Cut-First Branch-and-Price-Second for the Capacitated Arc-Routing Problem

Claudia Bode[a], Stefan Irnich[a]

[a] *Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany.*

Abstract

This paper presents the first full-fledged branch-and-price (bap) algorithm for the capacitated arc-routing problem (CARP). Prior exact solution techniques either rely on cutting planes or the transformation of the CARP into a node-routing problem. The drawbacks are either models with inherent symmetry, dense underlying networks, or a formulation where edge flows in a potential solution do not allow the reconstruction of unique CARP tours. The proposed algorithm circumvents all these drawbacks by taking the beneficial ingredients from existing CARP methods and combining them in a new way. The first step is the solution of the one-index formulation of the CARP in order to produce strong cuts and an excellent lower bound. It is known that this bound is typically stronger than relaxations of a pure set-partitioning CARP model. Such a set-partitioning master program results from a Dantzig-Wolfe decomposition. In the second phase, the master program is initialized with the strong cuts, CARP tours are iteratively generated by a pricing procedure, and branching is required to produce integer solutions. This is a cut-first bap-second algorithm and its main function is, in fact, the splitting of edge flows into unique CARP tours.

Key words: Capacitated arc-routing problem, column generation, branch-and-price, dual-optimal inequalities

1. Introduction

The capacitated arc-routing problem (CARP) is the basic multiple-vehicle arc-routing problem and has applications in waste collection, postal delivery, winter services such as snow plowing and salt gritting, meter reading, school bus routing and more. It was first introduced by Golden and Wong (1981) and has received a lot of attention since then; see for instance the edited book by Dror (2000) and the annotated bibliography by Corberán and Prins (2010).

This paper presents the first full-fledged branch-and-price (bap) algorithm for the CARP. Prior exact solution techniques either rely on cutting planes or the transformation of the CARP into a node-routing problem. As already pointed out by Letchford and Oukil (2009), the drawbacks are either models with inherent symmetry, dense underlying networks, or a formulation where edge flows in a potential solution do not allow the reconstruction of unique CARP tours. The proposed algorithm circumvents all these drawbacks by taking the beneficial ingredients from existing CARP methods and combining them in a new way. The first step is the solution of the one-index formulation of the CARP in order to produce strong cuts and an excellent lower bound. It is known that this bound is typically stronger than relaxations of a pure set-partitioning CARP model. Such a set-partitioning master program results from Dantzig-Wolfe decomposition. In the second phase, the master program is initialized with the strong cuts, CARP tours are iteratively generated by a pricing procedure, and branching is required to produce integer solutions. This is a cut-first bap-second algorithm and its main function is, in fact, the splitting of edge flows into unique CARP tours.

Email addresses: claudia.bode@uni-mainz.de (Claudia Bode), irnich@uni-mainz.de (Stefan Irnich)

The novelty of our approach comprises of the following aspects: First, pricing of CARP tours is fast because it can be performed on the original network that is sparse for real-world instances. A key property for not being forced to use a transformed network is that deadheading variables can be guaranteed to have non-negative reduced cost. The addition of dual-optimal inequalities (Ben Amor *et al.*, 2006) to the column generation master program is the device to ensure non-negativity. Second, in a CARP solution edges might be traversed more than once. This creates the problem that not all solutions with integer flows on edges impose integer path variables in the master problem. Therefore, a new hierarchical branching scheme is developed that is able to finally guarantee integer CARP solutions while being compatible with the pricing algorithm.

For a formal definition of the CARP, we assume an undirected graph $G = (V, E)$ with node set V and edge set E. Non-negative integer demands $q_e \geq 0$ are located on edges $e \in E$. Those edges with positive demand form the subset $E_R \subset E$ of required edges that have to be serviced exactly once. A fleet of K homogeneous vehicles stationed at depot $d \in V$ with capacity Q is given. The problem is to find minimum cost vehicle tours which start and end at the depot d, service all required edges exactly once, and respect the vehicle capacity Q. The tour costs consist of service costs c_e^{serv} for required edges e that are serviced and deadheading cost c_e whenever an edge e is traversed without servicing.

Throughout the paper we use the following standard notation. For any subset $S \subseteq V$ we denote by $\delta(S)$ the set of edges with exactly one endpoint in S and by $\delta_R(S) = \delta(S) \cap E_R$. For the sake of brevity, we write $\delta(i)$ instead of $\delta(\{i\})$. $E(S)$ is the set of edges with both endpoints in S and $E_R(S) = E(S) \cap E_R$. For any subset $F \subseteq E$ and any parameter or variable y, let be $y(F) = \sum_{e \in F} y_e$.

The remainder of this paper is structured as follows. Section 2 reviews existing exact approaches for the CARP. Section 3 describes the Dantzig-Wolfe decomposition. The key components (cutting, pricing, branching) of the bap algorithm and their implementation are described in Section 4. Section 5 analyzes the interplay between cycle elimination and branching. Computational results in Section 6 show the capability of the new approach. Final conclusions are drawn in Section 7.

2. Review of Models and Methods

In this section, we outline successful MIP-based exact algorithms for the CARP that have been presented in the literature. Two types of approaches can be distinguished: *Full* exact methods determine an optimal integer solution and show optimality by proving that its cost is a lower bound. Methods that use compact MIP models with aggregated variables (see below) also provide a lower bound, but are not able to determine a solution to the CARP. Instead, optimality is proved with the help of a heuristic whenever the heuristic solution matches the lower bound.

2.1. Node-Routing Transformation

Several researchers developed and applied transformations of arc-routing problems into node-routing problems (Pearn *et al.*, 1987; Longo *et al.*, 2006; Baldacci and Maniezzo, 2006). These approaches transform each required edge of the CARP into two or three associated nodes so that the number of nodes in the capacitated vehicle-routing problem (CVRP) is $2|E_R| + 1$ or $3|E_R| + 1$, respectively. The resulting CVRP instance is then solved with any CVRP algorithm.

Exact algorithms for the CVRP have been intensively studied in the past. The currently most successful algorithms are based on branch-and-cut (Lysgaard *et al.*, 2004), branch-and-price-and-cut (Fukasawa *et al.*, 2006), and a set-partitioning and cut-generation based approach that finally applies a standard IP solver (Baldacci *et al.*, 2010). Although these approaches are rather successful for the CVRP, Letchford and Oukil (2009) mentioned the following drawbacks: Even for relatively small CARP instances the resulting CVRP is defined over a larger number of nodes and edges. In particular, the increase in the number of edges can be quadratic as the CVRP graph is complete. As real-world CARP instances are based on street networks with typically very sparse underlying graphs, this effect is significant. Furthermore, specific graph structures allowing tailored CARP algorithms get lost by the transformation and the resulting CVRP instance might feature further symmetries.

2.2. Two-Index Formulation

Belenguer and Benavent (1998a) were the first to develop and analyze the following IP formulation for the CARP. This formulation is also referred to as *sparse* or *two-index formulation*. For every pair of an edge e and a vehicle k there are service and deadheading variables: x_e^k is equal to 1 if vehicle k services edge $e \in E_R$ and 0 otherwise. The variable y_e^k counts the number of times vehicle k traverses edge $e \in E$ without servicing. The two-index formulation is:

$$\min \quad \sum_{k \in K} c^{serv,\top} x^k + \sum_{k \in K} c^\top y^k \tag{1}$$

$$\text{s.t.} \quad \sum_{k \in K} x_e^k = 1 \quad \text{for all } e \in E_R \tag{2}$$

$$x^k(\delta_R(S)) + y^k(\delta(S)) \geq 2x_f^k \quad \text{for all } S \subseteq V \setminus \{d\},\ f \in E_R(S),\ k \in K \tag{3}$$

$$x^k(\delta_R(i)) + y^k(\delta(i)) = 2p_i^k \quad \text{for all } i \in V,\ k \in K \tag{4}$$

$$q^\top x^k \leq Q \quad \text{for all } k \in K \tag{5}$$

$$p^k \in \mathbb{Z}_+^{|V|} \quad \text{for all } k \in K \tag{6}$$

$$x^k \in \{0,1\}^{|E_R|} \quad \text{for all } k \in K \tag{7}$$

$$y^k \in \mathbb{Z}_+^{|E|} \quad \text{for all } k \in K \tag{8}$$

The objective (1) is the minimization of all service and deadheading costs. Since each required edge is serviced exactly once, as stated by (2), service costs c_e^{serv} have no impact on optimal decisions. The formulation in (Belenguer and Benavent, 1998a) therefore omits service costs. Constraints (3) are the subtour-elimination constraints (SEC). As discussed in (Belenguer and Benavent, 1998a, p. 169), the given SEC still allow disconnected components being deadheaded. A corresponding infeasible integer solution, denoted as *extended k-route* by Belenguer and Benavent (1998a), is not optimal and can be excluded by adding constraints of the form (3) with $2x_f^k$ replaced by $2y_f^k$. The parity constraints (4) ensure that each vehicle can leave a node i after entering. The capacity constraints are given by (5) and integrality constraints by (6)–(8).

The two-index formulation has two major drawbacks: First, the number of variables increases with the fleet size K. Second, the inherent symmetry with respect to the numbering of vehicles lets branch-and-bound based algorithms perform poorly. The computational results in (Belenguer and Benavent, 2003; Ahr, 2004) show that the two-index formulation can work well for small $K \leq 5$, but is not suited to solve CARP instances with a larger fleet.

2.3. One-Index Formulation

The one-index formulation, first considered independently by Letchford (1997) and Belenguer and Benavent (1998a), solely uses aggregated deadheading variables

$$y_e = \sum_{k \in K} y_e^k \in \mathbb{Z}_+,$$

one for each edge $e \in E$.

A formulation with deadheading variables alone seems appealing due to the small number of variables and, even more important, due to the eliminated symmetry regarding the numbering of the vehicles $k \in K$. Notably, no IP formulation with aggregated deadheading variables $y_e \in \mathbb{Z}_+$ alone is known for the CARP. In fact, the integer polyhedron of the following CARP model is a relaxation of the CARP and can therefore contain infeasible integer solutions (see Belenguer and Benavent, 2003, p. 709). Even worse, a feasible integer solution to the CARP that is represented by the deadheading variables y_e of the one-index formulation is not helpful. The reconstruction of tours from deadheading variables is $\mathcal{NP}$-hard (and typically also a hard problem in practice). The bap phase of our solution approach can be interpreted as such a flow decomposition algorithm.

The usefulness of the one-index formulation is, however, that its LP-relaxation often produces a very tight lower bound:

$$\begin{aligned}
\min \quad & c^\top y && (9)\\
\text{s.t.} \quad & y(\delta(S)) \geq 2K(S) - |\delta_R(S)| \quad \text{for all } \emptyset \neq S \subseteq V \setminus \{d\} && (10)\\
& y(\delta(S)) \geq 1 \quad \text{for all } \emptyset \neq S \subseteq V,\ |\delta_R(S)| \text{ odd} && (11)\\
& y \in \mathbb{Z}_+^{|E|} && (12)
\end{aligned}$$

The objective (9) just takes the cost for deadheadings into account as service costs are constant. The capacity inequalities (10) require that there are at least $2K(S)$ traversals (services and deadheadings) over the cutset $\delta(S)$. Thus, $K(S)$ is the minimum number of vehicles necessary to serve $E_R(S) \cup \delta_R(S)$ which can be approximated by $\lceil q(E_R(S) \cup \delta_R(S))/Q \rceil$ and computed exactly solving a bin-packing problem. The odd-cut inequalities (11) require at least one deadheading if there is an odd number of required arcs in the cut $\delta(S)$.

In combination with a powerful CARP heuristic, the one-index formulation provides a possible exact algorithm for the CARP: Only if the heuristic has comes across an optimal solution, the lower bound provided by (9)–(12) might prove optimality (as benchmark problems typically have integral costs, a gap less than one suffices).

