On Arc-Routing Problems

Dissertation
zur Erlangung des Grades eines Doktors der
wirtschaftlichen Staatswissenschaften
(Dr. rer. pol.)
des Fachbereichs Recht- und Wirtschaftswissenschaften
der Johannes Gutenberg-Universität Mainz

vorgelegt von
Dipl.-Math. Claudia Bode
in Mainz

im Jahre 2013

Tag der mündlichen Prüfung: 4.12.2013

Contents

List of Papers

- Claudia Bode[1], Stefan Irnich[1] (2012). Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research* 60 (5), pp. 1167–1182.
- Claudia Bode, Stefan Irnich (2013). The shortest-path problem with resource constraints with $(k, 2)$-loop elimination and its application to the capacitated arc-routing problem. Technical Report LM-2013-01, Chair of Logistics Management, Johannes Gutenberg University Mainz, Mainz, Germany. Available online at `http://logistik.bwl.uni-mainz.de/158.php`. *Under review (major revision) in European Journal of Operational Research*
- Claudia Bode, Stefan Irnich (2012). In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. *accepted by Transportation Science*
- Claudia Bode (2013). Park and Loop Delivery Problems. Technical report, Chair of Logistics Management, Mainz School of Management and Economics, Johannes Gutenberg University, Mainz, Germany.

[1] Johannes Gutenberg-Universität Mainz, Lehrstuhl für BWL insb. Logistikmanagement, Jakob-Welder-Weg 9, D-55128 Mainz, Germany

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Combinatorial optimization problems deal with finding a best solution regarding a given objective function which is feasible with respect to constraints coming from the particular problem. Practical importance in the area of logistics are problems concerned to the aspect of routing. Arc routing problems belong to one class of those problems, where a service along streets rather than at points is important. The basic multiple-vehicle variant is called capacitated arc routing problem (CARP) and was introduced by Golden and Wong (1981). The edited book by Dror (2000) and Corberán and Laporte (2013) and the annotated bibliography by Corberán and Prins (2010) report the increased attention since then. The CARP is defined on an undirected graph, which is typically sparse, and a homogeneous fleet of vehicles with capacity Q is stationed at a distinguished node called the depot. Every street where good has to be delivered or collected has to be serviced by exactly one vehicle, while the capacity restriction of each vehicle has to be met. An optimal CARP solution consists of a cost minimal set of feasible routes. Most types of applications seek to minimize the total cost consisting of traversing and servicing cost, costs for using a vehicle and/or other related cost like costs for dumping waste. Typical constraints are capacity constraints and those that ensure connected routes for each vehicle. Applications are wide spread and some examples are salt gritting, road marking, waste collection or mail delivery.

Tackling more practical problems the CARP is not sufficient in oder to meet all restrictions arising in real world problems. Considering for example the mail delivery aspect there exist different delivery types in praxis. A common praxis in Canada is the usage of relay boxes to refill the handcart, see (Roy and Rousseau, 1989), while in the United States all mail is transported in a vehicle from the depot but the postman can get off his vehicle to perform smaller walking loops (Levy and Bodin, 1989). Therefore the basic problem has to be embedded in a more complex context and new models handling the restrictions coming from praxis must be developed.

To deal with arc routing problems many different solution strategies are available. Heuristics and metaheuristics are used for computing good upper bounds. Successful approaches for the CARP include all kinds of techniques used in the field on heuristics and are for example tabu search, genetic or memetic algorithms, guided local search, variable neighborhood search, ant colony optimization, and many more (Prins, 2013). State-of-the-art exact methods are based on a transformation of the arc-routing problem into a node-routing problem. Then, successful algorithms for vehicle routing problems (VRP) are applied, resulting in solutions that can be transformed back into integer

solutions for the original arc routing problem. Even so these approaches are rather successful, Letchford and Oukil (2009) already mentioned that even for relatively small CARP instances the resulting VRP is defined over a larger number of nodes and edges. The typically very sparse underlying graph for the CARP is replaced by a complete graph for the VRP. Therefore, this sparsity should be exploited. So far, good lower bounds are either obtained by pure polyhedral approaches to the CARP or by a combination of cut-and-column generation. However, even with a following branch-and-bound procedure no integral route variables can be guaranteed.

1.1 Goal of the thesis

The contribution of the thesis is to develop an exact solution method that fully exploits the sparse network in order to obtain integral solutions for the basic multiple-postman arc routing problem. Furthermore, in this work the basic problem is embedded in a more complex context in order to solve more realistic problems.

The first step is to develop a full branch-and-price algorithm for the capacitated arc routing problem. Therefore, a hierarchical branching scheme is developed guaranteeing integral route variables. Performing all computations on the original sparse network for the generation of new routes, an efficient pricing is presented. So far, this algorithm is limited to some special relaxations of the pricing problem. A next step is to analyze the pricing problem in more detail and develop new labeling algorithms for the different relaxations. Both, theoretical and empirical results are presented, showing how an efficient dominance relation can be defined resulting in a fast labeling algorithm.

Considering mail delivery modi used in praxis, the capacited arc routing problem does not model all restrictions. By combining the routing problem with a location aspect, more realistic problems can be defined. This thesis will analyze different ways of combining routing and location and will present mathematical formulations modeling the resulting problems.

1.2 Structure of the theses

The structure of the thesis by publication is as follows. Chapter 2 (Bode and Irnich, 2012) presents a full-fledged branch-and-price algorithm for the CARP. The master and pricing problem are presented in Section 2.3. Analyzing the structure of the pricing problems leads to an extended aggregated master program with additional dual cuts. Those dual-optimal inequalities ensure non-negative reduced costs for some of the variables in the pricing problem which provides algorithmic advantages and often results in a faster convergence of the column generation process. The full approach consisting of an initial cutting phase and branch-and-price phase thereafter is presented in Section 2.4. Possible relaxations are analyzed in Section 2.5. However, so far, only one of the relaxations is compatible with the presented branching scheme. Computational results presented in Section 2.6 show the strength of the proposed algorithm.

As pointed out in Chapter 2, there are additional requirements for solving the pricing problem to be compatible with the branching scheme. Therefore Chapter 3 (Bode and Irnich, 2013b)and 4 (Bode and Irnich, 2013a) outline these requirements and present results for the branch-and-price algorithm with modified algorithms solving the pricing relaxations. Chapter 3 focuses on k-loop elimination. Combining two sets of tasks for loop elimination is straight forward. However, the crucial part is to develop an efficient labeling algorithm, i.e., define effective dominance rules. Section 3.2 presents theoretical results for the combined (k_1, k_2)-loop elimination. Computational results applying $(k, 2)$ for several reasonable values of k to the CARP are presented in Section 3.3.

The companion paper in Chapter 4 studies several alternative relaxations like the ng-path relaxation. The necessary modifications for the labeling algorithms to be compatible with the branching scheme are presented in Section 4.3. Additional acceleration techniques to speed up the pricing are introduced in Section 4.4. Computational results in Section 4.5 include analyzing the impact of the acceleration techniques and a comparison of $(k, 2)$-loop elimination and the modified ng-path relaxation.

In Chapter 5 (Bode, 2013) we focus on different models for other types of arc routing problems. Combinations of arc routing with the question of locating depots can lead to different design restrictions of the routes. Section 5.2 classifies the literature into different design categories and Section 5.3 presents a total of four models for two specific designs which are called park and loop problem and combined park and loop with curbline mode. In Section 5.4 lower bounds and integer solutions are analyzed.

Finally, Chapter 6 summarizes and concludes the thesis.

Chapter 2

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

Claudia Bode, Stefan Irnich

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.

2.1 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.

The novelty of our approach comprises 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 K 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 2.2 reviews existing exact approaches for the CARP. Section 2.3 describes the Dantzig-Wolfe decomposition. The key components (cutting, pricing, branching) of the bap algorithm and their implementation are described in Section 2.4. Section 2.5 analyzes the interplay between cycle elimination and branching. Computational results in Section 2.6 show the capability of the new approach. Final conclusions are drawn in Section 2.7.

2.2 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.

Some authors (e.g. Belenguer and Benavent, 1998; Longo *et al.*, 2006) assume that $|K|$ is just a lower bound on the fleet size, others (e.g. Belenguer and Benavent, 1998) fix the number $|K|$ of vehicles. We also assume a fixed fleet size with $|K|$ vehicles, but point out that this assumption can affect the strength of the lower bounds and the computing times.

2.2.1 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).

A very successful variation of the solution approach of Baldacci *et al.* (2010), tailored to the CARP, is the recent work of Bartolini *et al.* (2013b). As a prerequisite, an upper bound ub is required. First, the CARP is transformed into a generalized VRP (GVRP). Here each required edge $e \in E_R$ is represented by two nodes i_e, j_e (one for each direction of service) so that any GVRP solution must cover exactly one node of the cluster $\{i_e, j_e\}$. Second, an extended set covering formulation with lifted odd cuts, rounded capacity cuts (Belenguer and Benavent, 2003), and subset-row inequalities (Jepsen *et al.*, 2008) is solved by a sequence of lower bounding procedures (producing non-decreasing lower bounds lb_1, lb_2, lb_3, and lb_4). Finally, as in the approach of Baldacci *et al.* (2010), all feasible CARP routes with reduced cost not exceeding the integrality gap $ub - lb_4$ are enumerated and a corresponding set-partitioning problem is solved using a MIP solver. In essence, the method is a VRP method as pricing and enumeration are performed on a transformed network that is dense and requires the covering of nodes.

Although these approaches are rather successful, Letchford and Oukil (2009) mentioned the following drawbacks: Even for relatively small CARP instances the resulting VRP is defined over a larger number of nodes and edges. In particular, the increase in the number of edges can be quadratic as the VRP 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 VRP instance might feature further symmetries.

2.2.2 Two-Index Formulation

Belenguer and Benavent (1998) 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. Auxiliary variables p_i^k for each node i and vehicle k are needed to ensure even node degrees. The two-index formulation is:

$$\min \sum_{k \in K} c^{serv,\top} x^k + \sum_{k \in K} c^\top y^k \tag{2.1}$$

$$\text{s.t.} \sum_{k \in K} x_e^k = 1 \quad \text{for all } e \in E_R \tag{2.2}$$

$$x^k(\delta_R(S)) + y^k(\delta(S)) \geq 2x_f^k \quad \text{for all } S \subseteq V \setminus \{d\},\ f \in E_R(S),\ k \in K \tag{2.3}$$

$$x^k(\delta_R(i)) + y^k(\delta(i)) = 2p_i^k \quad \text{for all } i \in V,\ k \in K \tag{2.4}$$

$$q^\top x^k \leq Q \quad \text{for all } k \in K \tag{2.5}$$

$$p^k \in \mathbb{Z}_+^{|V|}, x^k \in \{0,1\}^{|E_R|}, y^k \in \mathbb{Z}_+^{|E|} \quad \text{for all } k \in K \tag{2.6}$$

The objective (2.1) is the minimization of all service and deadheading costs. Since each required edge is serviced exactly once, as stated by (2.2), service costs c_e^{serv} have no impact on optimal decisions. The formulation in (Belenguer and Benavent, 1998) therefore omits service costs. Constraints (2.3) are the subtour-elimination constraints (SEC). As discussed in (Belenguer and Benavent, 1998, p. 169), the given SEC still allow disconnected components being deadheaded. A corresponding infeasible integer solution, denoted as *extended k-route*, is not optimal and can be excluded by adding constraints of the form (2.3) with $2x_f^k$ replaced by $2y_f^k$. The parity constraints (2.4) ensure that each vehicle can leave a node i after entering. The capacity constraints are given by (2.5) and integrality constraints by (2.6).

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.

2.2.3 One-Index Formulation

The *one-index formulation*, first considered independently by Letchford (1997) and Belenguer and Benavent (1998), 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:

$$\begin{aligned}
\min \quad & c^\top y && (2.7)\\
\text{s.t.} \quad & y(\delta(S)) \geq 2K(S) - |\delta_R(S)| \quad \text{for all } \emptyset \neq S \subseteq V \setminus \{d\} && (2.8)\\
& y(\delta(S)) \geq 1 \quad \text{for all } \emptyset \neq S \subseteq V,\ |\delta_R(S)| \text{ odd} && (2.9)\\
& y \in \mathbb{Z}_+^{|E|} && (2.10)
\end{aligned}$$

The objective (2.7) just takes the cost for deadheadings into account as service costs are constant. The capacity inequalities (2.8) 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 by solving a bin-packing problem. The odd-cut inequalities (2.9) require at least one deadheading if there is an odd number of required edges in the cut $\delta(S)$.

In combination with a powerful CARP heuristic, the one-index formulation provides a possible exact algorithm: Only if the heuristic comes across an optimal solution, the lower bound provided by (2.7)–(2.10) 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

$$\sum_{e \in E} d_{es} y_e \geq r_s \quad s \in \mathcal{S}, \tag{2.11}$$

where s is the index referring to a particular inequality, d_{es} is the coefficient of edge e in the inequality, and $\mathcal{S}$ the set of all valid inequalities. Some details and references on separation procedures are provided in Section 2.4.1 and in the appendix.

2.2.4 Extended Set-Covering Approach

Gómez-Cabrero *et al.* (2005) use a set-covering approach in which the standard set covering constraints are supplemented with the capacity inequalities (2.8) and odd-cut inequalities (2.9). 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 an initial 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 2.4.2.

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. The extended set-covering model for the CARP has binary decision variables λ_r for each route $r \in \Omega$ and is defined as follows:

$$\min \quad \sum_{r\in\Omega} c_r\lambda_r \tag{2.12}$$

$$\text{s.t.} \quad \sum_{r\in\Omega} \bar{x}_{er}\lambda_r \geq 1 \quad \text{for all } e \in E_R \tag{2.13}$$

$$\sum_{r\in\Omega} d_{sr}\lambda_r \geq r_s \quad \text{for all } s \in \mathcal{S} \tag{2.14}$$

$$\lambda_r \in \{0,1\} \quad \text{for all } r \in \Omega \tag{2.15}$$

The extension to the standard set-covering model (2.12), (2.13) and (2.15) is the addition of transformed cuts (2.14) derived from (2.11), 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) 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 (2.12)–(2.15) sometimes outperform those of the one-index formulation. The approach is therefore attractive but incomplete 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.

2.3 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 (2.1)–(2.6) together with the valid cuts (2.11). As in other routing problems, we assume that the covering/partitioning constraints (2.2) are the coupling constraints. Additionally, the valid cuts (2.11) (or any subset of active cuts) are coupling constraints.

2.3.1 Column-Generation Formulation

First we analyze the domain $D = \{(x, y, p) : \text{fulfilling (2.3)–(2.6)}\}$ in order to describe the general structure of the pricing problem as well as the extreme points of D that correspond to the variables of the column-generation formulation a.k.a. *extensive formulation* (see Lübbecke and Desrosiers, 2005). The domain D is separable by vehicle (index k) and can therefore be described as the cartesian product of domains $D^k = \{(x^k, y^k, p^k) : \text{that fulfill (2.3)–(2.6)}\}$ 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 combinations 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 to 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 2.3.4).

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 2.2.4. The integer master program (IMP) of the CARP reads as follows:

$$\min \quad \sum_{k \in K} c^\top \lambda^k \tag{2.16}$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{r \in \Omega} \bar{x}_{er} \lambda_r^k = 1 \quad \text{for all } e \in E_R \tag{2.17}$$

$$\sum_{k \in K} \sum_{r \in \Omega} d_{sr} \lambda_r^k \geq r_s \quad \text{for all } s \in \mathcal{S} \tag{2.18}$$

$$\sum_{r \in \Omega} \mathbf{1}^\top \lambda_r^k = 1 \quad \text{for all } k \in K \tag{2.19}$$

$$\lambda^k \geq \mathbf{0} \quad (\in \mathbb{R}^{|\Omega|}) \quad \text{for all } k \in K \tag{2.20}$$

$$x_e^k = \sum_{r \in \Omega} \bar{x}_{er} \lambda_r^k, \quad y_e^k = \sum_{r \in \Omega} \bar{y}_{er} \lambda_r^k \quad \text{for all } e \in E_R / e \in E,\ k \in K \tag{2.21}$$

$$x^k \in \{0,1\}^{|E_R|}, y^k \in \mathbb{Z}_+^{|E|} \quad \text{for all } k \in K \tag{2.22}$$

The objective (2.16) minimizes over the costs of all tours. Equalities (2.17) ensure that every required edge is covered exactly once. The reformulated cuts are given by (2.18). Equalities (2.19) are convexity constraints and require each vehicle to perform a CARP tour. Constraints (2.21) couple the variables for service and deadheading with the tour variables and constraints (2.22) ensure the integrality of the solution.

The LP-relaxation of (2.16)–(2.22) is the master program (MP). As there is no need to keep the coupling constraints (2.21) when integrality is relaxed, MP reduces to (2.16)–(2.20). 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.

2.3.2 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 partitioning constraints (2.17), $\beta = (\beta_s)_{s \in \mathcal{S}}$ to the cuts (2.18), and

$\mu = (\mu^k)_{k \in K}$ to the convexity constraints (2.19) be given. Omitting the index k of the vehicle, the pricing problem is

$$z_{PP} = \min \tilde{c}^{serv,\top} x + \tilde{c}^{\top} y - \mu \qquad \text{s.t.} \qquad (2.3)\text{–}(2.6),$$

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

$$\tilde{c}_e^{serv} = c_e^{serv} - \pi_e \text{ for all } e \in E_R \qquad \text{and} \qquad \tilde{c}_e = c_e - \sum_{s \in \mathcal{S}} d_{es} \beta_s \text{ for all } e \in E. \quad (2.23)$$

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 (2.3)–(2.6) except for the binary requirements 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, we allow a point being represented by one or several corresponding service sequences. Such a sequence arises naturally when routes are determined as shortest paths in pricing (see Section 2.4.2), which is the only efficient method currently available. 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 2.4.2.

2.3.3 Aggregation

In order to eliminate the symmetry from (2.16)–(2.22) with respect to the vehicles, aggregation over $k \in K$ identical subproblems can be applied. The aggregated service, deadheading, and route variables are

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

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

$$\min \quad \sum_{r \in \Omega} c_r \lambda_r \tag{2.24}$$

$$\text{s.t.} \quad \sum_{r \in \Omega} \bar{x}_{er} \lambda_r = 1 \quad \text{for all } e \in E_R, \qquad \sum_{r \in \Omega} d_{sr} \lambda_r \geq r_s \quad \text{for all } s \in \mathcal{S} \tag{2.25}$$

$$\mathbf{1}^\top \lambda = |K|, \qquad \lambda \geq \mathbf{0}, \lambda \in \mathbb{Z}^{|\Omega|} \tag{2.26}$$

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 (2.24)–(2.26) 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 2.4.3). 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 2.2.3.

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 (2.24)–(2.26).

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

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

The variable $h_{ee'}$ 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 $h_{ee'} = 0$ is equivalent to $x_e^k + x_{e'}^k \leq 1$ for all $k \in K$ in the two-index formulation, while $h_{ee'} = 1$ 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 undirected 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

$$f_{ee'} = \sum_{r \in \Omega} f_{ee'r} \lambda_r \in \{0, 1\} \quad \text{for all } e, e' \in E_R \tag{2.27}$$

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 by route $r \in \Omega$. Note that f is symmetric, i.e., $f_{ee'} = f_{e'e}$ holds.

The follower information suffices to ensure integrality of the route variables λ as can be seen as follows: Let $H \subset E_R \times E_R$ be the binary relation representing the values $h_{ee'}$, i.e., $(e, e') \in H$ if and only if $h_{ee'} = 1$. Similarly, let F be the follower relation with $(e, e') \in F$ if and only if $f_{ee'} = 1$. Then H is the reflexive and transitive closure of F. Hence, binary values of $f_{ee'}$ imply binary values of $h_{ee'}$, 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 2.4.3.

