44N_4K_A	40	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	39	38	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
44N_4K_B	**42**	**750**	1025	1674	930	2850	2100	1254	1224	1203	1664	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**
44N_4K_C	**41**	TL	TL	TL	TL	4428	2939	4439	**2721**	6534	3560	40	39	40	40	**41**	**41**	**41**	**41**	**41**	**41**
44N_5K_A	**44**	97	104	266	227	994	534	**95**	105	98	106	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**
44N_5K_B	**44**	520	47	368	**45**	461	255	517	48	522	49	**44**	**44**	**44**	**44**	**44**	43	**44**	**44**	**44**	**44**
44N_5K_C	**44**	199	**57**	388	178	1335	1354	195	58	205	59	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**
50N_4K_A	41	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	38	39	39	39	39	39	**41**	40	**41**	**41**
50N_4K_B	43	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	39	39	40	40	40	40	**43**	42	**43**	42
50N_4K_C	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	41	40	42	43	42	43	**44**	**44**	**44**	**44**
50N_5K_A	48	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	45	46	**48**	47	47	**48**	47	47	**48**	47
50N_5K_B	**49**	**778**	791	873	878	2785	2234	1079	918	882	1137	**49**	**49**	**49**	**49**	**49**	**49**	**49**	**49**	**49**	**49**
50N_5K_C	**50**	1458	933	1670	693	1593	TL	965	1449	987	**644**	**50**	**50**	**50**	**50**	**50**	49	**50**	**50**	**50**	**50**
60N_4K_A	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	39	39	40	42	40	42	**44**	43	**44**	43
60N_4K_B	45	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	44	41	42	44	42	44	**45**	44	**45**	44
60N_4K_C	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	41	41	41	40	41	40	**44**	43	**44**	43
60N_5K_A	56	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	**55**	54	54	54	54	54	**55**	54	**55**	**55**
60N_5K_B	50	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	46	45	48	48	48	48	**50**	49	**50**	49
60N_5K_C	53	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	49	47	50	51	50	51	**53**	52	**53**	52

Table 10: 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 threshold 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.