Disjoint-path inequalities are another class of valid inequalities first considered by Belenguer and Benavent (2003). For the development of our model, it suffices to know that the general form of all valid inequalities of the one-index formulation is

$$\sum_{e \in E} d_{es} y_e \geq r_s \quad s \in \mathcal{S}, \tag{13}$$

where s is the index referring to a particular inequality and $\mathcal{S}$ the set of all valid inequalities. Some details and references on separation procedures are provided in Section 4.1.

2.4. Extended Set-Covering Approach

Gómez-Cabrero *et al.* (2005) use a set-covering approach in which the standard covering/partitioning constraints are supplemented with the capacity (10) and odd-cut inequalities (11). As for the one-index formulation, the focus is on generating an excellent lower bound by solving the LP-relaxation of the model. As the number of tours and cuts grows exponentially with the size of the instance, both row and column generation is required. Opposed to our approach, cutting planes are added after a solution to the LP-relaxation of the initial or an extended set-covering model has been computed. Details on pricing out new routes and separating violated cuts will be discussed in comparison with the proposed cut-first bap-second approach in Section 4.2.

Let c_r indicate the cost of a route $r \in \Omega$ and let $\bar{x}_{er} \in \{0, 1\}$ and $\bar{y}_{er} \in \mathbb{Z}_+$ be the number of times that route r services and deadheads through edge e, respectively. (We distinguish between decision variables x, y and their values or coefficients $\bar{x}, \bar{y}$.) The extended set-covering model for the CARP is defined as follows:

$$\begin{aligned}
\min \quad & \sum_{r \in \Omega} c_r \lambda_r && (14)\\
\text{s.t.} \quad & \sum_{r \in \Omega} \bar{x}_{er} \lambda_r \geq 1 \quad \text{for all } e \in E_R && (15)\\
& \sum_{r \in \Omega} d_{sr} \lambda_r \geq r_s \quad \text{for all } s \in \mathcal{S} && (16)\\
& \lambda_r \in \{0, 1\} \quad \text{for all } r \in \Omega && (17)
\end{aligned}$$

The extension to the standard set-covering model (14), (15) and (17) is the addition of transformed cuts (16) derived from (13), i.e., the constraints of the one-index formulation. Herein, d_{sr} is the coefficient of the transformed cut $s \in \mathcal{S}$ for route r, which is $d_{sr} = \sum_{e \in E} d_{es} \bar{y}_{er}$.

Gómez-Cabrero *et al.* (2005) presented computational results for the LP-relaxation of (14)–(17). In their approach, they allow non-elementary tours in the sense that a tour may service a required edge more than once. This relaxation makes pricing out new tours a relatively easy problem (pseudo polynomial). Using 2-loop elimination techniques, first proposed for the CARP in (Benavent *et al.*, 1992), massive cycling on service edges can be prevented. The consequence of allowing non-elementary tours is however that coefficients $\bar{x}_{er}$ of tours are not necessarily binary, but can be non-negative integers. With these additional tour variables in the set-covering formulation, the lower bound of the LP-relaxation is weakened.

However, the bounds obtained with the LP-relaxation of (14)–(17) sometimes outperform those of the one-index formulation. The approach is therefore attractive but uncomplete as Gómez-Cabrero *et al.* (2005) do not present a branching scheme which is in general needed to determine an optimal integer solution. The devising of such a branching scheme is one of the major contributions of this paper.

Another set covering-based solution approach was presented, discussed, and empirically analyzed by Letchford and Oukil (2009). The main focus of this paper is on the impact that elementary pricing has on lower bounds. The authors neither extend their set-covering formulation with valid cuts, nor devise a branching scheme to produce integer CARP solutions. The final conclusion that can be drawn from the paper is that elementary routes improve the lower bounds at the cost of an dramatic increase in computation times. Comparing the results of (Letchford and Oukil, 2009) and (Gómez-Cabrero *et al.*, 2005), the contribution of elementary pricing to the quality of the lower bounds is on average smaller than the impact of the cuts.

3. Dantzig-Wolfe Decomposition

Dantzig-Wolfe decomposition is one of the most successful techniques when it comes to solving vehicle and crew routing and scheduling problems (Desaulniers *et al.*, 2005; Lübbecke and Desrosiers, 2005). The advantages of solving the resulting integer master program follow from (i) a typically stronger lower bound, (ii) the elimination of symmetry in vehicles or crew members, and (iii) the possibility to handle non-linear cost structures for routes and schedules. In the CARP case the first two aspects apply.

In the following, we propose the decomposition of the two-index formulation (1)–(8) together with the valid cuts (13). As in other routing problems, we assume that the covering/partitioning constraints (2) are the coupling constraints. Additionally, the valid cuts (13) (or any subset of active cuts) are coupling constraints.

3.1. Column-Generation Formulation

First we analyze the domain $D = \{(x, y, p) : \text{fulfilling (3)–(8)}\}$ in order to describe the general structure of the pricing problem as well as the extreme points of D that correspond with the variables of the column-generation formulation a.k.a. extensive formulation. The domain D is separable by vehicle (index k) and can therefore be described as the cartesian product of domains $D^k = \{(x^k, y^k, p^k) : \text{that fulfill (3)–(8)}\}$ for $k \in K$. As vehicles are assumed identical in the CARP, all domains D^k are identical.

Let $X = conv(D^k)$ be the polyhedron given by the convex hull of integral points in D^k. X consists of all convex combinations of its extreme points plus all non-negative linear combination of its extreme rays (Schrijver, 2003). For the sake of simplicity, we argue with the well-studied directed case (Ahuja *et al.*, 1993) to describe what extreme points and rays of X are in the CARP case.

Modeling constrained directed shortest paths can be done on directed networks, where the nodes represent states (combinations of resource consumptions and nodes), and arcs connect those states resulting from feasible movements (see Irnich and Desaulniers, 2005). When modeling constrained shortest paths, the depot is typically split into a source and a sink node, one unit of flow is sent from source to sink, and flow conservation must hold in all nodes. Here, the set of extreme points consists of all efficient feasible routes. The attribute *efficient* means that the route does not deadhead along a cycle in G. Such a cycle corresponds with a ray of X. Extreme rays are the simple deadheading cycles in G. *Simple* means that all nodes of the cycle have degree 2. Thus, any route is composed of an efficient route plus a non-negative combination (possibly null) of simple cycles.

It is clear that optimal solutions to the CARP do not include routes r with deadheading cycles. Hence, only efficient routes are needed for the formulation of the master program. However, we will see later on that the inclusion of simple deadheading cycles of the form $C_e = (e, e)$ for edges $e \in E$ is helpful (see Section 3.4).

We use the identical notation for routes $r \in \Omega$, coefficients of route costs c_r, service $\bar{x}_{er}$, deadheading $\bar{y}_{er}$, and cuts d_{sr} as in Section 2.4. The integer master program (IMP) of the CARP reads as follows:

$$\min \quad \sum_{k \in K} c^\top \lambda^k \tag{18}$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{r \in \Omega} \bar{x}_{er} \lambda_r^k = 1 \quad \text{for all } e \in E_R \tag{19}$$

$$\sum_{k \in K} \sum_{r \in \Omega} d_{sr} \lambda_r^k \geq r_s \quad \text{for all } s \in \mathcal{S} \tag{20}$$

$$\sum_{r \in \Omega} \mathbf{1}^\top \lambda_r^k = 1 \quad \text{for all } k \in K \tag{21}$$

$$\lambda^k \geq \mathbf{0} \quad (\in \mathbb{R}^{|\Omega|}) \quad \text{for all } k \in K \tag{22}$$

$$x_e^k = \sum_{r \in \Omega} \bar{x}_{er} \lambda_r^k \quad \text{for all } e \in E_R,\ k \in K \tag{23}$$

$$x^k \in \{0,1\}^{|E_R|} \quad \text{for all } k \in K \tag{24}$$

$$y_e^k = \sum_{r \in \Omega} \bar{y}_{er} \lambda_r^k \quad \text{for all } e \in E,\ k \in K \tag{25}$$

$$y^k \in \mathbb{Z}_+^{|E|} \quad \text{for all } k \in K \tag{26}$$

The objective (18) minimizes over the costs of all tours. Equalities (19) ensure that every required edge is covered exactly once. The reformulated cuts are given by (20). Equalities (21) are convexity constraints and require each vehicle to perform a CARP tour. Constraints (23) and (25) couple the variables for service and deadheading with the tour variables and constraints (24) and (26) ensure the integrality of the solution.

The LP-relaxation of (18)–(26) is the master program (MP). As there is no need to keep the coupling constraints (23) and (25) when integrality is relaxed, MP reduces to (18)–(22). Column generation solves MP by iteratively reoptimizing a restricted master program (RMP) over a proper subset of variables (columns) and generating missing variables with negative reduced costs.

3.2. Pricing Problem and Relaxations

The task of the pricing problem is exactly the generation of one or several variables with negative reduced cost or proving that no such variable exists. Let dual prices $\pi = (\pi_e)_{e \in E_R}$ to the covering constraints (19), $\beta = (\beta_s)_{s \in \mathcal{S}}$ to the cuts (20), and $\mu = (\mu^k)_{k \in K}$ to the convexity constraints (21) be given. Omitting the index k of the vehicle, the pricing problem is

$$z_{PP} \quad = \quad \min \tilde{c}^{serv,\top} x + \tilde{c}^\top y - \mu \tag{27}$$

$$\text{s.t.} \quad x(\delta_R(S)) + y(\delta(S)) \geq 2x_f \quad \text{for all } S \subseteq V \setminus \{d\},\ f \in E_R(S) \tag{28}$$

$$x(\delta_R(i)) + y(\delta(i)) = 2p_i \quad \text{for all } i \in V \tag{29}$$

$$q^\top x \leq Q \tag{30}$$

$$p \in \mathbb{Z}_+^{|V|}, y \in \mathbb{Z}_+^{|E|} \tag{31}$$

$$x \in \{0,1\}^{|E_R|}, \tag{32}$$

where reduced costs for service and deadheading can be associated to the edges:

$$\tilde{c}_e^{serv} = c_e^{serv} - \pi_e \text{ for all } e \in E_R \qquad \text{and} \qquad \tilde{c}_e = c_e - \sum_{s \in \mathcal{S}} d_{es} \beta_s \text{ for all } e \in E. \tag{33}$$

It is known that the determination of a minimum reduced cost route $r \in \Omega$ for the CARP is an $\mathcal{NP}$-hard problem. Even if practically tractable for small and mid-sized instances, computation times can become long (Letchford and Oukil, 2009). In order to keep the computational effort small, several researchers proposed the use of non-elementary CARP routes. We follow this idea in our cut-first bap-second approach and briefly discuss the impact of a relaxed pricing problem.

In contrast to elementary routes, a non-elementary route r services at least one of the required edges $e \in E_R$ more than once, i.e., has coefficient $\bar{x}_{er} \geq 2$. The effect for the MP is the enlargement of the set Ω which typically degrades the quality of the lower bound. The advantage is, however, that pricing becomes an easy problem. The currently most efficient non-elementary pricing algorithm was recently presented by Letchford and Oukil (2009) and has worst-case complexity $\mathcal{O}\left(Q(|E|+|V|\log|V|)\right)$, while elementary pricing is $\mathcal{NP}$-hard in the strong sense. In fact, pricing elementary or non-elementary routes plays with the tradeoff between the hardness of pricing and the quality of the lower bound resulting in branch-and-bound trees that can significantly differ in size.

Each route r, either elementary or non-elementary, is given as a point $(\bar{x}_{er}, \bar{y}_{er}, \bar{p}_{vr})$ satisfying the constraints (27)–(31), but not necessarily the binary requirements (32) for x_{er}. The point represents a solution in which an edge e is traversed $\bar{x}_{er} + \bar{y}_{er}$ times. The corresponding multiple copies of these edges form a Eulerian graph. Since a Eulerian tour is generally not unique, the tour does not automatically imply a particular sequence in which edges are serviced. In the following, however, we assume that for each tour r an associated unique sequence s_r for the service is given. Such a sequence arises naturally when routes are determined as shortest paths in pricing (see Section 4.2). More precisely, we write $s_r = (e_1^r, e_2^r, \ldots, e_{p^r}^r)$ for the sequence in which the required edges $e_1^r, e_2^r, \ldots, e_{p^r}^r \in E_R$ are serviced by route r.