2.3.4 Dual-Optimal Inequalities

The equation (2.23) 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):

$$\min \quad \sum_{r \in \Omega} c_r \lambda_r + \sum_{e \in E} (2c_e) z_e \tag{2.28}$$

$$\text{s.t.} \quad \sum_{r \in \Omega} \bar{x}_{er} \lambda_r = 1 \quad \text{for all } e \in E_R \tag{2.29}$$

$$\sum_{r \in \Omega} d_{sr} \lambda_r + \sum_{e \in E} (2d_{es}) z_e \geq r_s \quad \text{for all } s \in \mathcal{S} \tag{2.30}$$

$$\mathbf{1}^\top \lambda = |K| \tag{2.31}$$
$$\lambda \geq \mathbf{0}, z \geq \mathbf{0} \tag{2.32}$$

Obviously, deadheading twice through e produces a cost of $2c_e$ in (2.28), it has no impact on partitioning (2.29), but can have non-zero coefficients $2d_{es}$ in the cuts (2.30). As C_e is an extreme ray, it has coefficient zero in the generalized convexity constraint (2.31).

The additional variables z_e allow extended k-routes (Belenguer and Benavent, 1998) in eMP, the primal problem, and correspond to inequalities in the dual problem. In fact, the presence of z_e in eMP gives a dual inequality of the form

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

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 2.4.2.

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).

2.4 Cut-First Branch-and-Price-Second Approach

An outline of the overall cut-first bap-second approach is shown in Algorithm 1. Its three key components are the cut generation procedure, the pricer, and the branching scheme. These components will be explained in the following.

Algorithm 1: Cut-First Branch-and-Price-Second Algorithm

1. Solve LP-relaxation of one-index formulation (2.7)–(2.10) with a cutting-plane algorithm
2. Identify binding cuts (odd cuts, capacity cuts, disjoint-path inequalities DP1, DP2, DP3) and odd/capacity cuts with rhs>0 for singleton sets $S = \{i\}$ with $i \in V \setminus \{d\}$; the index set of these cuts is $\mathcal{S}$
3. Solve extended aggregated integer master program eMP (2.28)–(2.32) with branch-and-price; use set $\mathcal{S}$ from step 2 for valid inequalities (2.30)

2.4.1 Cutting

Before starting the bap phase, the one-index formulation (2.7)–(2.10) is solved with a cutting-plane algorithm in order to identify the cuts that are finally binding. Details about the particular separation procedures, the order in which they are invoked, the number of separated and finally binding cuts, and computation times are given in the appendix.

Here, we briefly sketch the separation routines. To find violated odd cuts (2.9) the efficient separation algorithm of Letchford *et al.* (2008) is applied. Capacity inequalities (2.8) are separated using 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 $\lceil q(E_R(S) \cup \delta_R(S))/Q \rceil$. Disjoint-path inequalities (of type dp1, dp2 and dp3) were presented by Belenguer and Benavent (2003) together with a rather complex heuristic scheme to separate violated inequalities. We implemented a similar heuristic trying to be as close as possible to the original algorithm.

Only those cuts that are finally binding in the one-index formulation (plus odd cuts for singleton sets) 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.

2.4.2 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 $\mathcal{O}\left(Q(|E| + |V| \log |V|)\right)$.

2.4.3 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. (2.21)). 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 2.3.3).

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 (2.30). Consequently, the equations (2.23) 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. (2.27)) 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 2.4.3 do 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 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 subsets E_R^ℓ 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 subset 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 subset of size less than or equal to 5. If e^* and $e^{*\prime}$ are already in the same subset, the addition of an associated constraint will not alter the given partitioning. Otherwise, the two subsets of e^* and $e^{*\prime}$, say E_R^ℓ and E_R^m are merged resulting in a single subset of size $|E_R^\ell| + |E_R^m|$. If no induced subset is of size

less than or equal to 5, we only consider pairs with smallest induced subset and proceed as described before. The idea behind this rule is that small subsets are algorithmically attractive because we have to enumerate permutations of their edges to map branching decisions to the pricing network (see Section 2.4.3).

Integer Solutions from Follower Information

The analysis undertaken in Section 2.3.3 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. A detailed example is provided in the appendix.

With a relaxed pricing, it can happen that all follower variables $f_{ee'}$ are binary but route variables $\bar{\lambda}$ are still fractional In this case, nevertheless, an integer solution can implicitly be obtained from the fractional master program eMP. In fact, the binary follower information implies a unique partitioning 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 $|K|$ subsets 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 subset 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 2.1. 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, dotted 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. The appendix provides a detailed example.

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

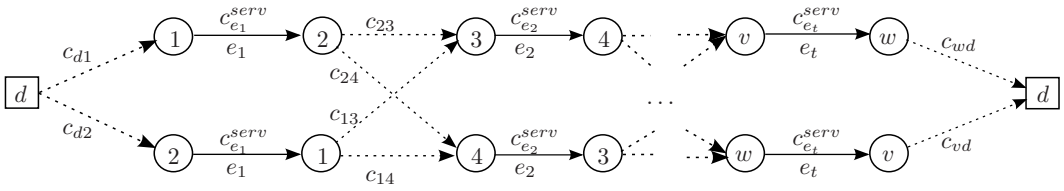


Figure 2.1: Auxiliary network

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 by summing up serviced costs $\tilde{c}^{serv}_e + \tilde{c}^{serv}_{e'}$ 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 2:

The computation of a minimum-cost path p is similar to the shortest-path computation in the auxiliary network described in 2.4.3. We just elaborate the differences: First, costs

Algorithm 2: 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 the partitioning $E_R^1 \cup E_R^2 \cup \ldots \cup E_R^q$ of E_R induced by F and N

for $\ell = 1, \ldots, q$ **do**

  if $|E_R^\ell| > 1$ **then**

    Delete all required edges $e \in E_R^\ell$ from the network

    Compute all feasible sequences $s = (e_1, e_2 \ldots, e_t)$ on all subsets of edges in E_R^ℓ

    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 $\sum_{m=1}^{t} q_{e_m}$, associated task is ℓ

c_e and c_e^{serv} have to be replaced by reduced costs $\tilde{c}_e$ and $\tilde{c}_e^{serv}$ defined by (2.23). Note that dual-optimal inequalities (see Section 2.3.4) 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). A detailed example of the network modification in included in the appendix.

The enumeration of all possible sequences s may produce a network with an exponential number of edges. By selecting edges in the follower branching rule carefully, we try to keep the resulting subsets small. Therewith, we postpone the exponential growing of the network to some deeper branches of the tree (that were not reached in our experiments).

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 $\sum_{m=1}^{t} q_{e_m}$, a single cost-minimal edge can be chosen while the other edges can be discarded.

2.5 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), Ahr (2004), and Gómez-Cabrero *et al.* (2005) did not reveal the full potential of cycle elimination. The reason is that no odd-cut inequalities (2.9), capacity inequalities (2.8), and disjoint-path inequalities were present in the column-generation formulation. While Section 2.6.1 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 2.3.2, 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 2.4.3). 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 2.4.3) such edges get 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 ℓ that are assigned to the new edges modeling a sequence through the same component E_R^ℓ (see last step of Algorithm 2) 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), is 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).

2.6 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) were also used to analyze the methods presented in the review (Section 2.2). 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, respectively. The latter set is based on parts of the road network of Cumbria.

The following results were performed on a standard PC with an Intel(R) Core(TM) i7-2600 at 3.4 GHz with 16 GB of main memory. The algorithm was coded in C++ and the callable library of CPLEX 12.2 was used to iteratively solve LPs and MIPs (reoptimization and separation) in the cutting phase and to reoptimize the RMP.

The CPU2006 benchmarks, available at `http://www.spec.org/cpu2006/results`, allow a comparison of running times of different machines. Since there are no results for the Pentium IV at 2.8 GHz and 2.4 GHz machines of Longo *et al.* (2006) and Baldacci and Maniezzo (2006), we use the factors 1.8 and 2.0 reported in Bartolini *et al.* (2013b). According to CPU2006 benchmarks the SPECint and SPECfp scores for the Intel(R) Core(TM) i7-2600 used in this paper are 42.7 and 55.1, whereas the scores for the Intel Xeon E5310 are 10.0 and 8.72. Therefore, our computer is approximately 5, 10 and 11 times faster than that of Bartolini *et al.* (2013b), Longo *et al.* (2006) and Baldacci and Maniezzo (2006), respectively.
For the computational analysis we use the following notation:

lb_*	lower bound obtained by algorithm of *; BB= cutting-plane algorithm of Belenguer and Benavent (2003); A= Ahr (2004); L= linear relaxation (root) by Longo *et al.* (2006); LO= linear relaxation (root, elementary routes) by Letchford and Oukil (2009); MPS= dual ascent and cutting-plane algorithm of Martinelli *et al.* (2013)
lb_{own}^{root} / lb_{own}^{tree}	lower bound obtained in our linear relaxation (root) or at termination of phase II (tree)
ub_{best}	best known upper bound
comp. by	first paper where the currently best upper bound was computed; H= Hertz *et al.* (2000); La= Lacomme *et al.* (2001); BE= Brandão and Eglese (2008); PD= Polacek *et al.* (2008); SCC= Santos *et al.* (2010); BLC= Baldacci *et al.* (2010); Be= Beullens *et al.* (2003)
opt	value of optimal solution
proved by *lb*	first paper where optimality is proved due to $\lvert ub - lb \rvert < 1$
proved by sol	first paper where optimality is proved by computing an integer solution; B= Benavent *et al.* (1992); BB= Belenguer and Benavent (2003); Lo= Longo *et al.* (2006); BM= Baldacci and Maniezzo (2006); LO= Letchford and Oukil (2009); own= own solution
time	computation in time seconds for phase I or phase II; the time limit for both phases is 4 hours (indicated by *4h*)
B&B nodes	number of solved nodes in our branch-and-bound tree
branching D/E/F	number of different branching decisions: D= node degree; E= edge flow; F= follower

For the sake of brevity, all computational results regarding phase I (cutting-plane

algorithm) are presented in the appendix.

2.6.1 Linear Relaxation Results

For phase II, the following two column-generation acceleration techniques were implemented: Before calling the exact pricer, we apply the heuristic of Letchford and Oukil (2009) (considering 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 (2.29) 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).

We begin with the presentation of the results on the linear relaxation of the master program eMP (2.28)–(2.32). These results are shown in the first three sections of the Tables 2.1-2.2 (results for the small-sized benchmark sets `kshs` and `gdb` are presented in the appendix).

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 eMP. Compared to the lower bounds lb_{BB} of Belenguer and Benavent (2003), the values lb^{root}_{own} are sometimes worse (for the `bccm` instances `2b` and `6b` as well as `egl` instance `s2-a`). We suspect that this is due to the fact that the separation of disjoint-path inequalities in phase I is performed with a complex heuristic that certainly differs at some points from the one used to compute lb_{BB} (see appendix for the discussion of the phase I results). Conversely, for several instances better lower bounds were obtained by column generation, i.e., $lb^{root}_{own} > \max\{lb_{BB}, lb_A\}$. This is the case for one `kshs` instance (`kshs4`), two `gdb` instances (`gdb8` and `gdb12`), nine `bccm` instances, and all `egl` instances except for `s2-a`.

Comparing the pure set-partition formulation (without additional cuts) of Letchford and Oukil (2009) with our approach, lb_{LO} never exceeds lb^{root}_{own} except for the two `egl` instances `e4-c` and `s1-c`. It seems that the addition of cuts to eMP is most of the time more important than using elementary routes alone.

Longo *et al.* (2006) presented detailed lower bound results only for `bccm` and `egl` instances. Their 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` except for `s3-a` with an identical bound). For the `bccm` instances `1c`, `3c`, and `5b`, however, lb^{root}_{own} exceeds lb_L.

Some concluding remarks on the comparison with (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 a 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.

name	$\|V\|$	$\|E\|$	$\|K\|$	lb_{BB}	lb_A	lb_L	lb_{LO}	lb_{BCL}	lb^{root}_{own}	lb^{tree}_{own}	ub_{best} or opt	comp. by	proved by lb	proved by sol	time phase I	time root phase II	time phase II	B&B nodes	branching D/E/F
1a	24	39	2	247	247	247	220	247	247	-	247	H	B	own	0.1	1.6	16.9	24	2/0/21
1b	24	39	3	247	247	247	225	247	247	-	247	H	B	own	0.2	1.1	4.1	13	1/0/11
1c	24	39	8	309	309	312	313	314	315	-	319	H	-	Lo	0.5	0.3	55.6	649	511/4/132
2a	24	34	2	298	298	298	277	298	298	-	298	H	BB	own	0.1	1.7	10.2	14	1/0/12
2b	24	34	3	330	328	329	304	330	328	-	330	La	BB	Lo	0.1	1.4	8.9	29	18/1/9
2c	24	34	8	526	526	528	528	528	528	-	528	H	Lo	Lo	0.4	0.1	0.4	1	0/0/0
3a	24	35	2	105	105	105	93	105	105	-	105	H	BB	own	0.1	1.1	5.5	22	1/0/20
3b	24	35	3	111	111	111	101	111	111	-	111	H	BB	own	0.3	0.6	2.2	17	1/0/15
3c	24	35	7	161	159	161	155	162	162	-	162	H	-	Lo	0.7	0.2	1.1	21	16/0/4
4a	41	69	3	522	522	522	478	522	522	-	522	H	BB	own	0.3	16.0	412.9	34	2/0/31
4b	41	69	4	534	534	534	492	534	534	-	534	H	BB	own	0.5	7.8	72.0	29	1/0/27
4c	41	69	5	550	550	550	515	550	550	-	550	La	BB	own	0.5	5.4	31.0	23	2/0/20
4d	41	69	9	644	642	648	621	649	646	-	652	La	-	own	8.0	3.8	707.8	836	821/5/9
5a	34	65	3	566	566	566	524	566	566	-	566	H	BB	own	0.3	6.7	75.4	39	1/0/37
5b	34	65	4	589	586	588	548	588	589	-	589	H	BB	own	0.5	4.9	24.4	34	1/0/32
5c	34	65	5	612	610	613	578	613	612	-	617	H	-	own	0.8	5.7	405.2	465	407/17/37
5d	34	65	9	714	714	716	689	717	715	-	718	PD	-	own	0.9	3.3	27.3	76	68/1/6
6a	31	50	3	330	330	330	305	330	330	-	330	H	B	own	0.1	2.6	19.4	30	3/0/25
6b	31	50	4	338	336	337	315	337	336	-	340	La	-	BM	0.3	1.7	208.7	522	227/14/21
6c	31	50	10	418	414	420	405	421	418	-	424	H	-	BM	2.0	0.6	4024.9	9683	4637/161/43
7a	40	66	3	382	382	382	358	382	382	382	382	H	B	-	0.1	5.8	*4h*	10579	2464/121/2704
7b	40	66	4	386	386	386	361	386	386	-	386	H	BB	own	0.3	3.9	34.5	81	9/0/31
7c	40	66	9	436	430	436	407	437	436	437	437	H	-	Lo	3.1	1.6	*4h*	19773	2542/1420/5924
8a	30	63	3	522	522	522	489	522	522	-	522	H	B	own	0.1	4.8	37.7	65	0/0/32
8b	30	63	4	531	531	531	502	531	531	-	531	H	B	own	0.4	2.6	17.3	65	0/0/32
8c	30	63	9	653	645	654	638	655	654	-	657	Be	-	own	2.1	1.6	23.0	245	113/1/8
9a	50	92	3	450	450	450	407	450	450	-	450	H	B	own	0.4	32.9	684.4	107	3/1/49
9b	50	92	4	453	453	453	412	453	453	-	453	H	B	own	0.3	13.3	153.7	123	8/2/51
9c	50	92	5	459	459	459	419	459	459	-	459	H	B	own	1.1	9.7	73.7	113	4/1/51
9d	50	92	10	509	505	512	484	512	510	515	515[a]	PD	own	own	3.0	7.6	*4h*	8235	3956/159/2
10a	50	97	3	637	637	637	590	637	637	-	637	H	B	own	0.2	27.3	541.3	125	5/1/56
10b	50	97	4	645	645	645	597	645	645	-	645	H	B	own	1.1	21.8	7554.2	3545	809/11/952
10c	50	97	5	655	655	655	609	655	655	-	655	La	BB	own	0.8	14.1	171.0	117	2/2/54
10d	50	97	10	732	731	734	695	734	734	-	734	PD	Lo	own	104.0	9.5	91.6	109	26/2/26

Table 2.1: Results for the `bccm` instances at the end of phase II.