	Iterations								Master-Sub Ratio								Cuts							
	LBBD								LBBD								LBBD							
	un-boosted		time - it		time - ud		BaC		un-boosted		time - it		time - ud		BaC		un-boosted		time - it		time - ud		BaC	
Instance	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS
30N_4K_A	24	19	24	19	24	19	23	14	0.07	0.17	0.06	0.19	0.06	0.13	0.02	0.07	400	280	400	280	400	280	232	**184**
30N_4K_B	14	16	14	16	14	16	5	19	0.36	1.63	0.39	1.79	0.40	1.66	0.28	0.57	240	208	240	208	240	208	**148**	152
30N_4K_C	86	92	86	92	86	92	69	80	0.28	1.28	0.28	1.29	0.29	1.38	0.01	0.08	1624	1668	1624	1668	1624	1668	**1036**	1060
30N_5K_A	3	3	3	3	3	3	33	17	0.07	0.43	0.02	0.33	0.02	0.43	0.01	0.15	165	**45**	165	**45**	165	**45**	485	220
30N_5K_B	4	7	4	7	4	7	10	17	0.09	0.45	0.09	0.43	0.07	0.36	0.02	0.12	230	**110**	230	**110**	230	**110**	260	265
30N_5K_C	46	48	46	48	46	48	31	25	0.15	0.97	0.17	1.01	0.16	1.06	0.01	0.34	975	960	975	960	975	960	520	**425**
40N_4K_A	78	97	78	97	78	97	97	142	0.61	1.21	0.64	1.37	0.69	1.21	0.18	2.94	**1832**	2032	**1832**	2032	**1832**	2032	1888	2160
40N_4K_B	76	127	76	127	76	127	128	209	0.92	15.26	0.94	9.73	1.02	8.75	0.50	3.80	**2096**	2604	**2096**	2604	**2096**	2604	2408	3060
40N_4K_C	86	145	131	195	131	169	155	227	12.60	17.72	3.73	9.37	3.74	8.57	1.03	1.95	**2260**	3128	2860	3556	2860	3288	3012	3688
40N_5K_A	34	15	34	15	34	15	19	18	0.06	0.22	0.07	0.30	0.07	0.32	0.02	0.07	1075	385	1075	385	1075	385	260	**240**
40N_5K_B	124	135	124	135	124	135	100	95	0.54	0.78	0.59	0.85	0.58	0.83	0.03	0.06	3510	3710	3510	3710	3510	3710	2145	**2045**
40N_5K_C	27	37	27	37	27	37	8	11	0.07	0.17	0.08	0.20	0.07	0.20	0.08	0.07	985	1045	985	1045	985	1045	**150**	170
44N_4K_A	83	151	187	299	317	431	297	393	116.49	215.85	42.49	124.35	28.81	100.03	32.64	153.95	**3616**	4084	5440	6592	6716	8056	6544	7292
44N_4K_B	153	225	251	304	244	374	429	391	1.77	14.27	1.84	13.65	1.94	16.67	1.91	29.12	**4016**	5324	5280	6468	5248	7244	7184	7268
44N_4K_C	168	212	239	274	430	383	310	343	28.33	108.24	14.04	30.19	11.92	28.59	10.66	76.03	**4988**	5340	5632	6020	7216	7084	5716	6200
44N_5K_A	94	136	94	136	94	136	112	162	0.22	0.30	0.23	0.29	0.23	0.30	0.02	0.07	3515	4005	3515	4005	3515	4005	**2655**	3125
44N_5K_B	389	144	389	144	389	144	51	103	0.39	0.50	0.38	0.49	0.39	0.51	0.01	0.03	8645	3890	8645	3890	8645	3890	**1175**	1885
44N_5K_C	214	151	214	151	214	151	136	207	0.58	0.69	0.61	0.67	0.62	0.71	0.02	0.22	6905	4870	6905	4870	6905	4870	**3045**	4265
50N_4K_A	67	192	413	632	604	636	738	1183	64.76	38.11	6.45	13.69	4.29	13.87	4.48	13.09	**4812**	5992	12316	16392	14672	15752	16104	23948
50N_4K_B	52	144	407	483	598	433	570	812	87.73	155.20	4.36	32.39	2.74	36.75	4.28	25.16	**3292**	4416	10796	12548	13576	11364	12188	16860
50N_4K_C	46	96	196	288	309	383	499	609	112.90	130.09	16.46	53.28	10.35	43.14	6.65	36.94	**2472**	2920	5808	7332	7440	8764	9164	11632
50N_5K_A	192	298	683	774	959	1391	1221	1563	27.13	112.20	4.30	35.56	3.09	17.11	2.91	18.35	**9865**	11540	23755	25050	28520	37945	34120	42425
50N_5K_B	154	217	231	273	206	310	296	353	3.69	20.97	2.84	20.25	2.97	23.24	1.08	11.24	**6085**	7425	7895	8745	7335	9355	7820	9245
50N_5K_C	513	568	390	687	390	481	173	167	1.00	10.23	0.79	13.36	0.82	8.63	0.02	0.11	14380	16005	12040	18335	12040	14260	**4260**	4490
60N_4K_A	17	93	609	864	586	892	818	915	385.39	67.87	1.25	5.09	1.12	4.41	1.77	12.19	**2504**	2916	17976	23140	17336	23424	19428	20368
60N_4K_B	50	243	528	887	527	921	777	1650	45.90	16.46	0.83	2.73	0.84	3.00	1.14	3.18	**5396**	7708	21196	26776	21424	27680	24700	39880
60N_4K_C	36	123	790	1120	801	1159	908	1378	117.79	89.90	1.65	11.66	1.74	10.45	3.05	15.33	**3644**	3896	22648	28900	22572	29036	22684	31824
60N_5K_A	289	541	619	1183	778	1240	1148	1571	4.49	11.38	1.41	5.59	1.05	5.14	0.83	5.34	**17390**	24160	28320	45580	33010	46330	33125	50675
60N_5K_B	46	150	760	1196	1018	1346	813	1404	181.78	269.00	4.93	24.13	3.10	23.99	6.57	35.34	**5820**	7170	32975	43680	39225	45890	28525	43365
60N_5K_C	50	126	226	791	495	1260	776	742	120.93	320.61	17.22	43.90	6.58	30.95	4.94	59.57	**5075**	5870	12860	27230	19935	35905	25720	24320

Table 11: Characteristics of the decomposition approaches. Column iterations states the number of iterations the algorithm completed. For the LBBD approaches this corresponds to the number of times the master problem has been solved. For the BaC approaches it is equal to the number of times the separation routine has been called. Column master-sub ratio provides the relative ratio of time spent in the master problem compared to those spent in the subproblems ($t_{\text{master}}/t_{\text{sub}}$). The last column (cuts) shows the total number of Benders cuts that have been added.