Whenever consecutively serviced edges are identical, i.e., $e_i^r = e_{i+1}^r$ for an index i, the route contains a so-called 2-loop. The simplest form of a 2-loop is the subroute (i, j, i) on a required edge $e = \{i, j\}$. Opposed to node routing, where the only 2-loops are subpaths (i, j, i), the CARP 2-loops may connect the endpoints of the required edge $e = \{i, j\}$ by any deadheading path between these endpoints. Thus, using the concept of tasks (on edges), in more detail described in (Irnich and Desaulniers, 2005), 2-loop free CARP routes are routes without task 1-cycles. Pricing 2-loop free routes can be done as efficiently as non-elementary pricing. We outline this approach in Section 4.2.

3.3. Aggregation

In order to eliminate the symmetry with respect to the vehicles, aggregation over $k \in K$ identical subproblems can be applied. The aggregated service, deadheading, and route variables are

$$x_e = \sum_{k \in K} x_e^k \quad \text{for } e \in E_R, \qquad y_e = \sum_{k \in K} y_e^k \quad \text{for } e \in E, \qquad \lambda_r = \sum_{k \in K} \lambda_r^k \quad \text{for } r \in \Omega.$$

This leads to the following aggregated integer master program (agg-IMP):

$$\min \quad \sum_{r \in \Omega} c_r \lambda_r \tag{34}$$

$$\text{s.t.} \quad \sum_{r \in \Omega} \bar{x}_{er} \lambda_r = 1 \quad \text{for all } e \in E_R \tag{35}$$

$$\sum_{r \in \Omega} d_{sr} \lambda_r \geq r_s \quad \text{for all } s \in \mathcal{S} \tag{36}$$

$$\mathbf{1}^\top \lambda = |K| \tag{37}$$

$$\lambda \geq \mathbf{0} \tag{38}$$

$$\lambda \in \mathbb{Z}^{|\Omega|}$$

A crucial point in our approach is that aggregation complicates the determination of feasible integer CARP solutions. We assume that a solution $\bar{\lambda}$ of the LP-relaxation of (34)–(38) is given. Clearly, if all variables λ are binary, an integer solution of the CARP is found. However, the development of a branching scheme to

force a solution to become integral is intricate for the CARP (see also Section 4.3). First, it is not possible to uniquely deduce the disaggregated values of $\bar{x}_e^k$ and $\bar{y}_e^k$ from $\bar{\lambda}$. Moreover, the values of the aggregated service variables are useless as $\bar{x}_e = 1$ holds. Finally, the aggregated deadheading variables $\bar{y}_e$ do not allow the reconstruction of tours, as already discussed for the one-index formulation in Section 2.3.

A second important aspect is that integrality of all variables y_e does not automatically force the tour variables λ to be binary. This implies that branching on the aggregated original variables is generally not sufficient to guarantee integrality. It is widely known that branching on the route variables λ_r has the disadvantages that it destroys the structure of the pricing problem and branches tend to be highly unbalanced (Villeneuve and Desaulniers, 2005). The effect is a typically untractable pricing problem together with a huge branch-and-bound tree. Instead integrality of agg-IMP must be controlled by additional constraints that are not included in the given formulation (34)–(38).

A first alternative is related to the branching rule that Ryan and Foster (1981) suggested for the LP-relaxation of a set-partitioning problem: in any fractional solution there exist two rows, say e and e', such that the fractional solution does not uniquely determine whether e and e' are covered by the same or by different columns. For the formulation agg-MP, it suffices to have

$$g_{ee'} = \sum_{r\in\Omega} (\bar{x}_{er}\bar{x}_{e'r})\lambda_r \in \{0,1\} \quad \text{for all } e, e' \in E_R.$$

The variable $g_{ee'}$ indicates whether the two required edges $e, e' \in E_R$ are served separately or by the same tour. These additional binary constraints ensure binary route variables λ. The condition $g_{ee'} = 0$ is equivalent to $x_e^k + x_{e'}^k \leq 1$ for all $k \in K$ in the two-index formulation, while $g_{ee'} = 1$ is equivalent to $x_e^k = x_{e'}^k$ for all $k \in K$. Even if these conditions can be expressed in the original variables, they destroy the structure of the pricing problem.

In our approach we use a second alternative to ensure integrality based on follower information. Compared to Ryan and Foster's rule it has the advantage that the structure of the pricing problem can be preserved. The follower conditions are given by

$$f_{ee'} = \sum_{r\in\Omega} f_{ee'r}\lambda_r \in \{0,1\} \quad \text{for all } e, e' \in E_R \tag{39}$$

where $f_{ee'r} = |\{1 \leq q < p^r : \{e, e'\} = \{e_q^r, e_{q+1}^r\}\}|$ counts how often the two edges e and e' are serviced in succession.

The follower information suffices to ensure integrality of the route variables λ as can be seen as follows: Let $G \subset E_R \times E_R$ be the binary relation representing the values $g_{ee'}$, i.e., $(e, e') \in G$ if and only if $g_{ee'} = 1$. Similarly, let F be the follower relation with $(e, e') \in F$ if and only if $f_{ee'} = 1$. Then G is the reflexive and transitive closure of F. Hence, binary values of $f_{ee'}$ imply binary values of $g_{ee'}$, and therewith binary route variables λ_r.

The algorithmic procedures that impose follower ($f_{ee'} = 1$) and non-follower ($f_{ee'} = 0$) conditions to the master and pricing problems are explained in Section 4.3.

3.4. Dual-Optimal Inequalities

The equation (33) for the reduced costs of deadheading along the edge $e \in E$ shows that for large values of dual prices β_s the reduced cost $\tilde{c}_e$ might become negative. Thus, an extreme ray corresponding to the cycle $C_e = (e, e)$ must be priced out.

Conversely, if an additional variable which represents that cycle C_e is already present in agg-MP, its reduced cost must be non-negative for any RMP solution. This implies that also the reduced costs $\tilde{c}_e$ are non-negative because $2\tilde{c}_e$ is the reduced cost of the cycle C_e.

The goal of this subsection is to briefly discuss the theoretical implications of adding variables for cycles C_e. Let $z_e \geq 0$ be the variable that represents the cycle $C_e = (e, e)$ corresponding to an extreme ray of the pricing polyhedron X. The addition of z_e to the LP-relaxation of agg-MP leads to the following

extended aggregated master program (eMP):

$$\min \quad \sum_{r\in\Omega} c_r\lambda_r + \sum_{e\in E}(2c_e)z_e \tag{40}$$

$$\text{s.t.} \quad \sum_{r\in\Omega} \bar{x}_{er}\lambda_r = 1 \quad \text{for all } e \in E_R \tag{41}$$

$$\sum_{r\in\Omega} \bar{y}_{sr}\lambda_r + \sum_{e\in E}(2d_{es})z_e \geq r_s \quad \text{for all } s \in \mathcal{S} \tag{42}$$

$$\mathbf{1}^\top\lambda = |K| \tag{43}$$

$$\lambda \geq \mathbf{0}, z \geq \mathbf{0} \tag{44}$$

Obviously, deadheading twice through e produces a cost of $2c_e$ in (40), it has no impact on covering (41), but can have non-zero coefficients $2d_{es}$ in the cuts (42). As C_e is an extreme ray, it has coefficient zero in the generalized convexity constraint (43).

Additional variables in eMP, the primal problem, correspond with inequalities in the dual problem. In fact, the presence of z_e in eMP gives a dual inequlity of the form

$$\sum_{s\in\mathcal{S}} d_{se}\beta_s \leq c_e \quad \text{for all } e \in E.$$

The concept of dual-optimal inequalities was first presented by Ben Amor *et al.* (2006). In our case, the positive impact of the dual cuts is twofold:

1. The reduced costs of deadheading edges are non-negative which provides algorithmic advantages in every pricing iteration as will be shown in Section 4.2.
2. The dual variables β_s are stabilized because their values are restricted by the dual-optimal inequalities. The result is a typically faster convergence of the column generation process (du Merle *et al.*, 1999; Ben Amor *et al.*, 2006).

4. Cut-First Branch-and-Price-Second Approach

The proposed cut-first bap-second approach has three key components, the cut generation procedure, the pricer, and the branching scheme. These components will be explained in the following.

4.1. Cutting

Before starting the bap phase cuts from the one-index formulation (9)–(12) are separated. We have implemented the following separation routines. To find violated odd cuts (11) the efficient separation algorithm of Letchford *et al.* (2004, 2008) is applied. The second separation routine identifies sets S which violates the capacity inequalities (10). We use the heuristic algorithm of Belenguer and Benavent (2003) and an exact MIP-based algorithm of Ahr (2004). In both cases, the minimum number of vehicles $K(S)$ is approximated by $\lceil q(E_R(S) \cup \delta_R(S))/Q \rceil$. Belenguer and Benavent (2003) presented the disjoint-path inequalities as a third class of valid cuts for the CARP. Since they did not describe a full separation routine, only candidate sets S which are generated in the previous step are tested. The idea behind this class of inequalities is that the vehicle also has to pick up load on the way from the depot to the set S. The cuts exclude combinations of tours where one of these vehicles is overloaded.

Only those cuts that are finally binding in the one-index formulation are active in the eMP. No additional cuts are added later in the column-generation procedure even if non-binding or other cuts might become violated in the branch-and-bound tree. Conversely, non-binding cuts are not eliminated from eMP.

4.2. Pricing

For pricing out non-elementary routes, Letchford and Oukil (2009) proposed labeling algorithms that work on the original CARP graph G and so exploit the sparsity of the network. They introduce both an exact and a heuristic pricer that consider labels with identical capacity consumption q in the sequence $q = 0, 1, 2, \ldots, Q$. Both algorithms have two types of path-extension steps (see Irnich and Desaulniers, 2005, for details):

1. An extension with service on a required edge $e \in E_R$ always creates new labels where the consumed capacity increases by $q_e > 0$. The reduced cost $\tilde{c}_e^{serv}$ is added to the cost component.
2. An extension along a deadheading edge $e \in E$ does not alter the capacity consumed and adds the value $\tilde{c}_e$ to the cost of the partial path. The dual inequalities of the eMP guarantee that $\tilde{c}_e \geq 0$ holds. All extensions over deadheading edges (for a given capacity consumption q) can be performed together using the Dijkstra algorithm.

As a result, the overall time complexity of the exact pricing routine is $\mathcal{O}\left(Q(|E| + |V| \log |V|)\right)$.

We adapted the presented pricing routines in order to eliminate 2-loops. This technique is straightforward following the ideas presented in (Houck *et al.*, 1980; Benavent *et al.*, 1992). The resulting worst-case time complexity is still $\mathcal{O}\left(Q(|E| + |V| \log |V|)\right)$.

4.3. Branching

Let $\bar{\lambda}$ be a fractional solution to eMP at a branch-and-bound node with associated values $\bar{x}$ and $\bar{y}$ (eqs. (23) and (25)). To obtain an integer solution a branching scheme has to be devised. Our hierarchical branching scheme consists of three levels of decisions:

1. branching on node degrees
2. branching on edge flows
3. branching on followers and non-followers

The idea behind this scheme is that decisions from the first two levels are more global decisions that typically have a stronger impact on the lower bounds. The decisions of the third level are more local, but they alone guarantee integrality (see Section 3.3).