[a] A new best (=optimal) integer solution with value 515 was found with a different branching scheme; see p. 28

name	$\|V\|$	$\|E\|$	$\|K\|$	lb_{BB}	lb_A	lb_L	lb_{LO}	lb_{MPS}	lb_{BCL}	lb^{root}_{own}	lb^{tree}_{own}	ub_{best} or opt	comp. by	proved by sol	time phase I	time phase II	$B\&B$ nodes	branching D/E/F
e1-a	77	98	5	3515	3516	3548	3425	3527	3548	3545	-	3548	La	Lo	49.3	1572	135	35/0/79
e1-b	77	98	7	4436	4436	4468	4291	4468	4498	4464	-	4498	La	BM	102.5	1733	617	588/12/16
e1-c	77	98	10	5453	5481	5542	5472	5513	5595	5523	5545	5595	La	BLC	145.5	*4h*	4248	3102/291/852
e2-a	77	98	7	4994	4963	5011	4832	4995	5012	4996	-	5018	La	BM	25.3	1966	312	268/4/38
e2-b	77	98	10	6249	6271	6280	6105	6273	6284	6273	6301	6317	BE	-	56.9	*4h*	3822	1996/641/1184
e2-c	77	98	14	8114	8155	8234	8187	8165	8335	8202	8244	8335	BE	BLC	118.9	*4h*	5921	5915/0/0
e3-a	77	98	8	5869	5866	5898	5706	5898	5898	5894	-	5898	La	Lo	33.6	810	144	59/2/82
e3-b	77	98	12	7646	7649	7697	7541	7649	7711	7684	7728	7775	PD	-	51.2	*4h*	3298	3297/0/0
e3-c	77	98	17	10019	10119	10163	10086	10138	10244	10145	10191	10292	PD	-	102.8	*4h*	5658	5650/0/0
e4-a	77	98	9	6372	6378	6395	6233	6378	6395	6389	6408	6444	SCC	-	11.3	*4h*	1246	526/232/478
e4-b	77	98	14	8809	8838	8884	8678	8838	8935	8852	8892	8961	BLC	-	19.4	*4h*	3983	3905/0/0
e4-c	77	98	19	11276	11376	11427	11416	11383	11493	11411	11457	11562	BLC	-	257.0	*4h*	6031	6000/0/0
s1-a	140	190	7	4992	4975	5014	4985	5010	5018	5011	-	5018	La	BM	208.3	8462	100	73/3/23
s1-b	140	190	10	6201	6180	6379	6284	6368	6388	6370	6388	6388	BE	BLC	454.5	*4h*	364	360/3/1
s1-c	140	190	14	8310	8286	8480	8423	8404	8518	8418	8441	8518	La	BLC	605.3	*4h*	436	436/0/0
s2-a	140	190	14	9780	9718	9824	9667	9737	9825	9791	9803	9884	SCC	-	283.1	*4h*	434	434/0/0
s2-b	140	190	20	12286	12835	12968	12801	12901	13017	12949	12970	13100	BE	-	1136,3	*4h*	2344	2344/0/0
s2-c	140	190	27	16221	16216	16353	16262	16274	16425	16314	16352	16425	BE	BLC	2154.6	*4h*	2640	2615/0/0
s3-a	140	190	15	10025	9991	10143	9925	10083	10145	10143	10160	10220	SCC	-	927.8	*4h*	408	408/0/0
s3-b	140	190	22	13554	13520	13616	13388	13568	13648	13598	13631	13682	PD	-	1291.5	*4h*	1519	1519/0/0
s3-c	140	190	29	16969	16958	17100	17014	17019	17188	17058	17097	17188	BLC	BLC	2750.9	*4h*	3398	3398/0/0
s4-a	140	190	19	12027	12007	12143	11905	12026	12141	12126	12149	12268	SCC	-	476.9	*4h*	1165	1165/0/0
s4-b	140	190	27	15933	15897	16093	15891	16001	16098	16066	16105	16321	PD	-	1003.3	*4h*	2686	2683/0/0
s4-c	140	190	35	20179	20176	20375	20197	20256	20430	20340	20376	20481	BLC	-	1695.6	*4h*	4019	3997/0/0

Table 2.2: Results for the `egl` instances at the end of phase II.

name	ub_{best} or opt	lb^{root}	time phase II	lb^{root}	time phase II	lb^{root}	time phase II	lb^{root}	time phase II	remain. gap
elimin. of		$k=2$-loops		$k=3$-loops		$k=4$-loops		$k=5$-loops		
e1-a	3548	3545	30	3546 (+1)	74	3546 (+0)	368	3548 (+2)	2575	0
e1-b	4498	4464	12	4465 (+1)	46	4467 (+2)	196	4470 (+3)	2474	28
e1-c	5595	5523	13	5528 (+5)	31	5532 (+4)	56	5535 (+3)	1127	60
e2-a	5018	4996	24	4996 (+0)	98	4999 (+3)	1247	4999 (+0)	22301	19
e2-b	6317	6273	21	6280 (+7)	53	6283 (+3)	272	6283 (+0)	3611	34
e2-c	8335	8202	10	8227 (+25)	21	8263 (+36)	55	8269 (+6)	1083	66
e3-a	5898	5894	40	5895 (+1)	297	5895 (+0)	991	5895 (+0)	25083	3
e3-b	7775	7684	21	7699 (+15)	45	7704 (+5)	190	7709 (+5)	4246	66
e3-c	10292	10145	11	10176 (+31)	23	10182 (+6)	65	10183 (+1)	1169	109
e4-a	6444	6389	49	6389 (+0)	336	6389 (+0)	422	6389 (+0)	7571	55
e4-b	8961	8852	20	8862 (+10)	46	8865 (+3)	244	8870 (+5)	2195	91
e4-c	11562	11411	16	11438 (+27)	43	11463 (+25)	159	11465 (+2)	1774	97
s1-a	5018	5011	347	5012 (+1)	687	5013 (+1)	2517	5015 (+2)	11330	5
s1-b	6388	6370	437	6373 (+3)	559	6376 (+3)	2073	6376 (+0)	8635	12
s1-c	8518	8418	227	8457 (+39)	87	8468 (+11)	157	8475 (+7)	755	43
s2-a	9884	9791	263	9795 (+4)	784	9795 (+0)	1977	9798 (+3)	26124	89
s2-b	13100	12949	125	12955 (+6)	424	12960 (+5)	848	12963 (+3)	5976	137
s2-c	16425	16314	125	16332 (+18)	180	16338 (+6)	357	16340 (+2)	3612	85
s3-a	10220	10143	297	10145 (+2)	973	10145 (+0)	2718	10147 (+2)	41494	75
s3-b	13682	13598	177	13604 (+6)	298	13605 (+1)	1024	13606 (+1)	12671	76
s3-c	17188	17058	71	17089 (+31)	146	17090 (+1)	416	17095 (+5)	3271	93
s4-a	12268	12126	180	12129 (+3)	467	12129 (+0)	1248	12132 (+3)	32172	139
s4-b	16321	16066	124	16071 (+5)	328	16073 (+2)	688	16077 (+4)	15605	248
s4-c	20481	20340	133	20362 (+22)	243	20375 (+13)	352	20383 (+8)	3932	98

Table 2.3: Cycle Elimination for the `egl` instances.

Finally, the strongest relaxation bound lb_{BCL} obtained by Bartolini *et al.* (2013b) almost always exceeds lb^{root}_{own} as their relaxation is stronger. Elementary routes and different types of cuts (including the non-robust subset-row inequalities on the master program) are combined in the approach. The only cuts not used in their implementation are disjoint-path inequalities. This fact can explain why for the two `bccm` instances `1c` and `5b` $lb_{BCL} < lb^{root}_{own}$ holds.

Cycle Elimination Result

Recall from Section 2.5 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 2.3 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 becomes time consuming (in one case more than 10 hours). On average, the increase of the lower bound is 11.0 going from $k = 2$ to $k = 3$, 5.4 from $k = 3$ to $k = 4$, and 2.8 from $k = 4$ to $k = 5$. While for some instances the increase is marginal, it can become substantial (e.g. for `e2-c` an overall increase of 25+36+6=67). On the downside, average computation times increase also by factor 2.3 from $k = 2$ to $k = 3$, 3.0 from $k = 3$ to $k = 4$, and 12.9 from $k = 4$ to $k = 5$.

For the 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`, `s1-a`, `s3-a`, and `s4-c`; see also Table 2.2). Interestingly, these are often 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.

2.6.2 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 strategy. In case of a tie, the last node added is selected first. We found that this strategy is fundamental for finding integer solutions early 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 2.1-2.2 and columns headed lb^{tree}_{own} (see appendix for `kshs` and `gdb` instances). 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 2.1. For all `bccm` instances 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 the instance `9d`, we have found an optimal solution with value 515 during experiments with other branching schemes. Hence, for `7a`, `7c`, and `9d` the final setup was only able to prove optimality due to the tree lower bound. For five instances (`4d`, `5c`, `5d`, `8c`, and `9d`), optimality was proven 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 determine optimal integer solutions in 31 out of 34 cases. In contrast, the node-routing approaches of Bartolini *et al.* (2013b) determine *and* prove optimal solutions in 29 cases.

Results for the `egl` test set are presented in Table 2.2. With the given setup, five instances (`e1-a`, `e1-b`, `e2-a`, `e3-a`, and `s1-a`) are solved to proved optimality in 4 hours. These instances were already solved either by Longo *et al.* (2006) or Baldacci and Maniezzo (2006). For `s1-b` we are able to close the gap, but we were unable to determine an optimal solution. Thus, we prove optimality of `s1-b` by lower bound using the upper bound presented in (Brandão and Eglese, 2008).

(Bartolini *et al.*, 2013b) are able to solve ten of the 24 `egl` instances (however, `e2-a` remains unsolved in their approach). The better lower bounds lb_4 in the final bounding procedure allow them to close the gap for five additional instance `e1-c`, `e2-c`, `s1-c`, `s2-c`, and `s3-c`.

The lower bound lb_{own}^{tree} resulting from the partial solution of the branch-and-bound tree exceeds the best known lower bounds lb_L presented by Longo *et al.* (2006) for 14 instances. The bound lb_{own}^{tree} also exceeds the lower bound presented by Bartolini *et al.* (2013b) in six cases (`e2-b`, `e3-b`, `e4-a`, `s3-a`, `s4-a`, and `s4-b`). The appendix contains a table that lists the best known lower and upper bounds with references.

2.7 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 (i.e. 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 for 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 (1998) are solved now. For the (Eglese and Li, 1992) benchmark set, optimality of one more instance was shown (independently the results of Bartolini *et al.* (2013b)) and some lower bounds were improved.

We see the following possible avenues for future research: A deeper analysis of the polyhedral structure might lead to new strong valid inequalities and might help to further strengthen the initial lower bound. All 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.

2.8 Bibliography

Ahr, D. (2004). *Contributions to Multiple Postmen Problems.* Phd dissertation, Department of Computer Science, Heidelberg University, Heidelberg, Germany.

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.

Bartolini, E., Cordeau, J.-F., and Laporte, G. (2013b). Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming*, **137**(1-2), 409–452.

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

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

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

Benavent, E., Campos, V., Corberán, Á., and Mota, E. (1992). The capacitated arc routing problem: Lower bounds. *Networks*, **22**, 669–690.

Beullens, P., Muyldermans, L., Cattrysse, D., and van Oudheusden, D. (2003). A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, **147**(3), 629–643.

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, Á. and Prins, C. (2010). Recent results on arc routing problems: An annotated bibliography. *Networks*, **56**(1), 50–69.

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. M., and Benavent, E. (2005). Cutting planes and column generation for the capacitated arc routing problem. presented at ORP3, Valencia, Spain.

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.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**(2), 497–511.

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.

Letchford, A. N. (1997). *Polyhedral results for some arc routing problems*. Phd dissertation, Department of Management Science, Lancaster University.

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.

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

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

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

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

Martinelli, R., Poggi de Aragão, M., and Subramanian, A. (2013). Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, **40**(8), 2145–2160.

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

Polacek, M., Doerner, K., Hartl, R., and Maniezzo, V. (2008). A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, **14**, 405–423.

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.

Santos, L., Coutinho-Rodrigues, J., and Current, J. R. (2010). An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B*, **44**(2), 246 – 266.

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.

2.9 Appendix

Integer Solutions from Fractional Solutions

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 2.2, 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

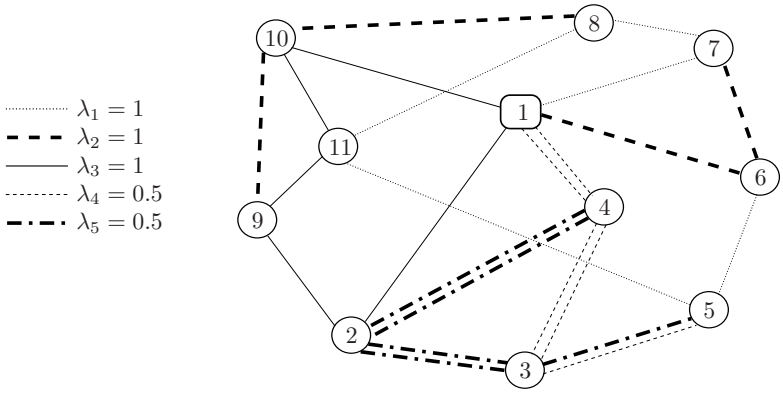


Figure 2.2: Fractional Solution

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 (2.27)).

Note that this convex combination of routes produces integer edge flows on all edges (see Figure 2.2). The implied follower information is

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

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 subsets $E_R^1 \cup E_R^2 \cup E_R^3 \cup E_R^4$. The follower relation defined by routes 4 and 5 results in one subset $E_R^4 = \{\{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{3,5\}\}$. This implies four sequences of required edges:

$$s_1 \quad = \quad (\{1,7\}, \{7,8\}, \{8,11\}, \{11,5\}, \{5,6\})$$

$$
\begin{aligned}
s_2 &= (\{1,6\},\{6,7\},\{8,10\},\{10,9\},) \\
s_3 &= (\{2,9\},\{9,11\},\{11,10\},\{10,1\}) \\
s'_4 &= (\{1,4\},\{4,3\},\{3,5\},\{3,2\},\{2,4\})
\end{aligned}
$$

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

Network Modifications

Example: We consider a graph G with eight nodes $\{a, \dots, g\}$ and twelve edges (see Figure 2.3). 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 induce the subset $\{e_1, e_2, e_3, e_4\}$ 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 violate 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 2.4.

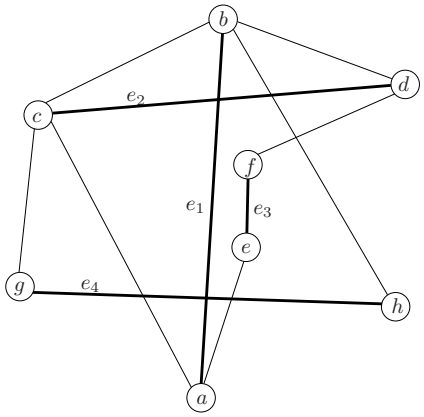


Figure 2.3: Original network

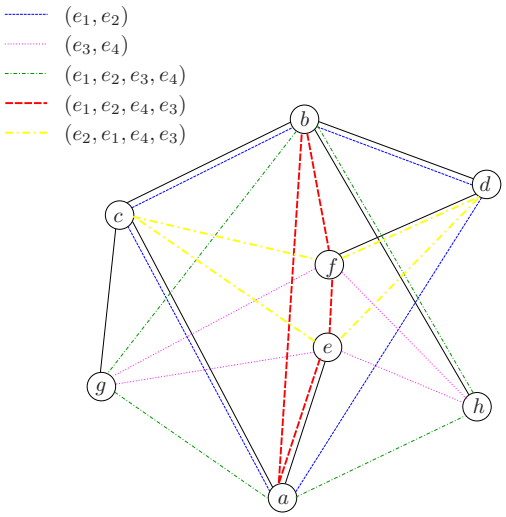


Figure 2.4: Modified network

Computational Results of Phase I

Recall from Section 2.4 that in phase I the LP-relaxation of the one-index formulation (2.7)–(2.10) is solved. The results obtained at the end of this phase are presented in the Tables 2.4–2.7. We use the following notation:

lb_{BB}	cutting-plane algorithm of Belenguer and Benavent (2003)
lb_A	cutting-plane algorithm of Ahr (2004)
lb_L	linear relaxation (root) by Longo *et al.* (2006)
lb_{LO}	linear relaxation (root with elementary routes) by Letchford and Oukil (2009)
lb_{MPS}	dual ascent and cutting-plane method of Martinelli *et al.* (2013)
lb_{own}	lower bound obtained in phase I
init cuts	number of initial cuts
component	number of odd cuts and capacity cuts separated on components
odd cuts	number of separated odd outs/number of binding odd cuts
cap cuts	number of separated/binding capacity cuts
dp1	number of separated/binding dp1 cuts
dp2	number of separated/binding dp2 cuts
comp. time	in time seconds of phase I

The meaning of all entries in the tables and the way we produced the results is explained in the following:

We initialize the one-index formulation with all odd cut constraints $y(S) \geq 1$ for singleton sets $S = \{v\}$ where $v \in V$ is an odd node. Additionally, odd cut and capacity cut constraints $y(S) \geq k(S) - |\delta_R(S)|$ are added to the model with the initialization procedure described in (Belenguer and Benavent, 2003). The number of all these inequalities is shown in column "init cuts".

We start every iteration of the separation procedure with a fast separation heuristic that considers the components of the support graph $(V, E_R \cup \{e \in E : y_e > 0\})$ in order to generate candidate sets S. Associated violated odd cuts or capacity cuts are added first. Their number is shown in column "component".

If no cut has been found so far, the efficient separation routine of Letchford *et al.* (2008) is used to separate odd cuts exactly. Moreover, capacity cuts are then separated in a heuristic way with the procedure of Belenguer and Benavent (2003) by solving a maximum-flow problem both on an auxiliary graph and perturbed variants of that auxiliary graph (the edge demand q_e is perturbed). This procedure is guaranteed to find all violated *fractional* capacity cuts. Those sets S that do not violate the fractional cut $y(S) \geq q(E_R(S) \cup \delta_R(S))/Q - \delta_R(S)$ are checked to violate $y(S) \geq k(S) - |\delta_R(S)|$.

If no cut is found, the algorithm of Ahr (2004) is then used to exactly separate capacity cuts.

The separation of disjoint-path inequalities is the most time consuming component. After analyzing the computation times, we decided to start with the separation of disjoint-path inequalities of type dp2. We proceed with the separation procedures for disjoint-path inequalities of type dp1 followed by dp3.

In all separation procedures, we use a threshold of $\varepsilon = 0.01$ for minimum violation,

i.e., a cut is only violated if its rhs exceeds the lhs by at least ε. The only exception from this rule is the MIP-based separation algorithm for capacity cuts of Ahr (2004), where the threshold is set to $\varepsilon = 0.1$.

Using this cascade of separation procedures, we never found violated cuts of type dp3 (last procedure in the cascade). Therefore, the Tables 2.4–2.7 do not report results on disjoint-path inequalities of type dp3. However, when we change the order so that dp3 cuts were considered before dp1 and dp2, we find at least some violated inequalities of type dp3.

The number of separated cuts and the number of cuts used to initialize the eMP (phase II) are shown in the columns "odd cuts", "capacity cuts", "dp1", and "dp2", respectively. The last column presents the computation time of phase I.

Next, we compare our results with the results from the literature. For the `kshs` and `gdb` instances, we obtained at least the lower bounds lb_A and lb_{BB} as presented in (Ahr, 2004; Belenguer and Benavent, 2003). In the case of `gdb8`, we are able to increase the lower bound (this is possible as Belenguer and Benavent (2003) do not use an exact separation procedure for capacity cuts and Ahr (2004) did not separate disjoint-path inequalities). For `gdb8` and `gdb12`, one binding dp1 inequality was separated (but non of type dp2). The computation times for both sets of instances are all small (sometimes too small to be recorded properly; note that all times are rounded up with precision 0.1).

For the `bccm` instances, the lower bounds lb_{own} is always at least as strong as lb_A. This is consistent as we use the same MIP approach as Ahr (2004) to exactly separate capacity cuts. Compared to (Belenguer and Benavent, 2003), in three cases (`2b`, `6b` and `9d`), the lower bounds lb_{BB} are two units better than our bound lb_{own}. We suspect that this is due to the fact that the separation of disjoint-path inequalities is performed with our version of the heuristic that certainly differs from the one used to compute lb_{BB}. For `10d`, we are able to increase the lower bound compared to lb_{BB} by one unit.

The lower bounds lb_L of Longo *et al.* (2006) obtained in the root node of their CVRP branch-and-price-and-cut algorithm are at least as good as our lower bounds obtained at the end of the cutting plane algorithm (11 cases better). The only exception is the instance `5b` were our bound is slightly better.

For the `egl` benchmark set, our lower bounds lb_{own} are at least as strong as the both lower bounds lb_A and lb_{BB} (except for `s2-a`). As for the `bccm` instances, the reported lower bound lb_L are always better than our lower bounds. The recent paper by Martinelli *et al.* (2013) reports computational results only for the `egl` instances. There, the lower bounds at the end of the cutting-plane algorithms are sometimes worse (9 cases) and sometimes better (9 cases) than our lower bounds. This might be due to the fact that they do not separate disjoint-path inequalities, but might use smaller thresholds $\varepsilon > 0$ for violation.