			LB gap [%]									UB gap [%]								
				LBBD									LBBD							
				un-boosted		time - it		time - ud		BaC			un-boosted		time - it		time - ud		BaC	
Instance	LB*	UB*	CM	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	CM	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS
30N_4K_A	**30**	**30**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
30N_4K_B	**29**	**29**	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
30N_4K_C	**30**	**30**	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
30N_5K_A	**30**	**30**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
30N_5K_B	**30**	**30**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
30N_5K_C	**30**	**30**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
40N_4K_A	**38**	**38**	5.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	5.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
40N_4K_B	**38**	**38**	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
40N_4K_C	**37**	**37**	21.6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	8.1	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
40N_5K_A	**40**	**40**	22.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
40N_5K_B	**40**	**40**	2.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
40N_5K_C	**40**	**40**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
44N_4K_A	40	41	5.0	2.5	5.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	7.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	3.3	3.9
44N_4K_B	**42**	**42**	28.6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	4.8	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
44N_4K_C	**41**	**41**	22.0	2.4	4.9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	7.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
44N_5K_A	**44**	**44**	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
44N_5K_B	**44**	**44**	4.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
44N_5K_C	**44**	**44**	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
50N_4K_A	41	44	36.6	7.3	4.9	**0.0**	2.4	**0.0**	**0.0**	**0.0**	**0.0**	13.6	**0.0**	4.5	**0.0**	6.8	**0.0**	**0.0**	9.1	9.1
50N_4K_B	43	47	18.6	9.3	9.3	**0.0**	2.3	**0.0**	2.3	2.3	4.7	6.4	**0.0**	**0.0**	**0.0**	2.1	2.1	**0.0**	4.3	4.3
50N_4K_C	44	46	31.8	6.8	9.1	**0.0**	**0.0**	**0.0**	**0.0**	2.3	2.3	8.7	2.2	2.2	2.2	**0.0**	**0.0**	2.2	6.2	6.5
50N_5K_A	48	49	60.4	6.2	4.2	2.1	2.1	**0.0**	2.1	**0.0**	**0.0**	2.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2.0	2.0	2.0
50N_5K_B	**49**	**49**	26.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
50N_5K_C	**50**	**50**	28.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
60N_4K_A	44	56	34.1	11.4	11.4	**0.0**	2.3	**0.0**	2.3	4.5	2.3	7.1	**0.0**	3.6	3.6	5.4	3.6	5.4	5.7	5.4
60N_4K_B	45	56	-	2.2	8.9	**0.0**	2.2	**0.0**	2.2	**0.0**	2.2	7.1	**0.0**	1.8	5.4	5.4	5.4	5.4	6.9	6.5
60N_4K_C	44	53	38.6	6.8	6.8	**0.0**	2.3	**0.0**	2.3	2.3	4.5	13.2	**0.0**	1.9	7.5	7.5	7.5	7.5	8.5	9.4
60N_5K_A	56	60	33.9	1.8	3.6	1.8	3.6	1.8	1.8	3.6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
60N_5K_B	50	59	50.0	8.0	10.0	**0.0**	2.0	**0.0**	2.0	**0.0**	4.0	1.7	**0.0**	**0.0**	**0.0**	1.7	**0.0**	1.7	1.7	1.7
60N_5K_C	53	58	32.1	7.5	11.3	**0.0**	1.9	**0.0**	1.9	3.8	1.9	3.4	**0.0**	**0.0**	**0.0**	1.7	**0.0**	1.7	3.4	3.4

Table 12: Overview of relative gaps to the best known lower (column LB gap) and upper (column UB gap) bounds, respectively. Columns LB* and UB* report the best known lower and upper bounds obtained across all algorithms. Entries in column LB* are marked bold if the corresponding solution is provably optimal. Per instance smallest gaps are marked bold.