First, if there exists a node $i \in V$ with node degree $\bar{d}_i = \bar{x}(\delta(i)) + \bar{y}(\delta(i))$ not even (either fractional or odd), the two branches $x(\delta(i)) + y(\delta(i)) \leq 2p$ and $x(\delta(i)) + y(\delta(i)) \geq 2p + 2$ are created with $p \in \mathbb{Z}_+$ defined by $2p < \bar{d}_i < 2p + 2$. Second, if for an edge $e \in E$ the edge flow $\bar{\phi}_e = \bar{x}_e + \bar{y}_e$ is fractional, the two branches $x_e + y_e \leq \lfloor \bar{\phi}_e \rfloor$ and $x_e + y_e \geq \lfloor \bar{\phi}_e \rfloor + 1$ are generated. Both types of branching decisions only have an impact on the master program, where a linear constraint must be added. This constraint has the same form as the cuts (42). Consequently, the equations (33) can still be used to compute the reduced cost of service and deadheading edges. Third, if for any two required edges e and e' the follower information $\bar{f}_{ee'}$ (see eq. (39)) is fractional, two branches with constraints $f_{ee'} = 0$ and $f_{ee'} = 1$ are induced. This means that all routes variables λ not respecting the constraint are removed from eMP. Moreover, no routes violating these decisions must be priced out. We guarantee compatible routes by modifying the underlying graph on which pricing is carried out. The network modifications that we describe in Section 4.3.2 does not destroy the structure of the pricing problem.

The specific variable to branch on is determined as follows. For branching on node degrees, we first compute for each node $i \in V$ the distance of $\bar{d}_{i^*}$ to the next even integer, i.e., $\min\{2p + 2 - \bar{d}_{i^*}, \bar{d}_{i^*} - 2p\}$ for an integer p with $2p \leq \bar{d}_{i^*} < 2p + 2$. We select the node i^* for which

$$\frac{\min\{2p + 2 - \bar{d}_{i^*}, \bar{d}_{i^*} - 2p\}}{\alpha + \beta 2p}$$

is maximal. We experimented with different values for α and β. For example, $\alpha = 1$ and $\beta = 0$ chooses the largest absolute distance of the node degree to the next even integer. In the final implementation we have chosen $\alpha = 6$ and $\beta = 1$ and ties are broken arbitrarily. The idea of this rule is that the distance of $\bar{d}_i$ to

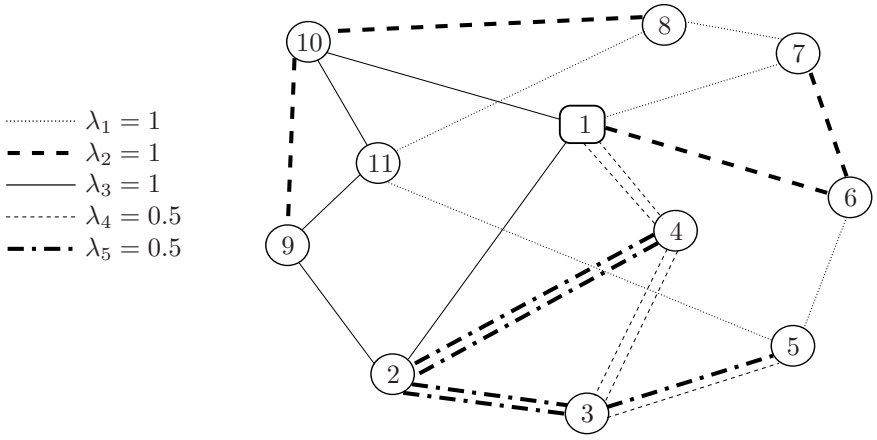

Figure 1: Fractional Solution

the next even integer is biased towards selecting nodes with a smaller node degree. For branching on edge flows, an edge e^* with fractional flow $\bar{y}_e$ closest to 0.5 is chosen.

In order to describe the rule for the third level, we partition the set E_R of required edges. The active branch-and-bound constraints induce the follower relation F and the non-follower relation N, i.e., $F = \{(e, e') : f_{ee'} \text{ is fixed to 1}\}$ and $N = \{(e, e') : f_{ee'} \text{ is fixed to 0}\}$. The reflexive and transitive closure of $F \cup N$ defines the partitioning $E_R = E_R^1 \cup E_R^2 \cup \ldots \cup E_R^q$. The partitions E_R^k characterize which required edges are directly or indirectly connected by active follower and non-follower constraints. If there exists no follower or non-follower relation for an edge $e \in E_R$, the partition consists of that edge alone.

For the selection of a variable $f_{e^*e^{*\prime}}$, we search for the pair $(e^*, e^{*\prime})$ with fractional value $\bar{f}_{e^*e^{*\prime}}$ closest to 0.5 inducing a partition of size less than or equal to 5. If e^* and $e^{*\prime}$ are already in the same partition, the addition of an associated constraint will not alter the given partitioning. Otherwise, the two partitions of e^* and $e^{*\prime}$, say E_R^k and E_R^ℓ are merged resulting in a single partition of size $|E_R^k| + |E_R^\ell|$. If no induced partition is of size less than or equal to 5, we only consider pairs with smallest induced partition and proceed as described before. The idea behind this rule is that small partitions are algorithmically attractive because we have to enumerate permutations of their edges to map branching decisions to the pricing network (see Section 4.3.2).

The analysis undertaken in Section 3.3 has shown that branching on followers alone guarantees binary master program variables λ. A crucial point in the proof is that routes were assumed being elementary. In fact, for relaxed pricing with 2-loop free routes the variables λ can still be fractional.

Example: Consider a CARP instance with nodes $\{1, \ldots, 11\}$ and 19 edges that are all required. The depot is located in node 1. The edge demands are all equal to 1 and the vehicle capacity is $Q = 5$. Four vehicles are needed to service these edge demands. In Figure 1, a solution of eMP in one of the branch-and-bound nodes is shown. This eMP solution consists of five (fractional) routes that are 2-loop free. Below, the corresponding node sequences ("=" indicates a service and "−" a deadheading) and service sequences s_r are shown. Additionally, the value of the corresponding route variable λ is given.

route r_1 is $(1 = 7 = 8 = 11 = 5 = 6 - 1)$	servicing $s_1 = (\{1,7\}, \{7,8\}, \{8,11\}, \{11,5\}, \{5,6\})$	$\bar{\lambda}_1 = 1$
route r_2 is $(1 = 6 = 7 - 8 = 10 = 9 - 2 - 1)$	servicing $s_2 = (\{1,6\}, \{6,7\}, \{8,10\}, \{10,9\})$	$\bar{\lambda}_2 = 1$
route r_3 is $(1 = 2 = 9 = 11 = 10 = 1)$	servicing $s_3 = (\{1,2\}, \{2,9\}, \{9,11\}, \{11,10\}, \{10,1\})$	$\bar{\lambda}_3 = 1$
route r_4 is $(1 = 4 = 3 = 5 - 3 = 4 = 1)$	servicing $s_4 = (\{1,4\}, \{4,3\}, \{3,5\}, \{3,4\}, \{4,1\})$	$\bar{\lambda}_4 = 0.5$
route r_5 is $(1 - 4 = 2 = 3 = 5 - 3 = 2 = 4 - 1)$	servicing $s_5 = (\{4,2\}, \{2,3\}, \{3,5\}, \{3,2\}, \{2,4\})$	$\bar{\lambda}_5 = 0.5$

The eMP uses routes r_1, r_2 and r_3 one time each, but the last two routes r_4 and r_5 are used only 0.5 times. The only edge serviced by two routes is $\{3,5\}$ (by route 4 and 5). Moreover, the edges $\{1,4\}, \{2,3\}, \{2,4\}$ and $\{3,4\}$ are serviced twice within the same route, either by route 4 or 5, i.e., $f_{\{1,4\},\{3,4\},4} = f_{\{3,4\},\{3,5\},4} = f_{\{2,4\},\{2,3\},5} = f_{\{2,3\},\{3,5\},5} = 2$ (see equation (39)).

Note that this convex combination of routes produces integer edge flows on all edges (see Figure 1). The implied follower information is

$$f_{\{1,4\},\{3,4\}} = f_{\{3,4\},\{3,5\}} = f_{\{3,5\},\{2,3\}} = f_{\{2,3\},\{2,4\}} = 1.$$

4.3.1. Integer Solutions from Follower Information

With a relaxed pricing, it can happen that all follower variables $f_{ee'}$ are binary but route variables $\bar{\lambda}$ are still fractional (see example above). In this case, nevertheless, an integer solution can implicitly be obtained from the fractional master program eMP. In fact, the binary follower information implies unique partitions and sequences of required edges that can be utilized to construct a solution.

We assume that the relation F is given by those pairs of required edges that are followers in the eMP, i.e., fulfill $\bar{f}_{ee'} = 1$. The reflexive and transitive closure $\bar{F}$ of the follower relation F defines exactly $|K|$ partitions of the required edges, i.e., $E_R = \bigcup_{k \in K} E_R^k$. A required edge $e \in E_R$ can be in follower relation to either no other edge, to a single edge, or to exactly two edges. In other words, the graph $(E_R, \bar{F})$ has exactly $|K|$ components consisting of paths (possibly with length 0). Hence, F implies for each partition E_R^k a sequence $s_k = (e_1^k, e_2^k, \ldots, e_{t_k}^k)$ of required edges. This sequence is unique except for reversal.

The final step is the determination of cost-minimal routes r_k that service the required edges exactly in the sequence s_r. This task consists of two types of decisions, the identification of deadheading paths connecting subsequent required edges and the fixing of directions in which required edges are serviced. A cost-minimal route can be computed as a shortest path in the auxiliary network depicted in Figure 2. We assume that for each pair of nodes a shortest deadheading path between these nodes is pre-computed such that c_{ij} is the shortest deadheading distance between the nodes i and j. Recall that c_e^{serv} is the cost for servicing a required edge e. The auxiliary network consists of two copies for each required edge $e_1, \ldots, e_t$ modeling the two possible directions when servicing. Starting and ending at nodes representing the depot d, dashed arcs model the deadheadings between the required edges. Thus, the length of a path in the auxiliary network is exactly the cost of the shortest route that services $e_1, \ldots, e_t$ in the given sequence s_r.

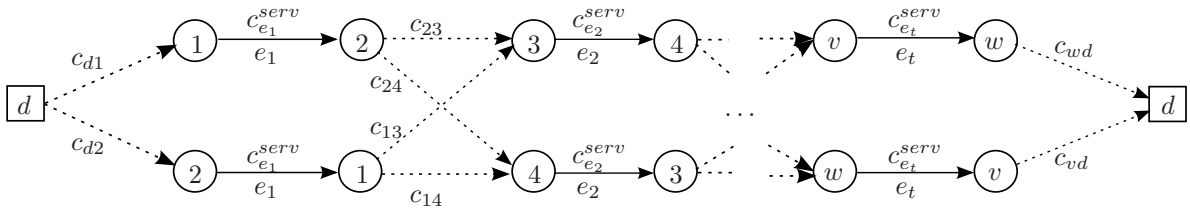

Figure 2: Auxiliary network

Example: In the previously given example, the reflexive and transitive closure $\bar{F}$ defined by the five routes separates the set of required edges into four partitions $E_R^1 \cup E_R^2 \cup E_R^3 \cup E_R^4$. The follower relation defined by routes 4 and 5 result in one partition E_R^4. This implies four sequences of required edges:

$$\begin{aligned} s_1 &= (\{1,7\},\{7,8\},\{8,11\},\{11,5\},\{5,6\}) \\ s_2 &= (\{1,6\},\{6,7\},\{8,10\},\{10,9\},) \\ s_3 &= (\{2,9\},\{9,11\},\{11,10\},\{10,1\}) \\ s_4' &= (\{1,4\},\{4,3\},\{3,5\},\{3,2\},\{2,4\}) \end{aligned}$$

The costs of the corresponding routes equal the costs of the fractional solution in the previous example.

4.3.2. Network Modifications

This section explains how follower and non-follower constraints can be handled in the pricing problem. The key property of our approach is that the active constraints can solely be implemented by network modifications. More precisely, only deletions and additions of required edges are performed on the original pricing network $G = (V, E)$. The basic structure of the pricing problem remains unchanged. We assume that active branch-and-bound constraints are given by the follower relation $F = \{(e, e') : f_{ee'} \text{ is fixed to } 1\}$ and the non-follower relation $N = \{(e, e') : f_{ee'} \text{ is fixed to } 0\}$.