Concerning the computation time presented in Tables 2.6 and 2.7, our implementation could further be accelerated. The work of Martinelli *et al.* (2013) shows that a warm-start of the cutting-plane algorithm can further reduce computation times. They use a dual-ascent heuristic for the warm-start. However, even without such a warm-start the computation times for phase I are typically much smaller than those of phase II.

name	$\|V\|$	$\|E\|$	$\|K\|$	lb_{BB}	lb_A	lb_{own}	init cuts	component	odd cuts	cap cuts	dp1	dp2	time
kshs1	8	15	4	14661	14661	14661	5	3	0/4	1/2	0/0	0/0	0.1
kshs2	10	15	4	9863	9863	9863	9	3	0/9	1/2	0/0	0/0	0.2
kshs3	6	15	4	9320	9320	9320	7	0	0/6	0/1	0/0	0/0	0.2
kshs4	8	15	4	11098	11098	11098	5	0	0/3	1/2	0/0	0/0	0.1
kshs5	8	15	3	10957	10957	10957	5	1	0/5	0/1	0/0	0/0	0.1
kshs6	9	15	3	10197	10197	10197	5	2	0/6	0/1	0/0	0/0	0.2

Table 2.4: Results for the `kshs` instances at the end of phase I.

name	$\|V\|$	$\|E\|$	$\|K\|$	lb_{BB}	lb_A	lb_{own}	init cuts	component	odd cuts	cap cuts	dp1	dp2	time
gdb1	12	22	5	316	316	316	7	8	0/11	1/2	0/0	0/0	0.2
gdb2	12	26	6	339	339	339	6	4	0/7	0/1	0/0	0/0	0.3
gdb3	12	22	5	275	275	275	8	2	0/8	0/1	0/0	0/0	0.2
gdb4	11	19	4	287	287	287	9	0	0/8	1/2	0/0	0/0	0.2
gdb5	13	26	6	377	377	377	7	7	0/11	2/3	0/0	0/0	0.2
gdb6	12	22	5	298	298	298	6	2	0/6	1/2	0/0	0/0	0.3
gdb7	12	22	5	325	325	325	7	10	0/15	1/2	0/0	0/0	0.1
gdb8	27	46	10	344	344	346	19	17	1/21	25/14	1/1	0/0	1.8
gdb9	27	51	10	303	303	303	17	15	0/21	14/10	0/0	0/0	0.8
gdb10	12	25	4	275	275	275	8	3	0/10	0/0	0/0	0/0	0.4
gdb11	22	45	5	395	395	395	18	5	0/20	0/1	0/0	0/0	0.4
gdb12	13	23	7	450	450	450	8	3	0/5	2/3	1/1	0/0	0.4
gdb13	10	28	6	536	536	536	6	2	0/6	0/1	0/0	0/0	0.4
gdb14	7	21	5	100	100	100	1	0	0/0	0/1	0/0	0/0	0.1
gdb15	7	21	4	58	58	58	1	0	0/0	0/1	0/0	0/0	0.1
gdb16	8	28	5	127	127	127	9	0	0/7	0/1	0/0	0/0	0.4
gdb17	8	28	5	91	91	91	9	0	0/8	0/0	0/0	0/0	0.5
gdb18	9	36	5	164	164	164	1	0	0/0	0/1	0/0	0/0	0.5
gdb19	8	11	3	55	55	55	5	1	0/5	0/1	0/0	0/0	0.1
gdb20	11	22	4	121	121	121	5	3	0/6	0/1	0/0	0/0	0.2
gdb21	11	33	6	156	156	156	5	1	0/5	0/1	0/0	0/0	0.5
gdb22	11	44	8	200	200	200	5	1	0/5	0/1	0/0	0/0	1.0
gdb23	11	55	10	233	233	233	1	0	0/0	1/2	0/0	0/0	1.6

Table 2.5: Results for the `gdb` instances at the end of phase I.

name	$\|V\|$	$\|E\|$	$\|K\|$	lb_{BB}	lb_A	lb_L	lb_{own}	init cuts	component	odd cuts	cap cuts	dp1	dp2	time
1a	24	39	2	247	247	247	247	16	16	0/31	0/0	0/0	0/0	0.1
1b	24	39	3	247	247	247	247	16	16	0/31	0/0	0/0	0/0	0.2
1c	24	39	8	309	309	312	309	16	15	0/23	14/13	0/0	0/0	0.5
2a	24	34	2	298	298	298	298	17	4	0/19	2/3	0/0	0/0	0.1
2b	24	34	3	330	328	329	328	17	6	0/15	3/6	0/0	0/0	0.1
2c	24	34	8	526	526	528	526	17	11	0/9	13/14	0/0	0/0	0.4
3a	24	35	2	105	105	105	105	16	9	0/21	1/2	0/0	0/0	0.1
3b	24	35	3	111	111	111	111	16	10	0/19	2/5	0/0	0/0	0.3
3c	24	35	7	161	159	161	161	17	16	0/7	21/19	0/0	3/3	0.7
4a	41	69	3	522	522	522	522	24	11	0/29	3/4	0/0	0/0	0.3
4b	41	69	4	534	534	534	534	24	11	0/28	3/4	0/0	0/0	0.5
4c	41	69	5	550	550	550	550	24	12	0/28	6/7	0/0	0/0	0.5
4d	41	69	9	644	642	648	644	24	19	6/30	57/25	3/2	1/1	8.0
5a	34	65	3	566	566	566	566	19	26	0/43	1/2	0/0	0/0	0.3
5b	34	65	4	589	586	588	589	20	33	0/47	3/4	0/0	1/1	0.5
5c	34	65	5	612	610	613	612	20	33	0/46	6/6	0/0	1/1	0.8
5d	34	65	9	714	714	716	714	21	38	1/37	11/11	0/0	0/0	0.9
6a	31	50	3	330	330	330	330	19	27	0/41	0/1	0/0	0/0	0.1
6b	31	50	4	338	336	337	336	19	27	0/38	1/2	0/0	0/0	0.3
6c	31	50	10	418	414	420	418	19	26	1/22	38/20	1/0	1/1	2.0
7a	40	66	3	382	382	382	382	21	7	0/26	0/0	0/0	0/0	0.1
7b	40	66	4	386	386	386	386	21	7	0/26	1/1	0/0	0/0	0.3
7c	40	66	9	436	430	436	436	22	6	0/21	27/21	2/2	1/0	3.1
8a	30	63	3	522	522	522	522	18	6	0/22	0/1	0/0	0/0	0.1
8b	30	63	4	531	531	531	531	18	6	0/21	2/3	0/0	0/0	0.4
8c	30	63	9	653	645	654	653	20	13	0/17	20/15	2/2	6/6	2.1
9a	50	92	3	450	450	450	450	32	34	0/64	0/0	0/0	0/0	0.4
9b	50	92	4	453	453	453	453	32	21	0/50	0/1	0/0	0/0	0.3
9c	50	92	5	459	459	459	459	32	21	0/49	0/1	1/1	0/0	1.1
9d	50	92	10	509	505	512	507	32	26	0/50	15/14	1/1	1/1	3.0
10a	50	97	3	637	637	637	637	32	18	0/48	0/1	0/0	0/0	0.2
10b	50	97	4	645	645	645	645	32	41	15/83	1/2	0/0	0/0	1.1
10c	50	97	5	655	655	655	655	32	51	0/77	3/4	0/0	0/0	0.8
10d	50	97	10	732	731	734	733	33	35	38/64	38/15	4/2	6/4	104.0

Table 2.6: Results for the `bccm` instances at the end of phase I.

name	$\|V\|$	$\|E\|$	$\|K\|$	lb_{BB}	lb_A	lb_L	lb_{MPS}	lb_{own}	init cuts	component	odd cuts	cap cuts	dp1	dp2	time
e1-a	77	98/51	5	3515	3516	3548	3527	3545	41	65	133/146	317/152	7/5	0/0	49
e1-b	77	98/51	7	4436	4436	4468	4468	4464	41	73	62/49	396/136	0/0	0/0	103
e1-c	77	98/51	10	5453	5481	5542	5513	5515	41	110	68/64	365/216	2/2	0/0	146
e2-a	77	98/72	7	4994	4963	5011	4995	4995	55	87	35/112	157/87	0/0	0/0	25
e2-b	77	98/72	10	6249	6271	6280	6273	6271	56	125	72/67	191/104	0/0	0/0	57
e2-c	77	98/72	14	8114	8155	8234	8165	8163	56	117	19/53	243/106	4/4	0/0	119
e3-a	77	98/87	8	5869	5866	5898	5898	5894	67	93	6/77	117/52	0/0	0/0	34
e3-b	77	98/87	12	7646	7649	7697	8649	7677	67	88	4/60	184/88	0/0	1/1	51
e3-c	77	98/87	17	10019	10119	10163	10138	10125	68	123	14/42	193/75	0/0	0/0	103
e4-a	77	98/98	9	6372	6378	6395	6378	6378	60	75	10/87	73/56	0/0	0/0	11
e4-b	77	98/98	14	8809	8838	8884	8838	8838	61	107	7/59	141/72	0/0	0/0	19
e4-c	77	98/98	19	11276	11376	11427	11383	11382	62	160	28/43	139/68	1/1	0/0	257
s1-a	140	190/75	7	4992	4975	5014	5010	5010	49	325	420/284	542/353	0/0	0/0	208
s1-b	140	190/75	10	6201	6180	6379	6368	6368	50	422	290/253	640/482	0/0	0/0	454
s1-c	140	190/75	14	8310	8286	8480	8404	8404	51	517	165/168	671/600	0/0	0/0	605
s2-a	140	190/147	14	9780	9718	9824	9737	9758	106	217	39/116	409/204	0/0	12/7	283
s2-b	140	190/147	20	12286	12835	12968	12901	12914	109	427	47/85	465/115	7/5	1/1	1136
s2-c	140	190/147	27	16221	16216	16353	16274	16247	109	493	33/95	559/224	0/0	0/0	2155
s3-a	140	190/159	15	10025	9991	10143	10083	10119	107	260	70/105	375/129	10/7	7/6	928
s3-b	140	190/159	22	13554	13520	13616	13568	13576	110	505	82/94	405/156	16/4	4/0	1291
s3-c	140	190/159	29	16969	16958	17100	17019	17007	110	459	32/84	426/141	1/1	1/0	2751
s4-a	140	190/190	19	12027	12007	12143	12026	12052	111	120	38/94	283/111	8/4	10/9	477
s4-b	140	190/190	27	15933	15897	16093	16001	16016	112	132	23/84	474/112	2/2	1/1	1003
s4-c	140	190/190	35	20179	20176	20375	20256	20235	112	158	16/70	580/175	0/0	0/0	1696

Table 2.7: Results for the `egl` instances at the end of phase I.

Computational Results of Phase II

This section presents the results on the linear relaxation of the master program eMP (phase II) for the small-sized benchmark sets `kshs` and `gdb`. We use the same notation as in the journal article:

lb_*	lower bound obtained by algorithm of *; BB= cutting-plane algorithm of Belenguer and Benavent (2003); A= Ahr (2004); L= linear relaxation (root) by Longo *et al.* (2006); LO= linear relaxation (root, elementary routes) by Letchford and Oukil (2009); MPS= dual ascent and cutting-plane algorithm of Martinelli *et al.* (2013)
lb_{own}^{root} / lb_{own}^{tree}	lower bound obtained in our linear relaxation (root) or at termination of phase II (tree)
ub_{best}	best known upper bound
comp. by	first paper where the currently best upper bound was computed; H= Hertz *et al.* (2000); La= Lacomme *et al.* (2001); BE= Brandão and Eglese (2008); PD= Polacek *et al.* (2008); SCC= Santos *et al.* (2010); BLC= Bartolini *et al.* (2013b); Be= Beullens *et al.* (2003)
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= Benavent *et al.* (1992); BB= Belenguer and Benavent (2003); Lo= Longo *et al.* (2006); BM= Baldacci and Maniezzo (2006); LO= Letchford and Oukil (2009); own= own solution
comp. time	computation in time seconds for phase I or phase II
B&B nodes	number of solved nodes in our branch-and-bound tree
branching D/E/F	number of different branching decisions: D= node degree; E= edge flow; F= follower

name	$\|V\|$	$\|E\|$	$\|K\|$	lb_{BB}	lb_A	lb_{own}^{root}	*opt*	proved by lb	proved by sol	time phase I	time phase II	*B&B* nodes	branching D/E/F
kshs1	8	15	4	14661	14661	14.661	14661	BB	Lo	0.1	0.1	2	0/0/1
kshs2	10	15	4	9863	9863	9.863	9863	BB	Lo	0.2	0.1	2	0/0/1
kshs3	6	15	4	9320	9320	9.320	9320	BB	Lo	0.2	0.1	3	0/0/2
kshs4	8	15	4	11098	11098	11.498	11498	Lo	Lo	0.1	0.1	1	0/0/0
kshs5	8	15	3	10957	10957	10.957	10957	BB	Lo	0.1	0.1	2	0/0/1
kshs6	9	15	3	10197	10197	10.197	10197	BB	Lo	0.2	0.1	2	0/0/1

Table 2.8: Results for the `kshs` instances at the end of phase II.

name	$\|V\|$	$\|E\|$	$\|K\|$	lb_{BB}	lb_A	lb^{root}_{own}	opt	proved by lb	proved by sol	time phase I	time phase II	$B\&B$ nodes	branching D/E/F
gdb1	12	22	5	316	316	316	316	BB	Lo	0.2	0.1	9	0/0/8
gdb2	12	26	6	339	339	339	339	BB	Lo	0.3	0.1	6	0/0/5
gdb3	12	22	5	275	275	275	275	BB	Lo	0.2	0.1	7	0/0/6
gdb4	11	19	4	287	287	287	287	BB	Lo	0.2	0.1	3	0/0/2
gdb5	13	26	6	377	377	377	377	BB	Lo	0.2	0.1	7	0/0/6
gdb6	12	22	5	298	298	298	298	BB	Lo	0.3	0.1	5	0/0/4
gdb7	12	22	5	325	325	325	325	BB	Lo	0.1	0.1	9	0/0/8
gdb8	27	46	10	344	344	347	348	Lo	Lo	1.8	0.8	12	5/0/6
gdb9	27	51	10	303	303	303	303	BB	Lo	0.8	1.3	11	6/0/4
gdb10	12	25	4	275	275	275	275	BB	Lo	0.4	0.1	8	0/0/7
gdb11	22	45	5	395	395	395	395	BB	Lo	0.4	1.0	21	0/0/20
gdb12	13	23	7	450	450	451	456	Lo	Lo	0.4	0.2	15	12/0/2
gdb13	10	28	6	536	536	536	536	BB	Lo	0.4	0.2	4	0/0/3
gdb14	7	21	5	100	100	100	100	BB	Lo	0.1	0.1	3	0/0/2
gdb15	7	21	4	58	58	58	58	BB	Lo	0.1	0.1	6	1/0/4
gdb16	8	28	5	127	127	127	127	BB	Lo	0.4	0.2	14	2/0/11
gdb17	8	28	5	91	91	91	91	BB	Lo	0.5	0.2	20	1/0/18
gdb18	9	36	5	164	164	164	164	BB	Lo	0.5	0.2	17	0/0/16
gdb19	8	11	3	55	55	55	55	BB	Lo	0.1	0.1	1	0/0/0
gdb20	11	22	4	121	121	121	121	BB	Lo	0.2	0.2	6	0/0/5
gdb21	11	33	6	156	156	156	156	BB	Lo	0.5	0.2	12	2/0/9
gdb22	11	44	8	200	200	200	200	BB	Lo	1.0	0.4	22	3/0/18
gdb23	11	55	10	233	233	233	233	BB	Lo	1.6	0.6	23	3/0/19

Table 2.9: Results for the `gdb` instances at the end of phase II.

Best Known Lower and Upper Bounds for the `egl` Instances

The following table summarizes the best known lower and upper bounds for the `egl` instances and list the corresponding references:

name	lb_{best}	comp. by	ub_{best}	comp. by	*opt*	proved by
e1-a	-	-	3548	Lacomme *et al.* (2001)	3548	Longo *et al.* (2006)
e1-b	-	-	4498	Lacomme *et al.* (2001)	4498	Baldacci and Maniezzo (2006)
e1-c	-	-	5595	Lacomme *et al.* (2001)	5595	Bartolini *et al.* (2013b)
e2-a	-	-	5018	Lacomme *et al.* (2001)	5018	Baldacci and Maniezzo (2006)
e2-b	6301	own	6317	Brandão and Eglese (2008)	-	-
e2-c	-	-	8335	Brandão and Eglese (2008)	8335	Bartolini *et al.* (2013b)
e3-a	-	-	5898	Lacomme *et al.* (2001)	5898	Longo *et al.* (2006)
e3-b	7728	own	7775	Polacek *et al.* (2008)	-	-
e3-c	10244	Bartolini *et al.* (2013b)	10292	Polacek *et al.* (2008)	-	-
e4-a	6408	own	6444	Santos *et al.* (2010)	-	-
e4-b	8935	Bartolini *et al.* (2013b)	8961	Bartolini *et al.* (2013b)	-	-
e4-c	11493	Bartolini *et al.* (2013b)	11562	Bartolini *et al.* (2013b)	-	-
s1-a	-	-	5018	Lacomme *et al.* (2001)	5018	Baldacci and Maniezzo (2006)
s1-b	-	-	6388	Brandão and Eglese (2008)	6388	Bartolini *et al.* (2013b)
s1-c	-	-	8518	Lacomme *et al.* (2001)	8518	Bartolini *et al.* (2013b)
s2-a	9825	Bartolini *et al.* (2013b)	9884	Santos *et al.* (2010)	-	-
s2-b	13017	Bartolini *et al.* (2013b)	13100	Brandão and Eglese (2008)	-	-
s2-c	-	-	16425	Brandão and Eglese (2008)	16425	Bartolini *et al.* (2013b)
s3-a	10160	own	10220	Santos *et al.* (2010)	-	-
s3-b	13648	Bartolini *et al.* (2013b)	13682	Polacek *et al.* (2008)	-	-
s3-c	-	-	17188	Bartolini *et al.* (2013b)	17188	Bartolini *et al.* (2013b)
s4-a	12149	own	12268	Santos *et al.* (2010)	-	-
s4-b	16105	own	16321	Polacek *et al.* (2008)	-	-
s4-c	20430	Bartolini *et al.* (2013b)	20481	Bartolini *et al.* (2013b)	-	-

Table 2.10: Bounds for the `egl` instances.

Optimal solutions for the `bccm` instances:

Optimal solutions for the `bccm` instances that have not been presented before are the following:

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
5c	$z = 617$	
	veh 1	1=2-3=9=15=21-22=23=28=27=26=31=30=29=18=12=6-1
	veh 2	1=6=18=19=25=29-30=25=26=20=14=8-14-13=2-1
	veh 3	1=7-13=19=20=21=27=32=31-32=33=34=28=22=15-14=13=12-6-1
	veh 4	1-2=3=10-3=4=10=16=17=11=5=4=11=10=9=8=2-1
	veh 5	1-6=7=13-14=15=16=23=24=11-17=24=34-33=28-22=21-20-19=12-6-1
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

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

2.10 Bibliography

Ahr, D. (2004). *Contributions to Multiple Postmen Problems.* Phd dissertation, Department of Computer Science, Heidelberg University, Heidelberg, Germany.

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

Bartolini, E., Cordeau, J.-F., and Laporte, G. (2013b). Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming*, **137**(1-2), 409–452.

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

Beullens, P., Muyldermans, L., Cattrysse, D., and van Oudheusden, D. (2003). A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, **147**(3), 629–643.

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

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

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.

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.

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

Longo, H., Poggi 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.

Martinelli, R., Poggi de Aragão, M., and Subramanian, A. (2013). Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, **40**(8), 2145–2160.

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

Polacek, M., Doerner, K., Hartl, R., and Maniezzo, V. (2008). A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, **14**, 405–423.

Santos, L., Coutinho-Rodrigues, J., and Current, J. R. (2010). An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B*, **44**(2), 246 – 266.

Chapter 3

The Shortest-Path Problem with Resource Constraints with $(k, 2)$-Loop Elimination and Its Application to the Capacitated Arc-Routing Problem

Claudia Bode, Stefan Irnich

Abstract

In many branch-and-price algorithms, the column generation subproblem consists of computing feasible constrained paths. In the capacitated arc-routing problem (CARP), elementarity constraints concerning the edges to be serviced and additional constraints resulting from the branch-and-bound process together impose two types of loop-elimination constraints. To fulfill the former constraints, it is common practice to rely on a relaxation where loops are allowed. In a k-loop elimination approach all loops of length k and smaller are forbidden. Following Chapter 2 for solving the CARP, branching on followers and non-followers is the only known approach to guarantee integer solutions within branch-and-price. However, it comes at the cost of additional task-2-loop elimination constraints. In this Chapter, we show that a combined $(k, 2)$-loop elimination in the shortest-path subproblem can be accomplished in a computationally efficient way. Overall, the improved branch-and-price often allows the computation of tighter lower bounds and integer optimal solutions for several instances from standard benchmark sets.