For the sake of clarity, we first consider a single follower or non-follower constraint (F or N is $\{(e, e')\}$). Afterwards the general case with multiple active constraints will be described.

On the non-follower branch $f_{ee'} = 0$, servicing edge e immediately followed by edge e' is forbidden. Note that in the undirected network $G = (V, E)$ all constraints are symmetric so that e and e' can be interchanged and that the two services do not need to be directly connected but can be connected with any deadheading path. This constraint can be implemented using the concept of task-2-loop free paths as presented in (Irnich and Desaulniers, 2005; Irnich and Villeneuve, 2006). All required edges represent different tasks except for e and e' which represent the same task. Any task-2-loop free path ensures that the non-follower constraint $f_{ee'} = 0$ is respected.

On the follower branch $f_{ee'} = 1$, the service of edge e must immediately follow the service of edge e' (or vice versa). First, the two original required edges are deleted from the network (deadheading along e and e' remains possible). Second, four new edges are added to the network. They model the consecutive service to $e = \{i, j\}$ and $e' = \{k, l\}$. Since the direction of service is unknown for both edges, there are four possible paths:

- path $p_1 = (i, j)+$ deadheading from j to $k + (k, l)$
- path $p_2 = (i, j)+$ deadheading from j to $l + (l, k)$
- path $p_3 = (j, i)+$ deadheading from i to $k + (k, l)$
- path $p_4 = (j, i)+$ deadheading from i to $l + (l, k)$

The four additional edges $\{i, l\}$, $\{i, k\}$, $\{j, l\}$ and $\{j, k\}$ represent these paths. Consequently, they all have the same resulting demand $q_e + q_{e'}$ and the same associated task. The latter implies that no two of these four edges can be served consecutively. The costs of the new edges is calculated from summing up serviced costs $\tilde{c}_e^{serv} + \tilde{c}_{e'}^{serv}$ plus the costs for deadheading along the respective paths.

The modifications become even more intricate if several follower and non-follower conditions are active. In general the proceeding is outlined in Algorithm 1:

Algorithm 1: Network modification for follower and non-follower constraints

input : A network with costs $\tilde{c}_e^{serv}$ and $\tilde{c}_e$, active follower and non-follower relations F and N
output: A modified network

Compute partitions $E_R^1 \cup E_R^2 \cup \ldots \cup E_R^q$ of E_R induced by F and N ;
for *k=1,. . . ,q* **do**
  if $|E_R^k| > 1$ **then**
    Delete all required edges $e \in E_R^k$ from the network;
    Compute all feasible sequences $s = (e_1, e_2 \ldots, e_t)$ on all subsets of edges in E_R^k ;
    for *all sequences* $s = (e_1, e_2 \ldots, e_t)$ **do**
      for *the four pairs* (i, j) *of endpoints of* $e_1 = \{i_1, i_2\}$ *and* $e_t = \{j_1, j_2\}$ **do**
        Compute the minimum-cost path p servicing sequence s;
          p starts at node i and ends in node j;
        Add a required edge $\{i, j\}$ to the network;
          with demand $\sum_{p=1}^{t} q_{e_p}$, associated task is k;

The computation of a minimum-cost path p is similar to the shortest-path computation in the auxiliary network described in 4.3.1. We just elaborate the differences: First, costs c_e and c_e^{serv} have to be replaced by reduced costs $\tilde{c}_e$ and $\tilde{c}_e^{serv}$ defined by (33). Note that dual-optimal inequalities (see Section 3.4) guarantee that all reduced deadheading costs are non-negative. Therefore, the distances $\tilde{c}_{ij}$ of the shortest deadheading paths can be computed with the Dijkstra algorithm (between every pair of nodes i and j). Second, the path p starts in node i and ends in node j. Hence, the deadheading part from the depot d to the first required edge e_1 and from the last required edge e_t to the depot d is omitted in the auxiliary network. Moreover, the directions of e_1 and e_t are fixed in each iteration (third for-loop).

The following example makes the network modifications clear.

Example: We consider a graph G with eight nodes $\{a, \ldots, g\}$ and twelve edges (see Figure 3). We assume the following active (non)-follower decisions given by $F = \{(e_1, e_2), (e_3, e_4)\}$ and $N = \{(e_1, e_3)\}$ with $e_1 = \{a, b\}$, $e_2 = \{c, d\}$, $e_3 = \{e, f\}$ and $e_4 = \{g, h\}$. The edges e_1, e_2, e_3 and e_4 form a partition of E_R. The set of all possible subsequences s is

$$\{(e_1, e_2), (e_3, e_4), (e_1, e_2, e_3, e_4), (e_1, e_2, e_4, e_3), (e_2, e_1, e_4, e_3)\}$$

(For example, the sequences (e_2) and (e_2, e_3, e_4) are infeasible as they violates the follower condition $f_{e_1,e_2} = 1$.) For each possible subsequence and for each pair of start and end nodes, a shortest-path problem in the auxiliary network has to be solved. These twenty (five sequences and four pairs) new edges are added to the network depicted in Figure 4.

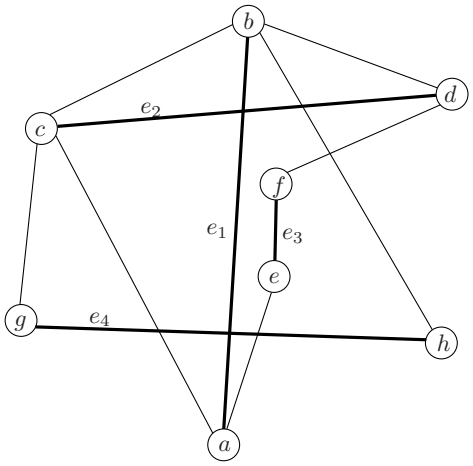

Figure 3: Original network

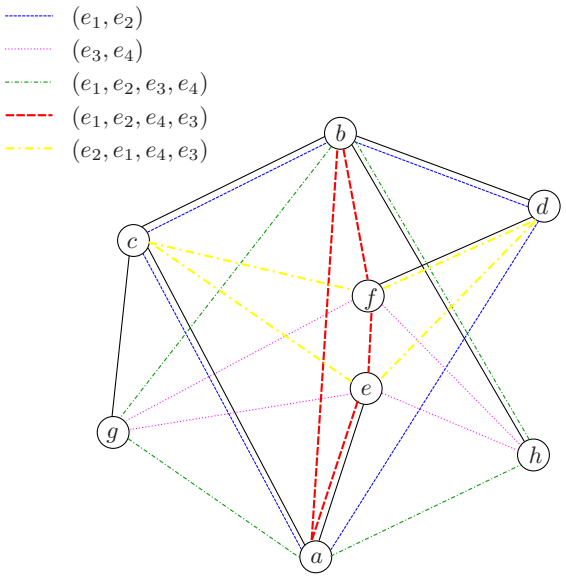

Figure 4: Modified network

A final remark is that parallel edges might result from feasible sequences s that have identical first and last edges. In this case, for identical demand $\sum q_{e_p}$, a single cost-minimal edge can be chosen while the other edges can be discarded.

5. Cycle Elimination

It is known for a long time that for routing problems cycle-elimination techniques can improve lower bounds of master programs (Houck *et al.*, 1980; Feillet *et al.*, 2004; Irnich and Villeneuve, 2006; Desaulniers *et al.*, 2008). For the CARP, Letchford and Oukil (2009) analyzed the impact of elementary CARP routes. Although their study shows that elementary routes can improve lower bounds, a comparison with the cutting-plane approaches of Benavent *et al.* (1992) and Ahr (2004) did not reveal the full potential of cycle elimination. The reason is that no odd-cut inequalities (11), capacity inequalities (10), and disjoint-path inequalities were present in the column-generation formulation. While Section 6.1 will quantify lower bound improvements due to cycle elimination, this section focuses on the interplay between cycle elimination and branching from a conceptual point of view.

In Section 3.2, task-2-loop elimination was introduced to improve the master program lower bound. Task-2-loop (i.e., task-1-cycle) elimination excludes the occurrence of two consecutive required edges labeled with the same task. Thus, by giving edges the same task the non-follower branching rule was implemented (see Section 4.3.2). The crucial point is here that branching requires task-k-loop exactly for $k = 2$. For

$k > 2$, the non-follower branching rule would be incorrect as also not direct following occurrences of the two tasks would be excluded.

In conclusion, branching only works with task-2-loop elimination while using task-k-loop for $k > 2$ helps to improve master program lower bounds. At first glance, both requirements ($k = 2$ and k as large as possible) seem incompatible in the presented bap approach. However, we can resolve this incompatibility by differentiating between the two corresponding classes of tasks:

- tasks $\mathcal{T}^E$ for modeling the elementary routes
- tasks $\mathcal{T}^B$ for branching

In essence, a shortest-path problem where paths are elementary w.r.t. $\mathcal{T}^E$ and task-2-loop free w.r.t. $\mathcal{T}^B$ must be solved. Relaxations to the elementarity w.r.t. $\mathcal{T}^E$ can again play with the tradeoff between hardness of the problem and strength of the relaxation. We outline the meaning and handling of the two task sets in the following.

For tasks $\mathcal{T}^E$, each required edge $e \in E_R$ forms an individual task so that the sets can be considered identical ($\mathcal{T}^E = E_R$). Branching does not modify this definition. A branch with a follower constraint, however, triggers a network modification: if a sequence of required edges is modeled by one or four new edges (see Section 4.3.2) such an edges gets the associated task sequence. The concept of task sequences associated to arcs (and nodes) has already been sketched in (Irnich and Desaulniers, 2005, p. 40).

For tasks $\mathcal{T}^B$, only active non-follower constraints have to be considered and cause the insertion of tasks into the set $\mathcal{T}^B$. More precisely, only those tasks k that are assigned to the new edges modeling a sequence through the same component E_R^k (see last step of Algorithm 1) have to be included in $\mathcal{T}^B$. Consequently, the task set $\mathcal{T}^B$ is empty at the root node of the bap tree. The k-cycle elimination method, as presented in (Irnich and Villeneuve, 2006), are therefore applicable when the root in bap is solved.

Our ongoing research is on combining $\mathcal{T}^B$-2-loop and $\mathcal{T}^E$-k-cycle elimination (and is going to be presented in a separate paper). It is obvious that not only k-cycle elimination but also alternative relaxations for $\mathcal{T}^E$-elementarity are promising, e.g., partial elementarity or NG-route relaxations (Desaulniers *et al.*, 2008; Baldacci *et al.*, 2009).

6. Computational Results

We tested our cut-first bap-second approach on the four standard benchmark sets for the CARP. The same benchmark sets (or a subset of instances) was also used to analyze the methods presented in the review (Section 2). The complete data with additional information can be downloaded from `http://www.uv.es/~belengue/carp/`. The two benchmark sets `kshs` and `gdb` contain 6 and 23 artificially generated instances. While the underlying graphs are sparse for the `gdb` set, some `kshs` instances are defined over a complete graph. Two other benchmark sets, `bccm` and `egl`, have 34 and 24 instances and are based on parts of the road network of Valencia and Cumbria, respectively.

The following results were all performed on a standard PC with an Intel Core Duo at 3 GHz with 4 GB of main memory. The algorithm was coded in C++ and the callable library of CPLEX 12.1 was used to iteratively solve LPs and MIPs (reoptimization and separation) in the cutting phase and to reoptimize the RMP.

For the computational analysis we use the following notation:

$\|V\|$	number of nodes
$\|E\|$	number of edges
$\|E_R\|$	number of required edges (not given for `kshs`, `gdb` and `bccm` as $E = E_R$)
K	number of vehicles
lb_{BB}	lower bound obtained by the cutting-plane algorithm of Belenguer and Benavent (2003)
lb_A	lower bound obtained by the cutting-plane algorithm of Ahr (2004)
lb_L	lower bound obtained in linear relaxation (root) by Longo *et al.* (2006)
lb_{LO}	lower bound obtained in linear relaxation (root, elem. routes) by Letchford and Oukil (2009)

lb^{root}_{own}	lower bound obtained in our linear relaxation (root)
lb^{tree}_{own}	lower bound obtained at termination of our bap algorithms (only given if time limit was reached)
ub_{BE}	upper bound obtained by heuristic of Brandão and Eglese (2008)
ub	upper bound presented in Belenguer and Benavent (2003)
ub_{best}	best known upper bound
comp. by	first paper where the currently best upper bound was computed P^{etal}= Pearn *et al.* (1987); H^{etal}= Hertz *et al.* (2000); La^{etal}= Lacomme *et al.* (2001)
opt	value of optimal solution
proved by lb	first paper where optimality is proved due to $\|ub - lb\| < 1$
proved by sol	first paper where optimality is proved by computing an integer solution B^{etal}= Benavent *et al.* (1992); BB= Belenguer and Benavent (2003); Lo^{etal}= Longo *et al.* (2006) BM= Baldacci and Maniezzo (2006); LO= Letchford and Oukil (2009); own= own solution
comp. time	in time seconds (for bap: 2h if time limit of two hours was reached)
B&B nodes	number of solved nodes in our branch-and-bound tree

6.1. Linear Relaxation Results

We begin with the presentation of the results on the linear relaxation of the master program eMP. These results are shown in the first three sections of the Tables 1-4. The cutting-plane algorithm in phase one solves the LP-relaxation of the one-index formulation (9)–(12). Here, we obtained at least the lower bounds lb_A as presented in (Ahr, 2004). Compared to (Belenguer and Benavent, 2003), however, our lower bounds are often weaker for the `bccm` instances. Actually, the description of the separation heuristics in (Belenguer and Benavent, 2003) leaves some details open and it seems that our code misses some important features. A more effective separation heuristic would certainly offer room for improvements.

Moreover, as our cutting-plane code is (up to now) not fine-tuned for maximum speed (using sophisticated separation heuristics, strategies to alleviate the tailing-off effect etc.) we decided to not present computation times. Thus, all presented computation times do not include the time spent in phase one. However, for the small and mid-sized instances (`kshs`, `gdb` and `bccm`) the observed cutting times are small compared to the overall bap computation times.

Table 1: Results for the `kshs` instances

instance	$\|V\|$	$\|E\|$	K	lb_{BB}	lb_A	lb^{root}_{own}	opt	proved by lb	proved by sol	comp. time	B&B nodes
kshs1	8	15	4	14661	14661	14661	14661	BB	Lo^{etal}	0.1	2
kshs2	10	15	4	9863	9863	9863	9863	BB	Lo^{etal}	0.1	2
kshs3	6	15	4	9320	9320	9320	9320	BB	Lo^{etal}	0.1	4
kshs4	8	15	4	11098	11098	11498	11498	LO	Lo^{etal}	0.1	1
kshs5	8	15	3	10957	10957	10957	10957	BB	Lo^{etal}	0.1	1
kshs6	9	15	3	10197	10197	10197	10197	BB	Lo^{etal}	0.1	1

For phase two, the following two column generation acceleration techniques were implemented: Before calling the exact pricer, we apply the heuristic of Letchford and Oukil (2009) (consider only tours with consecutive services) and a pricing heuristic where weakly dominant labels are ignored (see (Irnich and Desaulniers, 2005, p. 58)). Moreover, the partitioning constraints (41) in eMP are replaced by covering constraints (≥ 1) together with the constraint $\sum_{r\in\Omega}\sum_{e\in E_R}\bar{x}_{er}\lambda_r \leq |E_R|$ in order to stabilize the column generation process (du Merle *et al.*, 1999).

The focus of the following analysis is on lower bounds. The lower bounds lb^{root}_{own} obtained with our column-generation algorithm in the root node are always at least as good as the bounds lb_A obtained by Ahr (2004). This is a direct consequence of using all active odd-cut and capacity constraints to initialize the

Table 2: Results for gdb instances

instance	$\|V\|$	$\|E\|$	K	lb_{BB}	lb_A	lb_{own}^{root}	opt	proved by lb	proved by sol	comp. time	B&B nodes
gdb1	12	22	5	316	316	316	316	BB	Lo^{etal}	0.1	6
gdb2	12	26	6	339	339	339	339	BB	Lo^{etal}	0.4	6
gdb3	12	22	5	275	275	275	275	BB	Lo^{etal}	0.1	2
gdb4	11	19	4	287	287	287	287	BB	Lo^{etal}	0.1	1
gdb5	13	26	6	377	377	377	377	BB	Lo^{etal}	0.1	5
gdb6	12	22	5	298	298	298	298	BB	Lo^{etal}	0.1	6
gdb7	12	22	5	325	325	325	325	BB	Lo^{etal}	0.1	3
gdb8	27	46	10	344	344	348	348	BB	Lo^{etal}	0.7	9
gdb9	27	51	10	303	303	303	303	BB	Lo^{etal}	1.4	11
gdb10	12	25	4	275	275	275	275	BB	Lo^{etal}	0.2	11
gdb11	22	45	5	395	395	395	395	BB	Lo^{etal}	1.2	17
gdb12	13	23	7	450	450	451	458	-	Lo^{etal}	0.2	9
gdb13	10	28	6	536	536	536	536	BB	Lo^{etal}	0.2	5
gdb14	7	21	5	100	100	100	100	BB	Lo^{etal}	0.1	5
gdb15	7	21	4	58	58	58	58	BB	Lo^{etal}	0.1	10
gdb16	8	28	5	127	127	127	127	BB	Lo^{etal}	0.3	14
gdb17	8	28	5	91	91	91	91	BB	Lo^{etal}	0.2	17
gdb18	9	36	5	164	164	164	164	BB	Lo^{etal}	0.3	17
gdb19	8	11	3	55	55	55	55	BB	Lo^{etal}	0.1	1
gdb20	11	22	4	121	121	121	121	BB	Lo^{etal}	0.3	8
gdb21	11	33	6	156	156	156	156	BB	Lo^{etal}	0.3	13
gdb22	11	44	8	200	200	200	200	BB	Lo^{etal}	0.6	23
gdb23	11	55	10	233	233	233	233	BB	Lo^{etal}	0.8	26

eMP. Compared to the lower bounds lb_{BB} of Belenguer and Benavent (2003), the values lb_{own}^{root} are sometimes worse (for the `val` instances 2B, 5B, 6B and 7C) due to our less effective separation heuristic. Conversely, Ahr (2004) uses an exact separation procedure for capacity cuts which can lead to $lb_{BB} < lb_A$. It is obvious that the extension of our code with a more effective separation heuristic for disjoint-path inequalities in the cutting phase would result in $lb_{own}^{root} \geq \max\{lb_{BB}, lb_A\}$.

With the same set of cuts, the LP-relaxation of the master problem is a stronger formulation than the one-index formulation. Hence, for several instances better lower bounds were obtained by column generation, i.e., $lb_{own}^{root} \gneqq \max\{lb_{BB}, lb_A\}$. This is the case for one `kshs` instance (`kshs4`), two `gdb` instances (`gdb8` and `gdb12`), nine `bccm` instances, and all `egl` instances.

Longo *et al.* (2006) presented detailed lower bound results only for `bccm` and `egl` instances. There, lower bounds obtained in the root-node of the CVRP branch-and-price-and-cut algorithm are generally better than our bounds (7 times for `bccm` and always for `egl`). For the instances `1C` and `3C`, however, lb_{own}^{root} exceeds lb_L.

The results for the `egl` test set are presented in Table 4. Throughout all instances, the lower bounds lb_{own}^{root} obtained in our root node improve the lower bounds lb_A and lb_{BB} obtained with the cutting-plane algorithms, while the lower bounds lb_L obtained by Longo *et al.* (2006) are consistently better.

Some concluding remarks on the comparison with the Longo *et al.* (2006) can be given: Our computation times for root nodes are significantly smaller than those for the CVRP, sometimes by more than factor 100 (for example `bccm` instance `1C`). Since the branch-and-price-and-cut algorithm of Longo *et al.* (2006) is based on the algorithm by Fukasawa *et al.* (2006), CVRP pricing problems are solved with (node-)k-cycle elimination for $k \in \{2, 3, 4\}$. This partially explains their better lower bounds at the cost of a more complex and time-consuming pricing.

Table 3: Results for the `bccm` instances

instance	$\|V\|$	$\|E\|$	K	lb_{BB}	lb_A	lb_L	lb^{root}_{own}	lb^{tree}_{own}	ub_{best}	comp. by	opt	proved by lb	proved by sol	comp. time	B&B nodes
1A	24	39	2	247	247	247	247	-	247	H^{etal}	247	B^{etal}	own	15.4	29
1B	24	39	3	247	247	247	247	-	247	H^{etal}	247	B^{etal}	own	313.2	657
1C	24	39	8	309	309	312	314	-	319	H^{etal}	319		Lo^{etal}	133.2	1006
2A	24	34	2	298	298	298	298	-	298	H^{etal}	298	BB	own	7.4	9
2B	24	34	3	330	328	329	329	-	330	La^{etal}	330	BB	Lo^{etal}	297.9	378
2C	24	34	8	526	526	528	528	-	528	H^{etal}	528	Lo^{etal}	Lo^{etal}	0.2	1
3A	24	35	2	105	105	105	105	-	105	H^{etal}	105	BB	own	5.3	17
3B	24	35	3	111	111	111	111	-	111	H^{etal}	111	BB	own	1.6	10
3C	24	35	7	161	159	161	162	-	162	H^{etal}	162		Lo^{etal}	0.8	13
4A	41	69	3	522	522	522	522	-	522	H^{etal}	522	BB	own	168.4	41
4B	41	69	4	534	534	534	534	-	534	H^{etal}	534	BB	own	49.4	32
4C	41	69	5	550	550	550	550	-	550	La^{etal}	550	BB	own	22.5	21
4D	41	69	9	644	642	648	647	-	652	La^{etal}	**650**		**own**	589.7	758
5A	34	65	3	566	566	566	566	-	566	H^{etal}	566	BB	own	60.1	43
5B	34	65	4	589	586	588	587	-	589	H^{etal}	589	BB	own	60.5	69
5C	34	65	5	612	610	613	613	616	617	H^{etal}	617			*2h*	3951
5D	34	65	9	714	714	716	715	-	724	La^{etal}	**718**		**own**	23.8	73
6A	31	50	3	330	330	330	330	-	330	H^{etal}	330	B^{etal}	own	18.3	21
6B	31	50	4	338	336	337	336	337	340	La^{etal}	340	BB	BM	*2h*	4250
6C	31	50	10	418	414	420	419	-	424	H^{etal}	424		BM	1323.5	2884
7A	40	66	3	382	382	382	382	382	382	H^{etal}	382	B^{etal}	own	*2h*	2049
7B	40	66	4	386	386	386	386	-	386	H^{etal}	386	BB	own	748.3	665
7C	40	66	9	436	430	436	432	434	437	H^{etal}	437		Lo^{etal}	*2h*	7690
8A	30	63	3	522	522	522	522	-	522	H^{etal}	522	B^{etal}		32.8	36
8B	30	63	4	531	531	531	531	-	531	H^{etal}	531	B^{etal}	own	16.7	32
8C	30	63	9	653	645	654	653	-	663	La^{etal}	**657**		**own**	54.4	313
9A	50	92	3	450	450	450	450	-	450	H^{etal}	450	B^{etal}		558.3	47
9B	50	92	4	453	453	453	453	-	453	H^{etal}	453	B^{etal}		215.3	63
9C	50	92	5	459	459	459	459	-	459	H^{etal}	459	B^{etal}	own	94.2	56
9D	50	92	10	509	505	512	510	515	518	La^{etal}	**515***			*2h*	2493
10A	50	97	3	637	637	637	637	-	637	H^{etal}	637	B^{etal}	own	798.5	83
10B	50	97	4	645	645	645	645	-	645	H^{etal}	645	B^{etal}	own	358.5	68
10C	50	97	5	655	655	655	655	-	655	La^{etal}	655	BB	own	211.6	54
10D	50	97	10	732	731	734	734	-	739	La^{etal}	**734**		**own**	56.6	42