3.1 Introduction

In this Chapter, we extend the works of Irnich and Villeneuve (2006) and Chapter 2. The first paper considered k-cycle elimination for shortest-path problems with resource constraints (SPPRC, Irnich and Desaulniers, 2005). The elementary SPPRC (ESPPRC) is the subproblem of many column-generation formulations of routing problems. Since the ESPPRC is $\mathcal{NP}$-hard in the strong sense (Dror, 1994), early column-generation approaches solved the SPPRC, i.e., the corresponding non-elementary problem, or SPPRC with 2-cycle elimination (see, e.g. Houck *et al.*, 1980; Kohl *et al.*, 1999), which are both relaxations, with the consequence that the lower bounds computed by the column-generation master program often deteriorate. The elimination of k-cycles, i.e., cycles

with up to k edges, can be seen as a mean to gradually strengthen the linear relaxation of the column-generation master program while keeping the computational effort acceptable. Both from a practical and a worst-case point of view, k-cycle elimination is computationally attractive because there exist pseudo-polynomial labeling algorithms (Irnich and Villeneuve, 2006). Applied to the vehicle-routing problem with time windows some knowingly hard instances were solved for the first time. Nowadays, it seems that approaches based on solving ESPPRC (e.g. Jepsen *et al.*, 2008) or *ng*-path relaxations (Baldacci *et al.*, 2011b) are superior due to the extremely tight lower bounds produced. However, when routes become very long, solving even very few ESPPRC subproblems can become extremely time consuming (see e.g. Desaulniers *et al.*, 2008).

We apply loop-elimination for solving the capacitated arc-routing problem (CARP) with the branch-and-price algorithm of Chapter 2. The CARP is the basic multiple-vehicle arc-routing problem. It has applications in waste collection, postal delivery, winter services and more (Dror, 2000; Corberán and Prins, 2010). For a general overview on exact algorithms for the CARP we refer to the survey (Belenguer *et al.*, 2013). In Chapter 2, the first full-fledged branch-and-price algorithm for the CARP was presented. It can be characterized by the idea of exploiting sparsity of the underlying CARP network. The advantage of sparse networks is that new CARP tours can be priced out efficiently using the sparse network only (see also Letchford and Oukil, 2009). Chapter 2 discussed that the sparse network however comes at the cost of a more intricate branching and they developed an effective branching scheme based on follower and non-follower constraints. It means that for every two required edges e and e', one determines whether or not these two edges are serviced by the same vehicle and with no other intermediate service in between. Follower constraints, i.e., two edges e and e' have to be serviced consecutively, can be enforced by network modifications so that these constraints preserve the structure of the pricing subproblem. However, non-follower constraints, i.e., two edges e and e' must not be served consecutively, can only be implemented efficiently as a task-2-loop elimination constraint: The trick is to associate the same task to both edges e and e' (see also (Irnich and Desaulniers, 2005) for a discussion of tasks and task cycles). Hence, this branching scheme seemed limited to the case that an SPPRC subproblem with 2-loop elimination is employed (the term loop refers to cycles w.r.t. services edges; a 2-loop is the repetition of the same service). Moreover, Chapter 2 showed that pricing relaxations based on k-loop elimination can produce better column-generation lower bounds. However, a key question remained unclear: How can the branching scheme with branching on followers/non-followers be combined with k-loop elimination for $k \geq 3$.

A positive answer will be given here because we can show that a combined $(k, 2)$-cycle elimination accomplishes the above problem. More precisely, k-loops w.r.t. tasks associated with services on required edges and 2-loops w.r.t. tasks implied by non-follower constraints can be handled by a labeling approach presented here. Note that Chapter 4 studies several alternative relaxations including, e.g., extensions of the afore-mentioned *ng*-path relaxations. Its focus is the empirical comparison of these relaxations, the analysis of the impact of possible acceleration techniques, and the overall comparison of the different branch-and-price algorithms.

The contribution of this Chapter is therefore twofold: First, we provide the theoretical foundation of a labeling algorithm allowing the combined $(k, 2)$-loop elimination. This includes the definition of labels, the derivation of an effective dominance procedure, and a worst-case analysis. Second, we run the different branch-and-price algorithms for the CARP resulting from choosing different values of k, i.e., for $k = 2, 3$, and 4. It will be apparent that the controlled variation of k is beneficial when it comes to a comparison concerning the trade-off between the strength of a column-generation formulation and the computational burden for its resolution.

We expect that the branch-and-price-based approach, presented in this Chapter here, applies also for solving other vehicle routing problems (VRP) defined on a sparse networks: Examples are the general routing problem (Pandi and Muralidharan, 1995, typically defined on a sparse graph)) and some variants of the VRP defined on street networks (e.g., Nasiri and Letchford, 2013).

The remainder of this Chapter is structured as follows. Section 3.2 presents the new labeling algorithm and its theoretical analysis for SPPRC with combined $(k, 2)$-loop elimination. Section 3.3 presents computational results for using the respective relaxations as subproblems in a branch-and-price for the CARP. The Chapter ends with final conclusions in Section 3.4.

3.2 Labeling Algorithm for SPPRC with combined $(k, 2)$-loop Elimination

A general discussion of SPPRC solution approaches including a detailed discussion of dynamic programming labeling algorithms, at the moment the generally best performing methods, can be found in (Irnich and Desaulniers, 2005). Solution approaches tailored to ESPPRC are presented in (Feillet *et al.*, 2004; Boland *et al.*, 2006; Righini and Salani, 2006, 2008).

3.2.1 Basic SPPRC and Generic Labeling Algorithm

The SPPRC is defined over a digraph $G = (V, A)$ with node set V and arc set A. A start or origin node $s \in V$ is given. For notational convenience we assume that G is simple so that arcs $(i, j) \in A$ can be uniquely identified by their end nodes i and j. The last node of a path P, i.e., its end node is denoted by $v(P) \in V$. The extension of a path $P = (s, \ldots, v(P))$ along an arc (i, j) requires $v(P) = i$ and results in a new path $P' = (P, j) = (s, \ldots, v(P) = i, j)$. Resource extension functions $f = (f_{ij})_{(i,j)\in A}$ (REFs) handle the update of resources accumulated/consumed along the path. Thus, if path $P = (s, \ldots, i = v(P))$ has associated resources $r(P)$ (generally a multi-dimensional vector) then its extension $P' = (P, j)$ along arc (i, j) has resources $f_{ij}(r(P))$. Finally, multi-dimensional intervals $[a_j, b_j]$ for all nodes $j \in V$ are given.

In standard SPPRC, the problem is to compute a minimum-cost feasible path ending at each destination node $t \in V$. In case of non-decreasing REFs, a path $P' = (P, j)$ is *feasible* (w.r.t. resources) if $r(P') \in [a_{v(P')}, b_{v(P')}]$ holds and the predecessor path P is

also feasible. Then, for solving SPPRC, one typically computes, for each node, a set of paths $\{P_1, \ldots, P_q\}$ with $\{r(P_1), \ldots, r(P_q)\}$ forming the Pareto-optimal resource values. The particular importance of non-decreasing REFs is stressed in (Desaulniers *et al.*, 1998; Irnich and Desaulniers, 2005; Irnich, 2008).

In the CARP pricing subproblem, the network $G = (V, A)$ consists of the node set V defined by the CARP instance. The arc set A contains two types of arcs:

- Service arcs (i, j) and (j, i) correspond to proving service to a required edge $\{i, j\}$. These arcs consume a positive amount $q_{ij} > 0$ of the vehicle's capacity and have a (reduced) cost $\tilde{c}_{ij}^{serv}$ (not restricted in sign).

- Deadheading arcs (i, j) and (j, i) correspond to traversing an arbitrary edge $\{i, j\}$ without providing service. These arcs consume nothing from the vehicle's capacity, i.e., $q_{ij} = 0$. Even with valid inequalities present in the column-generation master program, their (reduced) cost $\tilde{c}_{ij}$ can be guaranteed to be non-negative (for details see Chapter 2).

Chapter 2 explained how the column-generation restricted master program (RMP) provides with its dual solution the reduced costs $\tilde{c}_{ij}^{serv}$ and $\tilde{c}_{ij}$. Summing up, the resources $r(P)$ in the CARP case consumed along a path P are given by $r(P) = (q(P), \tilde{c}(P))$, where $q(P)$ is restricted to integer values $0, 1, \ldots, C$ (C is the capacity of the vehicle) and $\tilde{c}(P)$ is the accumulated reduced costs.

The outline of a generic labeling approach for solving SPPRC is presented in Algorithm 3. Herein, each path P is represented by a label $L(P)$, i.e., a data structure that allows the reconstruction of the associated path via labels of the predecessor path, provides additional information such as the resource consumption $r(P)$, and allows to invoke a dominance algorithm. The set $\mathcal{U}$ is the set of unprocessed labels $L(P)$, i.e., paths P that are not extended along all arcs $(v(P), j) \in A$ of the forward star of node $v(P)$. The set $\mathcal{L}$ contains those labels that need to be kept.

Depending on the path selection rule in the path extension step, different label processing procedures result such as label setting and label correcting algorithms. The invocation of a dominance algorithm is optional in the sense that otherwise the algorithm enumerates all feasible paths starting at node s. Dominance is however crucial in the design of *efficient* labeling algorithms, and we devote Section 3.2.3 for the detailed presentation of this basic component. In general, the intension of the dominance algorithm is to identify those paths that do not necessarily be extended, i.e., one or several other paths still allow finding (Pareto-)optimal paths. It can be applied at any time in the course of the algorithm and might be delayed to a point where several new paths with identical end node have been generated and stored in $\mathcal{U}$. Any reasonable strategy for invoking the dominance algorithm will optimize the tradeoff between the computational effort and the risk that a path is extended before one finds out that it is dominated.

In the presence of additional path-structural constraints (such as cycle- or loop-elimination constraints discussed in Section 3.2.2, see also Section 2.2 of (Irnich and Desaulniers, 2005)), the set $\mathcal{L}$ must generally include additional labels for paths that are not necessarily

Algorithm 3: Generic SPPRC Dynamic Programming Labeling Algorithm

SET $\mathcal{U} := \{L(s)\}$, $\mathcal{L} := \emptyset$
while $\mathcal{U} \neq \emptyset$ **do**
  `// Path Extension Step`
  SELECT $L(P) \in \mathcal{U}$, REMOVE $L(P)$ from $\mathcal{U}$, and ADD $L(P)$ to $\mathcal{L}$
  for $(v(P), j) \in A$ **do**
    if *path* (P, j) *is feasible* **then**
      ADD $L(P, j)$ to $\mathcal{U}$
  `// Dominance Step`
  if */* any condition */* **then**
    APPLY dominance algorithm to labels $\mathcal{U} \cup \mathcal{L}$
`// Filtering Step`
IDENTIFY solutions $\mathcal{S} \subseteq \mathcal{L}$

Pareto-optimal. It this case, a final filtering step is needed to identify a Pareto-optimal subset. Efficient algorithms for that purpose can be found in (Bentley, 1980).

3.2.2 Task-Loops and Loop Elimination

In the column-generation context, a *task* is something (such as visiting a node, edge or arc) that needs to be performed by a column (a vehicle route or a schedule etc.). Generally, a task is associated with a set-partitioning or covering constraint of the master program, and it can be found at one or several nodes and arcs of the network. The work by Irnich and Villeneuve (2006) mainly addresses k-cycle elimination where every node of the subproblem's network represents an individual task (implied by standard node-elementarity constraints).

The Chapter at hand, however, addresses the more general case that an arbitrary sequence $\mathcal{T}_{ij}$ of tasks (including empty sequences) is associated with every arc $(i, j) \in A$. Then, feasible paths $P = (v_0, v_1, \ldots, v_p)$ are those where the joint task sequence $\mathcal{T}(P) := (\mathcal{T}_{v_0,v_1}, \mathcal{T}_{v_1v_2}, \ldots, \mathcal{T}_{v_{p-1}v_p})$ does not contain a task-k loop. It is important to highlight that the literature distinguishes between k-cycles and k-loops. The term k-cycle is traditionally used in the context of unique tasks associated with nodes. A 2-cycle is a cycle of length two such as (i, j, i), and a k-cycle is any cycle of length k or smaller. In contrast, a 2-loop is a repeated task (a, a) for any task $a \in \mathcal{T}$. A 2-loop can result from a subpath (i, j, i) where both arcs (i, j) and (j, i) (or an edge $\{i, j\}$ in the undirected case) have the task sequence $\mathcal{T}_{ij} = \mathcal{T}_{ji} = (a)$. However, the same task-2-loop results for (sub)path $P = (v_0, v_1, \ldots, v_p)$ if arc (v_0, v_1) has a task sequence $(\ldots, a)$ ending with task a, arcs $(v_1, v_2), \ldots, (v_{p-2}, v_{p-1})$ have an empty task sequence (also called deadheading), and arc (v_{p-1}, v_p) has a task sequence $(a, \ldots)$ starting with task a. In general, for $k > 2$ a task-k-loop is task-cycle of length $k - 1$ or smaller.

The rationale behind these seeming confusing definitions is that 2-cycle elimination and

task-2-loop elimination can be handled with almost identical algorithmic approaches: Dominance rules require that only a best and a second-best path with different last task need to be kept in $\mathcal{L}$ (see Algorihm 3). The dominance rules were first presented by (Houck *et al.*, 1980) for 2-cycle elimination in the node-routing context and by (Benavent *et al.*, 1992) for task-2-loop elimination and the CARP.

3.2.3 Dominance Rules in Combined (k_1, k_2)-loop Elimination

This section contains new theoretical results for labeling procedures that simultaneously consider two sets of tasks for which loop freeness must be guaranteed. In our CARP application, paths are desired to be k-loop-free w.r.t. tasks $\mathcal{T}^E$ induced by route's elementarity constraints. Here, we would like $k > 2$ to be as large as possible (of course there is the trade-off between strength of the relaxation and effort for pricing). Moreover, one needs paths to be exactly 2-loop-free w.r.t. the tasks $\mathcal{T}^B$ induced by non-follower constraints resulting from branching.

Generalizing, we will derive results for a combined (k_1, k_2)-loop elimination for the tasks sets $\mathcal{T}^1$ and $\mathcal{T}^2$. For simplicity, we abbreviate paths feasible w.r.t. both tasks sets $\mathcal{T}^1$ and $\mathcal{T}^2$ as (k_1, k_2)-loop-free paths. In particular, we suppress the prefix 'task-'.

It is rather simple to define attribute updates and extension rules for (k_1, k_2)-loop elimination. The crucial part for an effective labeling algorithm is however the definition of a dominance relation. Straightforward approaches allow dominance only between paths that have identical sequences of the last $k_1 - 1$ tasks of $\mathcal{T}^1$ and the last $k_2 - 1$ tasks of $\mathcal{T}^2$. This is rather easy, but turns out to be ineffective due a possible number of $\mathcal{O}\left(|\mathcal{T}^1|^{k_1-1} \cdot |\mathcal{T}^2|^{k_2-1}\right)$ labels at the same node and otherwise identical state (all resources except for cost; identical load in the CARP case). Irnich and Villeneuve (2006) discuss this point for node-k-cycle elimination in detail. Therefore, the decisive point is the development of effective dominance rules guaranteeing a small number of labels.

Such an effective dominance rule, based on the one for node-k-cycle elimination proposed by Irnich and Villeneuve (2006), does not only compare pairs of paths. Instead, several paths together may be needed to dominate another path. In the following, we will distinguish between paths and labels. Paths are represented by labels, but labels contain additional attributes needed to efficiently test for domination. Moreover, paths can mutually dominate each other, while we will make sure that dominance is uni-directional among labels. This can be achieved using a unique identifier (an ID) for each label, which breaks ties whenever two labels with identical resources are compared (for a more detailed discussion of that point see (Irnich and Villeneuve, 2006, p. 393f)).

The *dominance principle* says that labels $L(P_1), \ldots, L(P_g)$ $(g \geq 1)$ representing paths $P_1, \ldots, P_g$ dominate a label $L(P)$ representing path P if

1. $P_1, \ldots, P_g$ and P share the same end node $v(P_1) = \ldots = v(P_g) = v(P)$.
2. Every feasible completion Q of P, i.e., (P, Q) is a feasible path, must also result in a feasible path (P_j, Q) for at least one path P_j, $j \in \{1, \ldots, g\}$.
3. The cost of (P_j, Q) must not exceed the cost of (P, Q) for all $j \in \{1, \ldots, g\}$.

As a consequence, the label $L(P)$ does not need to be considered in a labeling algorithm because it can never produce a better feasible extension to the destination node than possible with at least one extension of the labels $L(P_1), \ldots, L(P_g)$. It is however crucial that the labels $L(P_1), \ldots, L(P_g)$ are kept.

The second condition (2.) is typically replaced by a (sufficient) condition that is easier to check, involving resource consumptions and task loops. In fact, all paths $P_1, \ldots, P_g$ must have resources not larger than the resources of P, i.e.,

$$r(P_1), \ldots, r(P_g) \leq r(P), \tag{3.1}$$

which is in the CARP case equivalent to $q(P_1), \ldots, q(P_g) \leq q(P)$ and $\tilde{c}(P_1), \ldots, \tilde{c}(P_g) \leq \tilde{c}(P)$, while feasibility regarding tasks loops is *not* checked via resources.

The fundamental idea for (k_1, k_2)-loop elimination is to efficiently encode the *set of possible extensions* of a path. For this purpose, let $\mathcal{E}(P)$ denote the set of loop-free extensions of the path P. $\mathcal{E}(P)$ solely considers task loops and not resource consumptions. The second condition (2.) above is fulfilled for $P_1, \ldots, P_g$ and P if (3.1) and

$$\bigcup_{i=1}^{g} \mathcal{E}(P_i) \supseteq \mathcal{E}(P) \tag{3.2}$$

holds. We will now describe how to encode this condition in order to handle two sets of tasks efficiently.

Encoding the Possible Extensions by Self-Hole Sets Recall that there are two sets of tasks $\mathcal{T}^1$ and $\mathcal{T}^2$ for which loop freeness has to be ensured. Let $\mathcal{S}$ be the set of all (k_1, k_2)-loop-free paths, i.e., k_1-loop-free w.r.t. tasks in $\mathcal{T}^1$ and k_2-loop-free with respect to tasks in $\mathcal{T}^2$. Let $P, Q \in \mathcal{S}$ be two feasible paths, where the end node $v(P)$ of P is identical with the start node of Q. Then, the concatenation (P, Q) is also a path in $\mathcal{S}$ if and only if both $(\mathcal{T}^1(P), \mathcal{T}^1(Q))$ is k_1-loop-free and $(\mathcal{T}^2(P), \mathcal{T}^2(Q))$ is k_2-loop-free. This condition holds if

$$(\mathcal{T}^1(P), \mathcal{T}^1(Q)) = (\ldots, t^1_{k_1-1}, \ldots, t^1_2, t^1_1, s^1_1, s^1_2, \ldots, s^1_{k_1-1}, \ldots) \text{ fulfills } t^1_p \neq s^1_q \; \forall p + q \leq k_1$$

and

$$(\mathcal{T}^2(P), \mathcal{T}^2(Q)) = (\ldots, t^2_{k_2-1}, \ldots, t^2_2, t^2_1, s^2_1, s^2_2, \ldots, s^2_{k_2-1}, \ldots) \text{ fulfills } t^2_p \neq s^2_q \; \forall p + q \leq k_2.$$

The relevant entries of $\mathcal{T}^1(Q)$ and $\mathcal{T}^2(Q)$ are the first $k_1 - 1$ and $k_2 - 1$ entries, and we denote these by $\mathcal{T}^1_{k_1}(Q)$ and $\mathcal{T}^2_{k_2}(Q)$, respectively. We assume in the following that both sequences $\mathcal{T}^1_{k_1}(Q)$ and $\mathcal{T}^2_{k_2}(Q)$ always contain exactly $k_1 - 1$ and $k_2 - 1$ elements, respectively, where missing tasks are represented by a '$\cdot$'. (Here we remind the reader about the notation that for $h = 1$ or $h = 2$ the term $\mathcal{T}^h$ (without subscript) refers to the set of all tasks, $\mathcal{T}^h_{ij}$ is the task sequence associated to an arc (i, j), and $\mathcal{T}^h_k(Q)$ is the $(k-1)$-tuple describing the sequence of the first $k-1$ tasks in a path Q possibly extended with succeeding $\cdot$.)