Table 4: Results for the `egl` instances

instance	$\|V\|$	$\|E\|/\|E_R\|$	K	lb_{BB}	lb_A	lb_L	lb_{LO}	lb^{root}_{own}	lb^{tree}_{own}	ub_{BE} or *opt*	comp. time	B&B nodes
egl-e1-A	77	98/51	5	3515	3516	3548	3425	3538	–	**3548**	109	30
egl-e1-B	77	98/51	7	4436	4436	4468	4291	4463	–	**4498**	6088	1874
egl-e1-C	77	98/51	10	5453	5481	5542	5472	5509	5542	5595	*2h*	2852
egl-e2-A	77	98/72	7	4994	4963	5011	4832	4988	–	**5018**	5033	1114
egl-e2-B	77	98/72	10	6249	6271	6280	6105	6273	6296	6317	*2h*	2201
egl-e2-C	77	98/72	14	8114	8155	8234	8187	8200	8238	8335	*2h*	3944
egl-e3-A	77	98/87	8	5869	5866	5898	5706	5884	–	**5898**	241	45
egl-e3-B	77	98/87	12	7646	7649	7697	7541	7690	7729	7777	*2h*	2094
egl-e3-C	77	98/87	17	10019	10119	10163	10086	10144	10184	10305	*2h*	3934
egl-e4-A	77	98/98	9	6372	6378	6395	6233	6389	6407	6456	*2h*	938
egl-e4-B	77	98/98	14	8809	8839	8884	8678	8854	8889	9000	*2h*	2728
egl-e4-C	77	98/98	19	11276	11376	11427	11416	11410	11455	11601	*2h*	4116
egl-s1-A	140	190/75	7	4992	4975	5014	4985	5011	5017	**5018**	*2h*	214
egl-s1-B	140	190/75	10	6201	6180	6379	6284	6370	6387	**6388**	*2h*	486
egl-s1-C	140	190/75	14	8310	8286	8480	8423	8412	8438	8518	*2h*	780
egl-s2-A	140	190/147	14	9780	9718	9824	9667	9792	9804	9956	*2h*	356
egl-s2-B	140	190/147	20	12286	12836	12968	12801	12943	12965	13165	*2h*	1589
egl-s2-C	140	190/147	27	16221	16216	16353	16262	16287	16319	16524	*2h*	2772
egl-s3-A	140	190/159	15	10025	9991	10143	9925	10129	10141	10260	*2h*	705
egl-s3-B	140	190/159	22	13554	13520	13616	13388	13589	13616	13807	*2h*	1264
egl-s3-C	140	190/159	29	16969	16958	17100	17014	17036	17074	17234	*2h*	2776
egl-s4-A	140	190/190	19	12027	12007	12143	11905	12119	12144	12341	*2h*	570
egl-s4-B	140	190/190	27	15933	15897	16093	15891	16067	16103	16462	*2h*	1560
egl-s4-C	140	190/190	35	20179	20176	20375	20197	20329	20365	20591	*2h*	2756

Cycle Elimination Result

Recall from Section 5 that we can provide results for pricing with k-loop elimination only for the linear relaxation eMP due to the incompatibility of the branching scheme with k-loop elimination for $k \geq 3$. Table 5 summarizes the impact of cycle elimination on the root node lower bound and the root node computation time. We present results only for the `egl` instances because these are the only ones where cycle elimination had a significant impact. We suspect that the presence of non-required edges conveys the appearance of cycles.

As could be expected, the results exactly reflect the trade-off between lower bound improvement and hardness of solving the respective linear relaxation. For $k = 5$-loop elimination, solving eMP become so time consuming (seven times more than 10 hours) that we stopped the computation prematurely. On average, the increase of the lower bound is 11 going from $k = 2$ to $k = 3$, and 6 from $k = 3$ to $k = 4$ (we omit results the step from $k = 4$ to $k = 5$ due to the missing entries). While for some instances the increase is marginal, it can become substantial (e.g. for `e2-C` an overall increase of 26+37+6=69). On the downside, average computation times increase also by factor 5.6 from $k = 2$ to $k = 3$, and by factor 5.5 from $k = 3$ to $k = 4$.

For instance `e1-A`, the 5-loop elimination entirely closes the integrality gap (remaining gap = 0). In seven other cases, 4-loop or 5-loop elimination gives a stronger root node relaxation than the CVRP root node relaxation computed in (Longo *et al.*, 2006) (`e1-B`, `e2-B`, `e2-C`, `e3-B`, `e3-C`, `e4-C`, and `s4-C`; see also Table 4). Interestingly, these are instances with relatively small computation times.

Concluding, the computational results indicate that the use of cycle-free routes is one of the key devices to improve lower bounds. In addition to the study of Letchford and Oukil (2009) it has become clear now that loop-elimination is still beneficial when cutting planes are already added to the eMP.

6.2. Branch-and-Price and Integer Solution Results

The final branch-and-bound component that has to be specified is the node selection rule. In order to increase the overall lower bound as fast as possible, nodes are processed according to the best-first search

Table 5: Cycle Elimination for the `egl` instances

instance	*ub* or *opt*	k = 2-loop elimination		k = 3-loop elimination		k = 4-loop elimination		k = 5-loop elimination		remain. gap
		lb^{root}	comp. time	lb^{root}	comp. time	lb^{root}	comp. time	lb^{root}	comp. time	
egl-e1-A	3548	3538	14	3540 (+2)	30	3540 (+0)	241	3548 (+8)	5786	0
egl-e1-B	4498	4463	11	4465 (+2)	101	4467 (+2)	319	4470 (+3)	3784	28
egl-e1-C	5595	5509	8	5522 (+13)	18	5525 (+3)	149	5528 (+3)	2030	67
egl-e2-A	5018	4988	26	4991 (+3)	120	4995 (+4)	1239	4995 (+0)	39754	23
egl-e2-B	6317	6273	14	6280 (+7)	83	6283 (+3)	559	6283 (+0)	12827	34
egl-e2-C	8335	8200	7	8226 (+26)	18	8263 (+37)	54	8269 (+6)	2151	66
egl-e3-A	5898	5884	43	5886 (+2)	650	5886 (+0)	1886			12
egl-e3-B	7777	7690	18	7703 (+13)	87	7707 (+4)	368	7714 (+7)	7499	63
egl-e3-C	10305	10144	8	10176 (+32)	24	10182 (+6)	65	10183 (+1)	2612	122
egl-e4-A	6456	6389	38	6390 (+1)	358	6390 (+0)	6575			66
egl-e4-B	9000	8854	17	8869 (+15)	64	8871 (+2)	259	8877 (+6)	3453	123
egl-e4-C	11601	11410	12	11436 (+26)	46	11463 (+27)	130	11465 (+2)	2418	136
egl-s1-A	5018	5011	263	5011 (+0)	1099	5013 (+2)	3708			5
egl-s1-B	6388	6370	113	6372 (+2)	364	6376 (+4)	2015	6377 (+1)	30419	11
egl-s1-C	8518	8412	33	8449 (+37)	50	8465 (+16)	94	8474 (+9)	888	44
egl-s2-A	9956	9792	93	9797 (+5)	436	9798 (+1)	3261			158
egl-s2-B	13165	12943	53	12950 (+7)	222	12957 (+7)	869	12964 (+7)	11261	201
egl-s2-C	16524	16287	32	16309 (+22)	89	16315 (+6)	252	16320 (+5)	5733	204
egl-s3-A	10260	10129	108	10132 (+3)	1186	10133 (+1)	9279			127
egl-s3-B	13807	13589	56	13594 (+5)	339	13597 (+3)	1209	13600 (+3)	24120	207
egl-s3-C	17234	17036	34	17069 (+33)	83	17072 (+3)	387	17076 (+4)	3420	158
egl-s4-A	12341	12119	117	12121 (+2)	830	12122 (+1)	3264			219
egl-s4-B	16462	16067	69	16071 (+4)	447	16073 (+2)	1553			389
egl-s4-C	20591	20329	59	20353 (+24)	214	20367 (+14)	378	20376 (+9)	2961	215

strategy. In case of a tie, the last node added is selected first. We found that this strategy is fundamental for finding integer solutions soon because there often exist large subtrees having nodes with identical lower bounds (some kind of plateaus). In these cases, the rule implies that the subtree is processed in a depth-first manner. Another crucial point for the effectiveness of the branching scheme is that the follower son node is always selected before its non-follower brother node (two unprocessed son nodes have identical lower bounds inherited from their father).

Results of the branch-and-price (bap) algorithm can be found in the rightmost section of the Tables 1-4 and columns headed lb_{own}^{tree}. All `kshs` and `gdb` instances were solved to proved optimality often in less than one second. We did *not* provide any upper bounds to help bap to terminate early. Instead, an optimal CARP solution was computed for these instances and the lower bound had to be raised to close the integrality gap (gap < 1).

For the `bccm` benchmark set, results are shown in Table 3. The information about optimal solutions differs from that given in (Letchford and Oukil, 2009) because results from a working paper by Ghiani, Laganá, Laporte, and Musmanno (from 2007) are unreliable. This working paper has been withdrawn (Laporte, 2011).

Except for the `bccm` instances `5C`, `6B`, and `7C`, we can either show that the ub_{best} is at the same time a lower bound or find an integer solution and prove its optimality. For instance `9D`, we have found the optimal solution with value 515 during experiments with other branching schemes. Hence, for `7A` and `9D` the final setup was only able to prove optimality due to the tree lower bound. For five instances (`4D`, `5D`, `8C`, `9D`, and `10D`), the values printed in bold indicate that the optimal solutions are found for the first time. This means the solution of all `bccm` benchmark problems that were open at the time of writing. Overall, we are able to find optimal integer solutions in 30 out of 35 cases. In contrast, the node-routing approach of Longo *et al.* (2006) and Baldacci and Maniezzo (2006), determine and prove optimal solutions in five and ten cases, respectively.

Results for the `egl` test set are presented in Table 4. With the given setup, four instances (`e1-A`, `e1-B`, `e2-A`, and `e3-A`) are solved to proved optimality in 2 hours. These instances and `s1-A` were already solved by Longo *et al.* (2006) and Baldacci and Maniezzo (2006). For `s1-A` and `s1-B` we are able to close the gap when we allow a little bit more time. While an optimal solution is easy to compute for `s1-A`, we were unable to determine one for `s1-B` even with an extended computation time of 24 hours. Concluding, we prove optimality of `s1-B` by lower bound using the upper bound presented in (Brandão and Eglese, 2008).

The lower bound lb_{own}^{tree} resulting from the partial solution of the branch-and-bound node improves the best known lower bounds lb_L presented by Longo *et al.* (2006) for nine instances.

7. Conclusions

In this paper we proposed a cut-first branch-and-price-second algorithm for the CARP. The strength of the new column-generation formulation results from strong lower bounds, symmetry elimination, efficient pricing, and an effective branching scheme. Strong lower bounds are obtained by the combination of cuts from the one-index formulation and the Dantzig-Wolfe reformulation inducing a set-partitioning type master program. This aggregated master program avoids vehicle-specific variables and therewith eliminates symmetry. Even so, the reconstruction of individual vehicle routes is possible. The generation of new routes can be done efficiently because all pricing computations can be performed on the original sparse underlying street network. A second fundamental finding is that negative reduced costs on deadheading edges can be prevented by adding dual-optimal inequalities to the extensive formulation. Non-negative reduced costs are algorithmically advantageous as parts of the pricing can now be carried out using Dijkstra's algorithm without the need to prevent cycling. Finally, the new branching scheme is the first one that ensures integral CARP solutions, while the structure of the pricing problem remains unchanged. Contrary to the node-routing case, the integrality of the aggregated variables of a original (=compact) formulation generally do not imply integer master program variables. The key device to finally obtain integer routes is branching on followers of required edges, where in one branch two required edges have to be serviced consecutively, and in the other branch subsequent service is forbidden. While branching on follower constraints is common

in node routing, the novelty of our approach is the handling of follower constraints referring to edges that might not be directly connected.