We are able to express the above condition as

$$\mathcal{T}^1_{k_1}(Q) \neq (\cdot, \ldots, \cdot, t^1_{p,i}, \cdot, \ldots, \cdot) \text{ for all } p \text{ with } 1 \leq p + i \leq k_1$$

and

$$\mathcal{T}^2_{k_2}(Q) \neq (\cdot, \ldots, \cdot, t^2_{p,i}, \cdot, \ldots, \cdot) \text{ for all } p \text{ with } 1 \leq p + i \leq k_2,$$

where i refers to the ith position in the right-hand-side vector, and $t^1_{p,i}$ and $t^2_{p,i}$ have the value t^1_p and t^2_p, respectively. The last $k_1 - 1$ entries of $\mathcal{T}^1(P)$, i.e., t^1_p with $p \in \{1, \ldots, k_1\}$, and the last $k_2 - 1$ entries of $\mathcal{T}^2(P)$, i.e., t^2_p with $p \in \{1, \ldots, k_2\}$ have to be compared with $\mathcal{T}^1_{k_1}(Q)$ and $\mathcal{T}^2_{k_2}(Q)$, respectively. It follows that any extension Q of path P is infeasible if $\mathcal{T}^1_{k_1}(Q)$ or $\mathcal{T}^2_{k_2}(Q)$ matches with the respective tuple (still '$\cdot$' refers to an unspecified entry).

These infeasible extensions can be represented by *set forms*, a concept introduced first in (Irnich and Villeneuve, 2006): The tuples on the right hand side of the above inequality are in fact set forms. The finite union of such set forms defines the self-hole set $H(P)$ of a path P.

Example: For $(4, 2)$-loop elimination in the CARP context, i.e., $k_1 = 4$, $k_2 = 2$ and $\mathcal{T}^1 = \mathcal{T}^E$, $\mathcal{T}^2 = \mathcal{T}^B$, let path P have $\mathcal{T}^E(P) = (\ldots, a, b, c)$ and $\mathcal{T}^B(P) = (\ldots, \alpha)$. It means that the last three required edges serviced were the edges a, b, and c. In addition, we are in a branch of the branch-and-price search tree where a non-follower constraint is active, e.g., say for the edges c and f, imposing that they have the new identical task α assigned in order to prevent c and f being serviced consecutively.

Then, any extension Q produces a feasible path w.r.t. loop elimination if

$$(\mathcal{T}^E_4(Q), \mathcal{T}^B_2(Q)) \neq (\cdot, \cdot, \cdot)(\alpha), (a, \cdot, \cdot)(\cdot), (b, \cdot, \cdot)(\cdot), (\cdot, b, \cdot)(\cdot), (c, \cdot, \cdot)(\cdot), (\cdot, c, \cdot)(\cdot), (\cdot, \cdot, c)(\cdot).$$

Equivalently, the self-hole set of P is

$$H(P) = (\cdot, \cdot, \cdot)(\alpha) \cup (a, \cdot, \cdot)(\cdot) \cup (b, \cdot, \cdot)(\cdot) \cup (\cdot, b, \cdot)(\cdot) \cup (c, \cdot, \cdot)(\cdot) \cup (\cdot, c, \cdot)(\cdot) \cup (\cdot, \cdot, c)(\cdot),$$

where each set form encodes the set of task sequences matching the respective pattern.

For example, if a path Q_1 produces the task sequence $\mathcal{T}^E_4(Q_1) = (d, a, b)$ and $\mathcal{T}^B_2(Q_1) = (\beta)$ then there is no match with $H(P)$, and the extension (P, Q_1) is feasible w.r.t. loop elimination. In contrast, for a path Q_2 with task sequence $\mathcal{T}^E_4(Q_2) = (d, e, c)$ there is a match with $(\cdot, \cdot, c)(\cdot)$ so that (P, Q_2) is infeasible.

The representation of $H(P)$ as the union of set forms is quadratic in k_1 and k_2, i.e., up to $\frac{k_1(k_1-1)}{2} + \frac{k_2(k_2-1)}{2}$ different set forms are necessary to describe all infeasible extensions of path P.

Now we consider a dominance situation where (3.1) and (3.2) are fulfilled for dominating paths $P_1, \ldots, P_g$ and a dominated path P. By de Morgan's law, we get

$$\bigcup_{i=1}^{g} \mathcal{E}(P_i) \supseteq \mathcal{E}(P) \quad \Longleftrightarrow \quad \bigcap_{i=1}^{g} H(P_i) \subseteq H(P) \tag{3.3}$$

so that the condition (3.2) for loop-free extensions can be equivalently stated with the help of self-hole sets. The point is now that any intersection of the self-hole sets, resulting on the right hand side, can be calculated and represented as a union of set forms again.

Theorem 1 *Let $P_1, P_2, \ldots, P_g$ and P be different paths ending at the same node $v(P_1) = \cdots = v(P_g) = v(P)$ with $r(P_1), \ldots, r(P_g) \leq r(P)$, and (3.3) is fulfilled.*

Then path P is dominated, i.e., any feasible completion Q of P results in at least one feasible path (P_j, Q) (for one $j \in \{1, 2, \ldots, g\}$) with $r(P_j, Q) \leq r(P, Q)$. (Note: Feasibility refers to both being (k_1, k_2)-loop free and feasible w.r.t. resource constraints.)

Example:[Example 3.2.3 continued] Let P' be another path with $\mathcal{T}^E(P') = (a, d)$ (just two edges serviced along P') and $\mathcal{T}^B(P') = (\beta)$. The self-hole set of P' is

$$H(P') = (\cdot,\cdot,\cdot)(\beta) \cup (a,\cdot,\cdot)(\cdot) \cup (\cdot,a,\cdot)(\cdot) \cup (d,\cdot,\cdot)(\cdot) \cup (\cdot,d,\cdot)(\cdot) \cup (\cdot,\cdot,d)(\cdot)$$

Then, the intersection of the self-hole sets is

$$\begin{aligned} H(P) \cap H(P') = &(a,\cdot,\cdot)(\alpha) \cup (\cdot,a,\cdot)(\alpha) \cup (d,\cdot,\cdot)(\alpha) \cup (\cdot,d,\cdot)(\alpha) \cup (\cdot,\cdot,d)(\alpha) \cup \\ &(a,\cdot,\cdot)(\beta) \cup (b,\cdot,\cdot)(\beta) \cup (\cdot,b,\cdot)(\beta) \cup (c,\cdot,\cdot)(\beta) \cup (\cdot,c,\cdot)(\beta) \cup (\cdot,\cdot,c)(\beta) \cup \\ &(a,d,\cdot)(\cdot) \cup (a,\cdot,d)(\cdot) \cup (b,a,\cdot)(\cdot) \cup (b,d,\cdot)(\cdot) \cup (b,\cdot,d)(\cdot) \cup (a,b,\cdot)(\cdot) \cup (d,b,\cdot)(\cdot) \cup \\ &(\cdot,b,d)(\cdot) \cup (c,a,\cdot)(\cdot) \cup (c,d,\cdot)(\cdot) \cup (c,\cdot,d)(\cdot) \cup (a,c,\cdot)(\cdot) \cup (d,c,\cdot)(\cdot) \cup (\cdot,c,d)(\cdot) \cup \\ &(a,\cdot,c)(\cdot) \cup (\cdot,a,c)(\cdot) \cup (d,\cdot,c)(\cdot) \cup (\cdot,d,c)(\cdot) \end{aligned}$$

The computation of the intersection of two unions of set forms, as in the above example, requires two algorithmic components: First, set forms need to be tested for inclusion. For example, $(a,\cdot,b,e)(\alpha)$ is included in $(\cdot,\cdot,b,\cdot)(\alpha)$, while $(a,e,b)(\cdot)$ is *not* included in $(a,\cdot,c)(\cdot)$. It can be shown similarly as for node-k-cycle elimination that this test requires only $\mathcal{O}(k_1 + k_2)$ time and space (Irnich and Villeneuve, 2006, p. 398).

Second, proper intersections of set forms need to be computed. For two set forms s and t, the intersection $s \cap t$ is empty if different entries are specified at the same position. For example, $s = (a,b,\cdot)(\alpha)$ and $t = (a,c,b)(\alpha)$ result in $s \cap t = \emptyset$. Moreover, by definition, the intersection is empty if an infeasible loop is created, e.g., the intersection of $(a,b,\cdot)(\alpha)$ and $(\cdot,b,a)(\cdot)$ is empty because it induces the 3-loop (a,b,a) w.r.t. tasks $\mathcal{T}^1$. In contrast, the set forms $(a,b,\cdot)(\alpha,\cdot)$ and $(\cdot,b,d)(\cdot,\cdot)$ have non-empty intersection $(a,b,d)(\alpha,\cdot)$. Here again, the computation including loop detection requires only $\mathcal{O}(k_1+k_2)$ amortized time and space. As a result, the computation of the intersection of two self-hole sets, say with p and q set forms each, requires $\mathcal{O}((k_1 + k_2)pq)$ amortized time and space; see (Irnich and Villeneuve, 2006, p. 398) for details.

With the above definition of the intersection of two set form, we are able to formally define the intersection of two hole sets H_1 and H_2. Algorithm 4 is similarly stated in (Irnich and Villeneuve, 2006, p. 398). Note that the first loop is included for the purpose of accelerating the subsequent steps, i.e., to produce fewer set forms s and t in the preliminary result set H having s included in t or vice versa. Such included set forms are eliminated in the last loop. Overall complexity of Algorithm 4 $\mathcal{O}(\text{kpq})$, where p and q is the number of set forms in the respective hole set.

In order to know the overall time complexity, it is important to quantify the maximum number of elements present in an intersection of two collections of set forms. The next paragraph will give an answer.

Algorithm 4: Intersection of hole sets

INPUT: Two hole sets $H_1 := s^1 \cup \cdots \cup s^p$ and $H_2 := t^1 \cup \cdots \cup t^q$ encoded as two unions of set forms
SET $H := \emptyset$
`// Check for inclusion in input sets`
for $(s, t) \in H_1 \times H_2$ **do**
  if $s \subseteq t$ **then** SET $H := H \cup \{s\}$, $H_1 := H_1 \setminus \{s\}$
  else if $t \subseteq s$ **then** SET $H := H \cup \{t\}$, $H_2 := H_2 \setminus \{t\}$
`// Compute intersection`
for $(s, t) \in H_1 \times H_2$ **do**
  if $s \cap t \neq \emptyset$ **then** SET $H := H \cup \{s \cap t\}$
`// Check for inclusion in result set`
for $(s, t) \in H \times H, s \neq t$ **do**
  if $s \subseteq t$ **then** SET $H := H \setminus \{s\}$
  else if $t \subseteq s$ **then** SET $H := H \setminus \{t\}$
OUTPUT: H

Upper Bound on the Number of Set Forms in an Intersection of Self-Hole Sets
For node-k-cycle elimination, any collection of set forms resulting from the intersection of self-hole sets does not contain more than $(k-1)!^2$ different set forms. This result is stated in (Irnich and Villeneuve, 2006, p. 399) for node-k-cycle elimination. Notice that in node-k-cycle elimination all paths ending at the same node share an identical last task (corresponding to that node), which therefore can be omitted. Task-k-loop elimination, however, must ensure that there are at least $k-1$ other tasks before a task is repeated. Therefore, in both cases, recording only $k-1$ elements is sufficient to encode all relevant dominance information, which results in the stated complexity.

The result for combined (k_1, k_2)-loop elimination in SPPRC is the following:

Theorem 2 *For combined (k_1, k_2)-loop elimination, the maximum number of different set forms needed to represent any intersection of self-hole sets $H(P_1) \cap H(P_2) \cap \cdots \cap H(P_l)$ of any set of l paths is $\omega(k_1, k_2) := (k_1 - 1)!^2 \cdot (k_2 - 1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$. This bound $\omega(k_1, k_2)$ is tight.*

A proof of this theorem and all other theoretical results is included in the Appendix. The following example shows how to construct instances where the bound is indeed tight.

Example: Consider a combined $(3, 2)$-loop elimination. Moreover, let P_1, P_2, and P_3 be three paths with no tasks in common. Thus,

$$\begin{aligned} H(P_1) &= (\cdot,\cdot)(\alpha) \cup (a,\cdot)(\cdot) \cup (b,\cdot)(\cdot) \cup (\cdot, b)(\cdot) \\ H(P_2) &= (\cdot,\cdot)(\beta) \cup (c,\cdot)(\cdot) \cup (d,\cdot)(\cdot) \cup (\cdot, d)(\cdot) \end{aligned}$$

$$H(P_3) \quad = \quad (\cdot,\cdot)(\gamma) \cup (e,\cdot)(\cdot) \cup (f,\cdot)(\cdot) \cup (\cdot,f)(\cdot)$$

giving rise to

$$\begin{aligned} H(P_1) \cap H(P_2) \cap H(P_3) \quad = \quad & (a,d)(\gamma) \cup (b,d)(\gamma) \cup (c,b)(\gamma) \cup (d,b)(\gamma) \cup (c,f)(\alpha) \cup (d,f)(\alpha) \\ & \cup (e,d)(\alpha) \cup (f,d)(\alpha) \cup (a,f)(\beta) \cup (b,f)(\beta) \cup (e,b)(\beta) \cup (f,b)(\beta). \end{aligned}$$

These are twelve set forms which is the maximum number $(k_1-1)!^2 \cdot (k_2-1)!^2 \cdot \binom{k_1-1+k_2-1}{k_1-1} = (3-1)!^2 \cdot (2-1)!^2 \cdot \binom{(3-1)+(2-1)}{3-1} = 4 \cdot 1 \cdot 3 = 12$.

Upper Bound on the Number of Paths with Identical State The paragraph above presented results on the number of set forms in an intersection of an arbitrary number of paths. The question considered in this paragraph is about the maximum number of paths P with identical state (resource vector except for cost; for the CARP, with identical load $q(P)$). Let a collection of g paths $P_1, \ldots, P_g$ with identical state ending at a node $i = v(P_1) = \ldots = v(P_g)$ be given. The corresponding labels can be sorted in a unique way using the IDs of the labels so that the following ordering is given:

$$L(P_1) \prec_{dom} L(P_2) \prec_{dom} \ldots \prec_{dom} L(P_g),$$

To be precise, we define $L(P_1) \prec_{dom} L(P_2)$ so that both paths end at the same node $v(P_1) = v(P_2)$, P_1 dominates P_2 with respect to resource consumption, i.e., $r(P_1) \leq r(P_2)$, and in case of identical resources $r(P_1) = r(P_2)$ the IDs control that the relation $\prec_{dom}$ is antisymmetric, i.e., $L(P_1) \prec_{dom} L(P_2)$ implies $L(P_1) \not\prec_{dom} L(P_2)$. (For this reason we distinguish between paths and labels.) For the above paths $P_1, P_2, \ldots, P_g$ the dominance relation also means that, e.g., $L(P_g)$ is dominated by all other labels $L(P_1)$, $L(P_2), \ldots, L(P_{g-1})$. It follows for the intersections of the self-hole sets of the dominating labels ($L(P_1)$ dominates $L(P_2)$, $L(P_1)$ and $L(P_2)$ dominate $L(P_3)$ etc.) that

$$I_1 := H(P_1) \quad \supseteq \quad I_2 := H(P_1) \cap H(P_2) \quad \supseteq \quad \ldots \quad \supseteq \quad I_g := \bigcap_{i=1}^{g} H(P_i).$$

holds. Irnich and Villeneuve (2006) have shown that a path P_t can be discarded if $I_t = I_{t-1}$ holds. The reason is that $I_t = I_{t-1}$ implies $H(P_1) \cap \ldots \cap H(P_{t-1}) \subseteq H(P_t)$ so that conditions (3.3) hold. Therefore, the *maximum length of a properly decreasing chain of intersections of self-hole sets* is a bound on the maximum number of labels to consider with identical state.

Theorem 3 *A collection of g dominating paths $P_1 \prec_{dom} P_2 \prec_{dom} \ldots \prec_{dom} P_g$ ending at the same node is given. Let the intersections of the corresponding self-hole sets $H(P_1), H(P_2), \ldots, H(P_g)$ form a* properly *decreasing chain, i.e., $H(P_1) \supsetneq H(P_1) \cap H(P_2) \supsetneq \cdots \supsetneq \bigcap_{i=1}^{g} H(P_i)$. Then, the length g of the properly decreasing chain is bounded by $\gamma(k_1, k_2) = [k_1 + k_2 - 1] \cdot (k_1-1)!^2 \cdot (k_2-1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$.*

Note that the bound $\gamma(k_1, k_2)$ is generally not tight as already shown for node-k-cycle elimination (Irnich and Villeneuve, 2006, p. 400f).

For the special case of a combined $(k, 2)$-loop elimination, the bound is $\gamma(k, 2) = (k+1)\cdot(k-1)!^2\cdot k = (k-1)!\cdot(k+1)!$. In particular, we get the bounds $\gamma(3, 2) = 2\cdot 24 = 48$ and $\gamma(4, 2) = 6\cdot 120 = 720$. For the CARP, it follows that the maximum number of labels to be kept at a node $v \in V$ is bounded by $(Q + 1)\gamma(k, 2)$.

As in (Irnich and Villeneuve, 2006), the new labeling approach will store the intersection of the self-hole sets of all dominating labels as the so-called *running-hole set*, i.e.,

$$H^{run}(P) := \bigcap_{i=1}^{g} H(P_i)$$

whenever $L(P)$ is dominated by $L(P_1), \ldots, L(P_g)$. The label $L(P)$ can be discarded if $H^{run}(P) \subseteq H(P)$ because this is equivalent to $I_g = \bigcap_{i=1}^{g} H(P_i) = H(P) \cap I_g =: I_{g+1}$.

For the labeling algorithm, it means that running-hole set is stored within the label for bookkeeping already identified dominance relations. Whenever one or several new labels are created (by the path extension step, see Section 3.2.1), they are compared for dominance with the existing (old) labels that are already present at the same node. If a new label $L(P)$ dominates an existing label $L(P')$, i.e., $L(P) \prec_{dom} L(P')$, the running-hole set of P' is replaced by $H^{run}(P') := H^{run}(P') \cap H(P)$. Conversely, if an old label $L(P')$ dominates new label $L(P)$, i.e., $L(P') \prec_{dom} L(P)$, the running-hole set of P is replaced by $H^{run}(P) := H^{run}(P) \cap H(P')$. Additional algorithmic tricks (to improve on the average run time) for storing the intersection and checking the above condition were discussed in (Irnich and Villeneuve, 2006, p. 399).