Computational experiments show that the proposed cut-first branch-and-price-second algorithm gives considerable results in all four benchmark sets. Several earlier exact approaches proved optimality of known heuristic solutions by matching lower and upper bounds, but were not able to deliver optimal CARP routes. Our branching scheme however enables us to compute feasible integer solutions and optimal ones in many cases. As a result, all open benchmark instances of Belenguer and Benavent (1998b) are solved now. For the (Eglese and Li, 1992) benchmark set, optimality of one more instance was shown and many lower bounds were improved significantly.

We see the following possible avenues for future research: To further strengthen the initial lower bound, a more efficient separation routine for disjoint-path inequalities could be implemented. These and other cuts might also be separated in the branch-and-bound tree and not only once at the beginning. Another topic, as already suggested by Letchford and Oukil (2009), are approaches that replace the weaker 2-loop free pricing with stronger relaxations. Since columns in the master program often contain 3-loops and longer loops, k-loop elimination for $k \geq 3$ would probably improve the lower bounds. We pointed out that this is a non-trivial task due to the incompatibility between the suggested branching rule on (non-)followers and k-loop elimination for $k \geq 3$. Various other relaxations of the elementary pricing problem that eliminate these loops should be analyzed (Irnich and Villeneuve, 2006; Desaulniers *et al.*, 2008; Baldacci *et al.*, 2009) for which we expect that pricing times remain acceptable as pricing on the original sparse graph is possible.

References

Ahr, D. (2004). *Contributions to Multiple Postmen Problems.* Ph.D. thesis, Ruprecht-Karls-Universität Heidelberg.

Ahuja, R., Magnanti, T., and Orlin, J. (1993). *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, New Jersey.

Baldacci, R. and Maniezzo, V. (2006). Exact methods based on node-routing formulations for undirected arc-routing problems. *Networks*, **47**(1), 52–60.

Baldacci, R., Mingozzi, A., and Roberti, R. (2009). Solving the vehicle routing problem with time windows using new state space relaxation and pricing strategies. Presented at AIRO 2008, EURO 2009, ISMP 2009, and AIRO 2009.

Baldacci, R., Bartolini, E., Mingozzi, A., and Roberti, R. (2010). An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science*, **7**, 229–268.

Belenguer, J. and Benavent, E. (1998a). The capacitated arc routing problem: Valid inequalities and facets. *Computational Optimization and Applications*, **10**(2), 165–187.

Belenguer, J. and Benavent, E. (2003). A cutting plane algorithm for the capacitated arc routing problem. *Operations Research*, **30**(5), 705–728.

Belenguer, J. M. and Benavent, E. (1998b). The capacitated arc routing problem: Valid inequalities and facets. *Computational Optimization and Applications*, **10**, 165–187.

Ben Amor, H., Desrosiers, J., and de Carvalho, J. (2006). Dual-optimal inequalities for stabilized column generation. *Operations Research*, **54**(3), 454–463.

Benavent, E., Campos, V., Corberán, A., and Mota, E. (1992). The capacitated arc routing problem: lower bounds. *Networks*, **22**, 669–669.

Brandão, J. and Eglese, R. (2008). A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research*, **35**(4), 1112–1126.

Corberán, A. and Prins, C. (2010). Recent results on arc routing problems: An annotated bibliography. *Networks*, **56**(1).

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation.* Springer, New York, NY.

Desaulniers, G., Lessard, F., and Hadjar, A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, **42**(3), 387–404.

Dror, M., editor (2000). *Arc Routing: Theory, Solutions and Applications.* Kluwer, Boston.

du Merle, O., Villeneuve, D., Desrosiers, J., and Hansen, P. (1999). Stabilized column generation. *Discrete Mathematics*, **194**, 229–237.

Eglese, R. and Li, L. (1992). Efficient routeing for winter gritting. *Journal of the Operational Research Society*, **43**(11), 1031–1034.

Feillet, D., Dejax, P., Gendreau, M., and Guéguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, **44**(3), 216–229.

Fukasawa, R., Longo, H., Lysgaard, J., Poggi de Aragão, M., Reis, M., Uchoa, E., and Werneck, R. F. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming, Series A*, **106**(3), 491–511.

Golden, B. and Wong, R. (1981). Capacitated arc routing problems. *Networks*, **11**, 305–315.

Gómez-Cabrero, D., Belenguer, J., and Benavent, E. (2005). Cutting plane and column generation for the capacitated arc routing problem. Presented at ORP3, Valencia.

Hertz, A., Laporte, G., and Mittaz, M. (2000). A tabu search heuristic for the capacitated arc routing problem. *Operations Research*, **48**(1), 129–135.

Houck, D., Picard, J., Queyranne, M., and Vemuganti, R. (1980). The travelling salesman problem as a constrained shortest path problem: Theory and computational experience. *Opsearch*, **17**, 93–109.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

Irnich, S. and Villeneuve, D. (2006). The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, **18**(3), 391–406.

Lacomme, P., Prins, C., and Ramdane-Chérif, W. (2001). A genetic algorithm for the capacitated arc routing problem and its extensions. In E. J. Boers, S. Cagnoni, J. Gottlieb, E. Hart, P. L. Lanzi, G. Raidl, R. E. Smith, and H. Tijink, editors, *Applications of Evolutionary Computing. EvoWorkshops2001: EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM. Proceedings*, volume 2037 of *LNCS*, pages 473–483, Como, Italy. Springer-Verlag.

Laporte, G. (2011). Personal Communication.

Lübbecke, M. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, **53**(6), 1007–1023.

Letchford, A., Reinelt, G., and Theis, D. (2004). A faster exact separation algorithm for blossom inequalities. In G. Nemhauser and D. Bienstock, editors, *Integer Programming and Combinatorial Optimization*, volume 3064, chapter 10. Springer, Berlin.

Letchford, A., Reinelt, G., and Theis, D. (2008). Odd minimum cut-sets and b-matchings revisited. *SIAM Journal on Discrete Mathematics*, **22**(4), 1480–1487.

Letchford, A. N. (1997). *Polyhedral Results for Some Constrained Arc-Routing Problems.* Ph.D. thesis, Lancaster University, Lancaster, UK.

Letchford, A. N. and Oukil, A. (2009). Exploiting sparsity in pricing routines for the capacitated arc routing problem. *Computers & Operations Research*, **36**(7), 2320–2327.

Longo, H., de Aragão, M., and Uchoa, E. (2006). Solving capacitated arc routing problems using a transformation to the CVRP. *Computers & Operations Research*, **33**(6), 1823–1837.

Lysgaard, J., Letchford, A., and Eglese, R. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, **100**(2), 423–445.

Pearn, W. L., Assad, A., and Golden, B. L. (1987). Transforming arc routing into node routing problems. *Computers & Operations Research*, **14**(4), 285–288.

Ryan, D. and Foster, B. (1981). An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, chapter 17, pages 269–280. Elsevier, North-Holland.

Schrijver, A. (2003). *Combinatorial Optimization. Polyhedra and Efficiency.* Number 24 in Algorithms and Combinatorics. Springer, Berlin, Heidelberg, New York.

Villeneuve, D. and Desaulniers, G. (2005). The shortest path problem with forbidden paths. *European Journal of Operational Research*, **165**(1), 97–107.

Appendix

Optimal solutions for the BCCM/Valencia instances (Benavent *et al.*, 1992):

4D z = 650

veh 1	1–2–3=4=5=6=12=11=17=16=15=10–9–3–2–1
veh 2	1–2–3–9=10=4–5=11=16–15=14=13=7=8–9–3–2–1
veh 3	1–2–3–9–14=24=25–31=35=36=32=26=19–16–15–10–9–3–2–1
veh 4	1=7–13=23=24=30=29=23–14=9–3–2–1
veh 5	1–2–3–9–10=11–17=18=22=28=27=21=22–21=20–19–16–15–10–9–3–2–1
veh 6	1–2–3–9–10–15=25=26=27=32=31=30–29–23–14–9–3–2–1
veh 7	1–2–3–9–10–15–25=31–35=34=38=39=36=37=33=27–20=17–11–10–9–3–2–1
veh 8	1–2–3–9–10–15–16=19=20=27–33–37=41=40=36–40=39=35–34=29–23=14–9–3–2–1
veh 9	1–2=8=9=3=2=1

5D z=718

veh 1	1=2=13=19=12=18=6–1
veh 2	1=6=7=1
veh 3	1–2=3=4=11=17–11=5=4=10=9=3–2–1
veh 4	1–2=8=9=15=14=8–14=13=12=6–1
veh 5	1–2–3=10=11=24=17=16=15–14–13=7–1
veh 6	1–2–3–10=16=23=24=34–28=23=22=21=20=14–13–12–6–1
veh 7	1–6–12–13–14–15=21=27=32=33=34=28=22=15–14–13–12–6–1
veh 8	1–6–18=19=20=26=31=30=29=18–6–1
veh 9	1–6–12–19=25=26=27=28=33–32=31–30=25=29–18–6–1

8C z = 657

veh 1	1=2=3=10=9=8=1
veh 2	1=5=8=7=6=4=5–1
veh 3	1=9–10=19=18–19=15=8–1
veh 4	1–2=9–10=15=14=13=17=14=8–1
veh 5	1–5–4=7=12=16=21=24=28=29–22=25–30=23–18–15–9–2–1
veh 6	1–5–4–6=11=20–27=28=21=29=22=17–14–8–1
veh 7	1–8=13=12=11=16=17=18=15–9–2–1
veh 8	1–2–9=15–18=23=26=30=29=25=30–26=18–15–9–2–1
veh 9	1–8–13=16=20=24=27=20=21=22=23–18–15–9–2–1

9D z = 515

veh 1	1=5=2=3=6=7=12=11=10=1
veh 2	1=9=14=15=16=1
veh 3	1=15=24=30=38=36=35=34=33=25=20=21=13=9–1
veh 4	1–5–2–3=4=7–12=19=32=40–32=31=17=16–1
veh 5	1–5=6=11=18=19–18=17=30=29=28=22=14–9–1
veh 6	1–5=10=17=24=23=22=13=8=9–1
veh 7	1–9–14=23–28=36=44=45=49–48=45=46=38–39=31–17–16–1
veh 8	1–15=23=29=37=38=39=40=47=39=30–29=24–15–1
veh 9	1–9–14–22=21=26=25–33=41=42=43=44–43=35=27=28=23–15–1
veh 10	1–15–23–28–27=26=34=42–43=48=49=50=46–50=45–44=37–29–24–15–1

10D z= 734

veh 1	1=2=7=13=12=6–1
veh 2	1=5=18=26=19=11=5=6=1
veh 3	1–6–7–2=3–4–10=17=25=24=23=22=21=20=12–6–1
veh 4	1–6=7=8=9=10=4=3=9=15=14=13=6–1
veh 5	1–6–13–14=8–9=16=24=32=37=36=27=30=20–12–6–1
veh 6	1–6–12=11–19=18=29=33=46=47=40=33=34=30–26–19=20–12–6–1
veh 7	1–6–13=21=27=31–37=43=45=38=32–28=25–17=16=15–14–13–6–1
veh 8	1–6–13–14=22=31=32=28=38–43–45=50=44=42=41=48–41=35=34–30=26–19–11–12–6–1
veh 9	1–6–13–14–15=23=31=37=38=43–42–44=49=48=47–40=34=26–19–11–12–6–1
veh 10	1–6–12–11–19–26=29=39=33–40=41=49=50=43=42=36=35=27–21–13–6–1

Note: "=" indicates a service and "–" a deadheading