3.2.4 Specifics and Complexity of the CARP Pricing Problem

As mentioned before, in the CARP case the only resources are load and cost. The number of possible states associated with any node $i \in V$ is always bounded by the capacity ($Q+1$ states $0, 1, \ldots, Q$).

Letchford and Oukil (2009) developed a tailored SPPRC labeling algorithm for the CARP that has a very attractive worst-case time complexity of $\mathcal{O}(CD(n, m))$, where $D(n, m)$ is the complexity of Dijkstra's algorithm on a digraph with n nodes and m edges. Using the Fibonacci-heap data structure, the best known bound is $D(n, m) = m+n\log(n)$ (Ahuja *et al.*, 1993).

Letchford and Oukil (2009) modify the label selection rule (for choosing the next path P to be extended) in the following way:

1. In an outer loop over possible values $q = 0, 1, 2, \ldots, Q$ of the load resource paths P with $q(P) = q$ are extended.

2. The extension is split into two parts, the extension along all deadheading arcs first and the extensions along all service arcs second.

3. The first extension (deadheading) produces only labels with identical load q. All extensions can be handled together using a Dijkstra-type of labeling. Note that

for the CARP, the only relevant resource is cost whenever load is fixed. By pre-assigning minimum-cost labels at all nodes, the time complexity $D(n, m)$ can be reached.

4. The latter extensions (service) produce not more than $\mathcal{O}(2m)$ new labels $L(P)$, all with load $q(P) > q$.

5. The overall complexity of all extensions is therefore dominated by the complexity of the Dijkstra algorithm. Taking the outer loop over all load values into account implies an overall complexity of $\mathcal{O}(QD(n, m))$.

In the presence of loop-elimination constraints, up to $\gamma(k_1, k_2)$ labels $L(P)$ with identical load $q(P) = q$ might exist as a consequence of Theorem 3. Therefore, the number of labels to extend can also grow by factor $\gamma(k_1, k_2)$.

Whenever a newly created label dominates another one w.r.t. resources, the update of the running-hole sets of the latter requires only $\mathcal{O}((k_1 + k_2)\omega(k_1, k_2))$ time. Note that dominance compares pairs of labels so that the overall factor is bounded by $\mathcal{O}((k_1 + k_2)\omega(k_1, k_2)\gamma(k_1, k_2)^2)$. This is a constant whenever k_1 and k_2 are fixed.

We have the following final result:

Theorem 4 *For fixed k, labeling for the CARP with combined $(k, 2)$-loop elimination can be performed in $\mathcal{O}(QD(n, m))$ time, where Q is the vehicle capacity and $D(n, m)$ the time of performing the Dijkstra algorithm.*

3.3 Computational Results

This section reports computational results of the branch-and-price algorithm for the CARP first presented in Chapter 2 when $(k, 2)$-loop free relaxations for $k \in \{2, 3, 4\}$ are used. We quantify the impact of the different $(k, 2)$-loop free relaxations on the computation time and the overall best lower bound achieved at the end of the branch-and-price. The branching scheme presented in Chapter 2 consists of three levels of branching decisions: First branching on non-even node degrees, and second branching on edges with fractional edge flow. Both decisions have no impact on the structure of the pricing problem. The third decision is branching on follower information, whenever the information if two edges are serviced consecutive is fractional. This branching rule, however, modifies the network of the underlying graph of the pricing problem. In particular, it requires a second task set to be handled in the SPPRC labeling algorithm that solves the pricing subproblem.

For the branch-and-price, no initial upper bound is given and the node selection rule in branch-and-bound is best-bound first. Note that the same formulation of the (restricted) master problem is used as in Chapter 2, while for the pricing subproblem the following modifications are made: Whenever possible, the simple k-loop elimination pricing is used. If, however, any non-follower constraints is active, the simple k-loop elimination pricing is replaced by $(k, 2)$-loop elimination pricing. Moreover, we use standard heuristic pricing procedures and acceleration techniques for exact pricing as presented in Chapter 4. The

two acceleration techniques applied are bounding with the 2-loop elimination relaxation and bi-direction labeling; for details we refer to (Mingozzi *et al.*, 1997; Baldacci *et al.*, 2011c,b; Righini and Salani, 2006).

The computational study uses two standard benchmark sets from the literature: The first benchmark set `egl` was introduced by Eglese and Li (1992) and can be downloaded from `http://www.uv.es/ belengue/carp/`. This set consists of 24 instances based on the road network of Cumbria. The first 12 instances have 77 nodes and 98 edges, whereas the remaining 12 instances are larger and have 140 nodes and 190 edges. Instances with the same graph size further differ in the number of required edges and the vehicle capacity. The second benchmark set `bmcv` consisting of 100 instances is obtained from the road network of Flanders, Belgium (Beullens *et al.*, 2003). These instances range from 26 to 97 nodes and 35 to 142 edges, where only a subset of the edges is required. The instances were kindly provided by Muyldermans (2012) and the transformed instances into the standard format can be downloaded from `http://logistik.bwl.uni-mainz.de/Dateien/bmcv.zip`. These instances comprise four subsets, where the underlying graph for individual instances of subset `C` and `E` is identical, but the vehicle capacity is 300 for the `C` set and 600 for the `E` set. The same holds for the subsets of instances named `D` and `F`.

All computations were performed on a standard PC with an Intel©Core^{TM} i7-2600 at 3.4 GHz processor with 16 GB of main memory. The algorithm was coded in C++ (MS-Visual Studio, 2010) and the callable library of CPLEX 12.2 was used to iteratively reoptimize the RMP. A hard time limit of four hours for computation has been set for the column-generation and branch-and-price algorithms.

To shorten the notation, we will skip the second entry in $(k, 2)$ so that, in the following, k-loop is a shortcut for $(k, 2)$-loop-free. Since a comprehensive study of linear relaxation results for k-loop elimination with and without activated acceleration techniques are presented in Chapter 2 and Chapter 4, this section focuses on integer results obtained when the branch-and-bound ends (either with an optimal solution or when the given time limit is reached). Tables 4.9–3.7 present the integer results for the `egl` and `bmcv` instances. The header entries in all tables have the following meaning:

instance	name of the instance (for `egl` instances the prefix `egl-` is omitted for the sake of brevity)
ub_{best} or $\underline{opt}$	the best known upper bound (not underlined) or the optimum (underlined) reported in Beullens *et al.* (2003) or Bartolini *et al.* (2013b)
lb^{tree}	lower bound provided by the branch-and-price algorithm within the time limit of four hours; (rounded up to the next integer) 'OPT' indicates that the instance is solved to proven optimality within four hours if the value of lb^{tree} matches the best known upper bound the gap was closed, but no integer optimal solution was computed within the time limit
time	computation time in seconds; if the time limit is reached it is indicated by *4h*

$B\&B$ nodes	report the number of solved branch-and-bound nodes
lb_{own}^{best}	best lower bound over all relaxations tested here
lb_{known}^{best}	best lower bounds round to a multiple of five reported in Beullens *et al.* (2003) or Bartolini *et al.* (2013b)
lb_{BCL}^{best}	best lower bounds reported in Bartolini *et al.* (2013a)

The following additional information is given for the respective relaxation:

Num lb_{own}^{best}	number of instances for which the best lower bound lb_{own}^{best} was reached
Num opt	number of integer optimal solutions
avg time	average time for branch-and-price (with maximum time *4h*)
avg %gap	average gap computed as $\frac{(ub_{best} - lb^{tree})}{ub_{best}} \times 100$

Lower bounds written in **bold** indicate that this bound is a new best bound exceeding the best known lower bounds from the literature. The upper bounds $ub = 11529$ for the instance `egl-e4-c` and $ub = 4650$ for the `bmcv` instance `E11` (written in **bold** also) result from new best integer solutions found with branch-and-price.

For the `egl`-instances, average lower bound values increases with increasing k: The average gap for 2-loop relaxation is 0.54, while it is 0.46 and 0.43 for 3-loop and 4-loop, respectively. There are four exceptions (`e4-a`, `s3-a`, `s3-b` and `s4-a`) where 2-loop relaxation results in better lower bound when the time limit of four hours is reached. Regarding the computation time, 2-loop relaxation performs better for the group of smaller instances (`egl-e`), while the two optimal solutions in the second group (`egl-s`) are computed fastest with 4-loop relaxation. Overall, three new best lower bounds are obtained for `e2-b`, `e3-b` and `e4-c` with 3-loop and 4-loop relaxation.

For the subsets `D` and `F` of `bmcv` instances, 2-loop relaxation gives the best results both regarding bounds and computation times, meaning that the number of best lower bounds and optimal integer solutions is the highest. Moreover, on average the computation times and the gap is also smaller compared to 3-loop or 4-loop. However, for the subsets `C` and `E` with smaller vehicle capacity, results are different: While the number of best lower bounds is still highest with 2-loop relaxation, 3-loop produces for the subset `C` the same number of integer solutions and the same average gap. Moreover, the average computation time decreases for 3-loop. Within the subset `E`, 4-loop relaxation results in more best lower bounds and obtains more integer solutions than 2-loop. Moreover, the average computation time and gap are also smaller for 4-loop than for 2-loop.

Overall, we are able to obtain 19 new best lower bounds out of 33 previously unsolved `bmcv`-instances (5 for subset `C`, 4 for subset `D`, 6 for subset `E` and 4 for subset `F`). Thereof, 15 instances (`C04`, `C19`, `C21`, `C24`, `D08`, `D14`, `D19`, `E11`, `E16`, `E19`, `E20`, `E24`, `F04`, `F08` and `F12`) are solved to optimality for the first time. Bartolini *et al.* (2013b) mentioned that the objective value is always a multiple of five. Using this fact, optimality can be proven

also for instances `D23` and `F23`. At the end, 12 `egl`-instances and 16 `bmcv`-instances remain open.

3.4 Conclusions

We have presented a new dynamic programming labeling algorithm for handling combined task-(k_1, k_2)-loop elimination (with $k_1, k_2 \geq 2$) in SPPRC for situations where loops with respect to two different task sets must be avoided. Compared to standard SPPRC without loop elimination, the proposed dominance relation is still efficient in the following sense: Labels need to be extended by additional attributes (the so-called set forms), where each set form has $k_1 + k_2$ entries and not more than $\omega(k_1, k_2) = (k_1 - 1)!^2 \cdot (k_2 - 1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$ different set forms need to be stored. While in standard SPPRC there is at most one label per state, the maximum number of labels with identical state cannot exceed $\gamma(k_1, k_2) = [k_1 + k_2 - 1] \cdot (k_1 - 1)!^2 \cdot (k_2 - 1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$. Even if these values grow fast with k_1 and k_2, for fixed k_1 and k_2, the bounds $\omega(k_1, k_2)$ and $\gamma(k_1, k_2)$ are constants. Together with the presented update procedures for the attributes these constants guarantee that, for fixed k_1 and k_2, the worst-case computational complexity for solving standard SPPRC and SPPRC with combined task-(k_1, k_2)-loop elimination is identical.

We have applied the new labeling algorithm for SPPRC with combined task-$(k, 2)$-loop elimination for solving pricing subproblems in a branch-and-price algorithm for the CARP. It was known from Chapter 2 that task-k-loop elimination can significantly improve bounds of the linear relaxation of the column-generation master program. However, branching, i.e., a genuine branch-and-price was not possible due to the branching rule implying 2-loop elimination constraints on a new task set. The new results using the SPPRC subproblem relaxation with task-$(k, 2)$-loop elimination allow for a comparison of overall computation times and lower bounds when the branch-and-price algorithm terminates. Using standard benchmark set, we have shown that the approach is competitive: Several new best lower bounds were presented and some knowingly hard instances were solved to proven optimality for the first time.

3.5 Bibliography

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

Baldacci, R., Mingozzi, A., and Roberti, R. (2011b). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011c). New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*.

Bartolini, E., Cordeau, J.-F., and Laporte, G. (2013a). An exact algorithm for the capacitated arc routing problem with deadheading demand. *Operations Research*, **61**(2), 315–327.

Bartolini, E., Cordeau, J.-F., and Laporte, G. (2013b). Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming*, **137**(1-2), 409–452.

Belenguer, J. M., Benavent, E., and Irnich, S. (2013). The capacitated arc routing problem: Exact algorithms. In Corberán and Laporte (2013), chapter 9. (In preparation.).

Benavent, E., Campos, V., Corberán, Á., and Mota, E. (1992). The capacitated arc routing problem: Lower bounds. *Networks*, **22**, 669–690.

Bentley, J. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, **23**(4), 214–229.

Beullens, P., Muyldermans, L., Cattrysse, D., and van Oudheusden, D. (2003). A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, **147**(3), 629–643.

Boland, N., Dethridge, J., and Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, **34**(1), 58–68.

Corberán, Á. and Prins, C. (2010). Recent results on arc routing problems: An annotated bibliography. *Networks*, **56**(1), 50–69.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., and Villeneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 3, pages 57–93. Kluwer Academic Publisher, Boston, Dordrecht, London.

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. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, **42**(5), 977–978.

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

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.

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. (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

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.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**(2), 497–511.

Kohl, N., Desrosiers, J., Madsen, O., Solomon, M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, **33**(1), 101–116.

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.

Mingozzi, A., Bianco, L., and Ricciardelli, S. (1997). Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research*, **45**(3), 365–377.

Muyldermans, L. (2012). Personal Communication.

Nasiri, S. D. and Letchford, A. N. (2013). Pricing routines for vehicle routing with time windows on road networks. Technical Report The Management School, Lancaster University.

Pandi, R. and Muralidharan, B. (1995). A capacitated general routing problem on mixed networks. *Computers & Operations Research*, **22**(5), 465–478.

Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, **3**(3), 255–273.

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, **51**(3), 155–170.

instance	ub_{best} or $\underline{opt}$	2-loop			3-loop			4-loop			lb_{best}^{own}	lb_{best}^{known}	lb_{best}^{BCL}
		lb^{tree}	time	B&B nodes	lb^{tree}	time	B&B nodes	lb^{tree}	time	B&B nodes			
e1-a	3548	OPT	176	23	OPT	516	19	OPT	589	11	3548	3548	3548
e1-b	4498	OPT	1343	659	OPT	1512	374	OPT	3827	311	4498	4498	4498
e1-c	5595	5545	*4h*	3326	5551	*4h*	3057	5555	*4h*	2271	5555	5595	5583
e2-a	5018	OPT	892	340	OPT	2955	227	OPT	6345	102	5018	5018	5018
e2-b	6317	6301	*4h*	3293	6301	*4h*	1376	**6306**	*4h*	779	**6306**	6301	6306
e2-c	8335	8242	*4h*	5601	8269	*4h*	5104	8303	*4h*	3907	8303	8335	8298
e3-a	5898	OPT	106	26	OPT	562	22	OPT	761	19	5898	5898	5898
e3-b	7775	7730	*4h*	3431	**7735**	*4h*	1519	7732	*4h*	607	**7735**	7728	7728
e3-c	10292	10191	*4h*	5396	10220	*4h*	4490	10226	*4h*	3479	10226	10244	10225
e4-a	6444	6408	*4h*	3361	6405	*4h*	280	6399	*4h*	66	6408	6408	6408
e4-b	8961	8892	*4h*	4392	8899	*4h*	1627	8900	*4h*	1096	8900	8935	8919
e4-c	**11529**	11456	*4h*	5924	11488	*4h*	5045	**11502**	*4h*	4316	**11502**	11493	11491
s1-a	5018	OPT	11683	97	OPT	7762	43	OPT	4312	14	5018	5018	5018
s1-b	6388	6386	*4h*	210	OPT	13072	130	OPT	12250	49	6388	6388	6388
s1-c	8518	8440	*4h*	354	8476	*4h*	314	8500	*4h*	310	8500	8518	8518
s2-a	9884	9805	*4h*	847	9806	*4h*	257	9804	*4h*	114	9806	9825	9838
s2-b	13100	12970	*4h*	2320	12978	*4h*	1548	12982	*4h*	1054	12982	13017	12999
s2-c	16425	16351	*4h*	2041	16377	*4h*	2189	16380	*4h*	1949	16380	16425	16395
s3-a	10220	10160	*4h*	547	10154	*4h*	66	10150	*4h*	13	10160	10160	10161
s3-b	13682	13630	*4h*	1515	13629	*4h*	800	13627	*4h*	274	13630	13648	13637
s3-c	17188	17096	*4h*	3102	17122	*4h*	2505	17125	*4h*	2217	17125	17188	17136
s4-a	12268	12149	*4h*	1617	12147	*4h*	271	12142	*4h*	62	12149	12149	12159
s4-b	16283	16104	*4h*	2366	16106	*4h*	1449	16105	*4h*	473	16106	16105	16114
s4-c	20481	20374	*4h*	2797	20397	*4h*	3556	20406	*4h*	3157	20406	20430	20405
Num lb_{own}^{best}		9			9			17					
Num opt		5			6			6					
avg %gap		0.54			0.46			0.43					

Table 3.3: Integer Results for `eg1` Instances

instance	ub_{best} or $\underline{opt}$	2-loop lb^{tree}	2-loop time	2-loop $B\&B$ nodes	3-loop lb^{tree}	3-loop time	3-loop $B\&B$ nodes	4-loop lb^{tree}	4-loop time	4-loop $B\&B$ nodes	lb^{own}_{best}	lb^{known}_{best}	lb^{BCL}_{best}
C01	4150	4144	*4h*	2834	4140	*4h*	1435	4140	*4h*	862	4144	4105	4150
C02	$\underline{3135}$	OPT	6	5	OPT	28	10	OPT	60	6	3135	3135	3135
C03	$\underline{2575}$	OPT	6494	3746	OPT	171	115	OPT	252	120	2575	2575	2575
C04	3510	**OPT**	2163	1317	**OPT**	2678	773	**OPT**	4403	532	**3510**	3480	3510
C05	$\underline{5365}$	OPT	59	81	OPT	169	103	OPT	272	81	5365	5365	5365
C06	$\underline{2535}$	OPT	163	310	OPT	314	157	OPT	196	53	2535	2535	2535
C07	$\underline{4075}$	OPT	278	456	OPT	561	465	OPT	724	401	4075	4075	4075
C08	$\underline{4090}$	OPT	751	474	OPT	778	347	OPT	634	178	4090	4090	4090
C09	5260	5244	*4h*	2989	5242	*4h*	1922	5242	*4h*	1454	5244	5235	5255
C10	$\underline{4700}$	OPT	1363	1604	OPT	1344	1087	OPT	1493	810	4700	4700	4700
C11	4635	4608	*4h*	2564	4608	*4h*	1332	4607	*4h*	612	4608	4585	4620
C12	4240	4234	*4h*	4356	4231	*4h*	2072	4226	*4h*	1211	4234	4210	4240
C13	$\underline{2955}$	OPT	288	612	OPT	456	411	OPT	589	356	2955	2955	2955
C14	$\underline{4030}$	4010	*4h*	5189	4021	*4h*	2810	4024	*4h*	1610	4024	4030	4030
C15	4940	4918	*4h*	1620	4915	*4h*	977	4916	*4h*	670	4918	4915	4935
C16	$\underline{1475}$	OPT	6	13	OPT	42	13	OPT	196	13	1475	1475	1475
C17	$\underline{3555}$	OPT	17	26	OPT	20	16	OPT	23	11	3555	3555	3555
C18	5620	5570	*4h*	1958	5568	*4h*	691	5563	*4h*	292	5570	5580	5590
C19	3115	**OPT**	2311	1324	**OPT**	3204	902	**OPT**	6706	978	**3115**	3100	3115
C20	$\underline{2120}$	OPT	12	20	OPT	136	26	OPT	392	9	2120	2120	2120
C21	3970	**OPT**	9947	3113	**OPT**	1007	130	**OPT**	3284	117	**3970**	3960	3970
C22	$\underline{2245}$	OPT	32	16	OPT	60	11	OPT	256	13	2245	2245	2245
C23	4085	**4073**	*4h*	2752	4072	*4h*	1078	4069	*4h*	400	**4073**	4035	4075
C24	3400	**OPT**	1358	454	**OPT**	975	124	**OPT**	2325	130	**3400**	3385	3400
C25	$\underline{2310}$	OPT	6	10	OPT	10	9	OPT	20	4	2310	2310	2310
Num lb^{best}_{own}		24			18			18					
Num opt		17			17			17					
avg time			5619			5086			5481				
avg %gap		0.13			0.13			0.14					

Table 3.4: Integer Results for `bmcv` Instances, Subset `C`

instance	ub_{best}^{t} or $\overline{opt}$	2-loop			3-loop			4-loop			lb_{best}^{own}	lb_{best}^{known}	lb_{best}^{BCL}
		lb^{tree}	time	B&B nodes	lb^{tree}	time	B&B nodes	lb^{tree}	time	B&B nodes			
E01	4910	**4898**	*4h*	3269	4896	*4h*	2064	4896	*4h*	1517	**4898**	4885	4890
E02	3990	3971	*4h*	4363	3985	*4h*	1835	OPT	862	126	3990	3990	3990
E03	2015	OPT	3	1	OPT	19	3	OPT	23	2	2015	2015	2015
E04	4155	OPT	1593	914	OPT	2584	707	OPT	2789	449	4155	4155	4155
E05	4585	OPT	506	448	OPT	128	61	OPT	68	17	4585	4585	4585
E06	2055	OPT	5	7	OPT	25	13	OPT	54	10	2055	2055	2055
E07	4155	4137	*4h*	4508	4149	*4h*	4079	OPT	6684	2053	4155	4155	4155
E08	4710	OPT	208	131	OPT	160	55	OPT	198	53	4710	4710	4710
E09	5820	5802	*4h*	1028	5800	*4h*	935	5798	*4h*	642	5802	5780	5810
E10	3605	OPT	9	9	OPT	21	6	OPT	48	5	3605	3605	3605
E11	4650	**4650**	*4h*	2218	**OPT**	4244	279	**4650**	*4h*	363	**4650**	4640	4650
E12	4180	4167	*4h*	3623	4169	*4h*	2435	4170	*4h*	1653	4170	4180	4180
E13	3345	OPT	155	220	OPT	264	240	OPT	437	249	3345	3345	3345
E14	4115	4108	*4h*	5195	OPT	1996	1453	OPT	2089	1052	4115	4115	4115
E15	4205	4199	*4h*	1819	4196	*4h*	712	4194	*4h*	292	4199	4190	4205
E16	3775	**OPT**	1287	793	**OPT**	246	103	**OPT**	380	82	**3775**	3755	3775
E17	2740	OPT	4	3	OPT	8	2	OPT	10	2	2740	2740	2740
E18	3835	3825	*4h*	1245	3825	*4h*	446	3825	*4h*	146	3825	3825	3825
E19	3235	**OPT**	7855	993	**OPT**	5905	591	**OPT**	13935	639	**3235**	3225	3235
E20	2825	2815	*4h*	5261	2820	*4h*	2175	**OPT**	3112	464	**2825**	2805	2825
E21	3730	3730	*4h*	5396	3730	*4h*	1434	3730	*4h*	263	3730	3730	3730
E22	2470	OPT	32	24	OPT	74	33	OPT	74	17	2470	2470	2470
E23	3710	3704	*4h*	548	3703	*4h*	385	3699	*4h*	223	3704	3690	3710
E24	4020	**OPT**	9489	2123	**4020**	*4h*	1933	**OPT**	13135	1130	**4020**	4005	4020
E25	1615	OPT	1	4	OPT	3	1	OPT	2	1	1615	1615	1615
Num lb_{own}^{best}		20			17			21					
Num opt		14			15			18					
avg time			7758			6963			6364				
avg %gap		0.12			0.08			0.07					

Table 3.5: Integer Results for `bmcv` Instances, Subset `E`

instance	ub_{best} or *opt*	2-loop lb^{tree}	2-loop time	2-loop B&B nodes	3-loop lb^{tree}	3-loop time	3-loop B&B nodes	4-loop lb^{tree}	4-loop time	4-loop B&B nodes	lb^{own}_{best}	lb^{known}_{best}	lb^{BCL}_{best}
D01	3215	OPT	50	13	OPT	992	17	OPT	4117	16	3215	3215	3215
D02	2520	OPT	26	22	OPT	86	14	OPT	286	15	2520	2520	2520
D03	2065	OPT	43	15	OPT	172	9	OPT	1472	9	2065	2065	2065
D04	2785	OPT	105	33	OPT	600	33	OPT	9022	26	2785	2785	2785
D05	3935	OPT	24	15	OPT	145	19	OPT	166	17	3935	3935	2935
D06	2125	OPT	20	18	OPT	97	5	OPT	1615	15	2125	2125	2125
D07	3115	3108	*4h*	3069	3102	*4h*	893	3098	*4h*	403	3108	3115	3115
D08	3045	**OPT**	3730	736	3041	*4h*	411	3027	*4h*	113	**3045**	2995	3045
D09	4120	OPT	106	36	OPT	929	47	OPT	1654	36	4120	4120	4120
D10	3340	OPT	33	21	OPT	195	23	OPT	493	17	3340	3340	3340
D11	3745	3745	*4h*	1061	OPT	1945	21	OPT	11009	28	3745	3745	3745
D12	3310	OPT	123	17	OPT	539	21	OPT	198	4	3310	3310	3310
D13	2535	OPT	997	741	OPT	61	15	OPT	605	15	2535	2535	2535
D14	3280	**3280**	*4h*	3513	**OPT**	564	35	**OPT**	1804	30	**3280**	3275	3280
D15	3990	OPT	602	41	OPT	3347	17	-	-	-	3990	3990	3990
D16	1060	OPT	7	7	OPT	66	8	OPT	677	10	1060	1060	1060
D17	2620	OPT	11	18	OPT	31	14	OPT	42	8	2620	2620	2620
D18	4165	OPT	2951	48	-	-	-	4165	*4h*	2	4165	4165	4165
D19	2400	**OPT**	552	225	**OPT**	3174	186	**OPT**	13090	195	**2400**	2395	2400
D20	1870	OPT	15	22	OPT	149	21	OPT	2004	20	1870	1870	1870
D21	3050	3005	*4h*	2615	2988	*4h*	261	2982	*4h*	75	3005	2985	3015
D22	1865	OPT	36	15	OPT	251	11	OPT	3200	15	1865	1865	1865
D23	3130	**3126**	*4h*	341	3114	*4h*	13	3111	*4h*	1	**3126**	3115	3125
D24	2710	2704	*4h*	884	2691	*4h*	132	2679	*4h*	45	2704	2680	2710
D25	1815	OPT	10	13	OPT	25	4	OPT	155	8	1815	1815	1815
Num lb^{best}_{own}		25			20			20					
Num opt		19			19			18					
avg time			3834			4567			6673				
avg %gap		0.08			0.15			0.20					

Table 3.6: Integer Results for `bmcv` Instances, Subset `D`

instance	ub_{best} or $\underline{opt}$	2-loop			3-loop			4-loop			lb^{own}_{best}	lb^{known}_{best}	lb^{BCL}_{best}
		lb^{tree}	time	B&B nodes	lb^{tree}	time	B&B nodes	lb^{tree}	time	B&B nodes			
F01	4040	OPT	10942	1103	OPT	988	26	OPT	2170	27	4040	4040	4040
F02	3300	OPT	40	27	OPT	166	13	OPT	1957	77	3300	3300	330
F03	1665	OPT	14	17	OPT	124	14	OPT	507	11	1665	1665	1665
F04	3485	**OPT**	198	48	**OPT**	6933	160	**OPT**	9654	39	**3485**	3480	3485
F05	3605	OPT	61	25	OPT	220	19	OPT	1023	27	3605	3605	3605
F06	1875	OPT	16	18	OPT	73	19	OPT	350	13	1875	1875	1875
F07	3335	OPT	46	15	OPT	190	15	OPT	226	11	3335	3335	3335
F08	3705	**OPT**	174	62	**OPT**	531	40	**OPT**	1927	46	**3705**	3695	3705
F09	4730	OPT	526	38	OPT	1533	34	4730	*4h*	42	4730	4730	4730
F10	2925	OPT	10	13	OPT	58	11	OPT	373	15	2925	2925	2925
F11	3835	OPT	209	33	OPT	1473	30	OPT	4889	26	3835	3835	3835
F12	3395	**OPT**	2341	289	**3395**	*4h*	255	3392	*4h*	107	**3395**	3390	3395
F13	2855	OPT	19	28	OPT	86	27	OPT	306	20	2855	2855	2885
F14	3330	OPT	23	9	OPT	130	14	OPT	822	13	3330	3330	3330
F15	3560	OPT	277	41	3560	*4h*	169	3560	*4h*	95	3560	3560	3560
F16	2725	OPT	44	18	OPT	147	10	OPT	543	9	2725	2725	2725
F17	2055	OPT	6	7	OPT	26	10	OPT	90	7	2055	2055	2055
F18	3075	3065	*4h*	962	3065	*4h*	72	3065	*4h*	36	3065	3065	3065
F19	2525	2515	*4h*	804	2515	*4h*	164	2514	*4h*	124	2515	2500	2525
F20	2445	OPT	56	21	OPT	820	27	OPT	2210	22	2445	2445	2445
F21	2930	OPT	112	33	OPT	1213	29	OPT	3582	37	2930	2930	2930
F22	2075	OPT	25	17	OPT	70	16	OPT	141	19	2075	2075	2075
F23	3005	**3003**	*4h*	140	2998	*4h*	53	2994	*4h*	26	**3003**	2995	3005
F24	3210	OPT	251	29	OPT	2575	32	OPT	10106	33	3210	3210	3210
F25	1390	OPT	4	15	OPT	20	10	OPT	89	10	1390	1390	1390
Num lb^{best}_{own}		25			24			22					
Num opt		22			20			19					
avg time			2344			3575			5095				
avg %gap		0.03			0.04			0.05					

Table 3.7: Integer Results for `bmcv` Instances, Subset `F`

3.6 Appendix

This section contains proofs of the worst-case complexity results for combined (k_1, k_2)-loop elimination as introduced in Section 3.2.3. Note that the proofs follow similar ideas as discussed in the first article on k-cycle elimination (focused on node-routing applications) and we refer the reader to this (Irnich and Villeneuve, 2006) for a more detailed motivation.

3.6.1 Proof of Theorem on Maximum Number of Set Forms

Theorem 2 *For combined (k_1, k_2)-loop elimination, the maximum number of different set forms needed to represent any intersection of self-hole sets $H(P_1) \cap H(P_2) \cap \cdots \cap H(P_l)$ of any set of l paths is $\omega(k_1, k_2) := (k_1 - 1)!^2 \cdot (k_2 - 1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$. This bound $\omega(k_1, k_2)$ is tight.*

Proof 1 *Define $I_1(s), I_2(s)$ of an arbitrary set forms $s = (s^1_1, \ldots, s^1_{k_1-1})(s^2_1, \ldots, s^2_{k_2-1})$ with $s^1_i \in \mathcal{T}^1 \cup \{\cdot\}$ and $s^2_j \in \mathcal{T}^2 \cup \{\cdot\}$ as*

$$I_1(s) := \{i \in \{1, \ldots, k_1 - 1\} | s^1_i = \cdot\} \qquad \text{and} \qquad I_2(s) := \{j \in \{1, \ldots, k_2 - 1\} | s^2_j = \cdot\}$$

Let the $I(s) = (I_1(s), I_2(s))$ be the type of an arbitrary set form s. To shorten the notation we will write $I = (I_1, I_2)$ instead of $I(s) = (I_1(s), I_2(s))$. We denote by $n_{k_1,k_2}(I)$ the maximum number of different set forms that can be generated from a set form of type I by intersection with arbitrarily chosen self-hole sets. n_{k_1,k_2} is defined on all subsets $I = (I_1, I_2) \subseteq (\{1, \ldots, k_1 - 1\}, \{1, \ldots, k_2 - 1\})$. The following recurrences are valid for n_{k_1,k_2}:

$$n_{k_1,k_2}(\emptyset, \emptyset) = 1$$

$$n_{k_1,k_2}(I) = \sum_{i \in I_1} (k_1 - i) n_{k_1,k_2}(I_1 \setminus \{i\}, I_2) + \sum_{j \in I_2} (k_2 - j) n_{k_1,k_2}(I_1, I_2 \setminus \{j\})$$
$$\forall I_1 \subseteq \{1, \ldots, k_1 - 1\} \text{ and } I_2 \subseteq \{1, \ldots, k_2 - 1\} \text{ and } I \neq (\emptyset, \emptyset)$$

The first equation is clear. The second equation is implied by the intersection operation. For each position l there are either $k_1 - l$ or $k_2 - l$ different possibilities to place an element of the self-hole set at this position. This recurrence is solved by

$$n_{k_1,k_2}(I) = \left[|I_1|! \prod_{i \in I_1} (k_1 - i) \right] \left[|I_2|! \prod_{j \in I_2} (k_2 - j) \right] \left[\binom{|I_1| + |I_2|}{|I_1|} \right].$$

This can be seen by induction on the cardinality of I. For $I = (\emptyset, \emptyset)$ this gives $n_{k_1,k_2}(\emptyset, \emptyset) = 1$, which is correct. Now assume, that the above equality is true for all subsets with cardinality $|I| - 1$.

$$n_{k_1,k_2}(I) = \sum_{i \in I_1} (k_1 - i) n_{k_1,k_2}(I_1 \setminus \{i\}, I_2) + \sum_{j \in I_2} (k_2 - j) n_{k_1,k_2}(I_1, I_2 \setminus \{j\})$$

$$
\begin{aligned}
&= \sum_{i \in I_1} (k_1 - i)(|I_1| - 1)! \prod_{l \in I_1 \setminus \{i\}} (k_1 - l)|I_2|! \prod_{m \in I_2} (k_2 - m) \binom{|I_1| + |I_2| - 1}{|I_1| - 1} + \\
&\quad \sum_{j \in I_2} (k_2 - j)|I_1|! \prod_{l \in I_1} (k_1 - l)(|I_2| - 1)! \prod_{m \in I_2 \setminus \{j\}} (k_2 - m) \binom{|I_1| + |I_2| - 1}{|I_1|} \\
&= \sum_{i \in I_1} (|I_1| - 1)!(k_1 - i) \prod_{l \in I_1 \setminus \{i\}} (k_1 - l)|I_2|! \prod_{m \in I_2} (k_2 - m) \binom{|I_1| + |I_2| - 1}{|I_1| - 1} + \\
&\quad \sum_{j \in I_2} |I_1|! \prod_{l \in I_1} (k_1 - l)(|I_2| - 1)!(k_2 - j) \prod_{m \in I_2 \setminus \{j\}} (k_2 - m) \binom{|I_1| + |I_2| - 1}{|I_1|} \\
&= \sum_{i \in I_1} (|I_1| - 1)! \prod_{l \in I_1} (k_1 - l)|I_2|! \prod_{m \in I_2} (k_2 - m) \binom{|I_1| + |I_2| - 1}{|I_1| - 1} + \\
&\quad \sum_{j \in I_2} |I_1|! \prod_{l \in I_1} (k_1 - l)(|I_2| - 1)! \prod_{m \in I_2} (k_2 - m) \binom{|I_1| + |I_2| - 1}{|I_1|} \\
&= \prod_{l \in I_1} (k_1 - l) \prod_{m \in I_2} (k_2 - m) \left[\sum_{i \in I_1} (|I_1| - 1)!|I_2|! \binom{|I_1| + |I_2| - 1}{|I_1| - 1} + \right. \\
&\quad \left. \sum_{j \in I_2} |I_1|!(|I_2| - 1)! \binom{|I_1| + |I_2| - 1}{|I_1|} \right] \\
&= |I_1|! \prod_{l \in I_1} (k_1 - l)|I_2|! \prod_{m \in I_2} (k_2 - m) \left[\binom{|I_1| + |I_2| - 1}{|I_1| - 1} + \binom{|I_1| + |I_2| - 1}{|I_1|} \right] \\
&= |I_1|! \prod_{l \in I_1} (k_1 - l)|I_2|! \prod_{m \in I_2} (k_2 - m) \binom{|I_1| + |I_2|}{|I_1|}
\end{aligned}
$$

The above expression proves that we can get at most $(k_1 - 1)!^2 \cdot (k_2 - 1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$ *different elements in the intersection. To show that this bound is tight we choose any* $\bar{k} = k_1 + k_2$ *different paths* $P_1, \ldots, P_{\bar{k}}$ *with disjoint predecessor tasks on both task-sets. Then the intersection of the corresponding self-hole sets consists of exactly* $(k_1 - 1)!^2 \cdot (k_2 - 1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$ *elements.*

3.6.2 Proof of Theorem on Maximum Number of Paths with Indentical State

Theorem 3 *A collection of* g *dominating paths* $P_1 \prec_{dom} P_2 \prec_{dom} \ldots \prec_{dom} P_g$ *ending at the same node is given. Let the intersections of the corresponding self-hole sets* $H(P_1), H(P_2), \ldots, H(P_g)$ *form a* properly *decreasing chain, i.e.,* $H(P_1) \supsetneq H(P_1) \cap H(P_2) \supsetneq \cdots \supsetneq \bigcap_{i=1}^{g} H(P_i)$*. Then, the length* g *of the properly decreasing chain is bounded by* $\gamma(k_1, k_2) = [k_1 + k_2 - 1] \cdot (k_1 - 1)!^2 \cdot (k_2 - 1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$.

Proof 2 *Every new element of the chain is a result of the intersections made before with*

one new intersection with a self-hole set $H(P_i)$. From Theorem 2 we know that there are at maximum $(k_1-1)!^2 \cdot (k_2-1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$ different set forms in such an intersection. Every set form has $(k_1-1)+(k_2-1)$ entries which results in $[(k_1-1)+(k_2-1)](k_1-1)!^2 \cdot (k_2-1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$ different entries in total. For the computation of the intersection there are two possible operations:

1. *A new set form is generated, where a previously free entry $\cdot$ is specified by an element $t^1 \in \mathcal{T}^1$ or $t^2 \in \mathcal{T}^2$. There exists at most $[(k_1-1)+(k_2-1)] \cdot (k_1-1)!^2 \cdot (k_2-1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$ possible entries to specify.*

2. *On the other hand, a set form can be deleted. This can happen at most $(k_1-1)!^2 \cdot (k_2-1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$ times.*

Since each intersection performs at least one of the above operations, this yields to an upper bound of $[(k_1-1)+(k_2-1)+1](k_1-1)!^2 \cdot (k_2-1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$.

3.7 Bibliography

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.

Chapter 4

In-Depth Analysis of Pricing Problem Relaxations for the Capacitated Arc-Routing Problem

Claudia Bode, Stefan Irnich

Abstract

In Chapter 2 we presented a cut-first branch-and-price-second algorithm for solving the capacitated arc-routing problem (CARP). The fundamental difference to other approaches from the literature for exactly solving the CARP is that the entire algorithm works directly on the typically sparse underlying graph representing the street network. This enables the use of highly efficient dynamic programming-based pricing algorithms for solving the column-generation subproblem also known as the pricing problem. The contribution of this paper is the in-depth analysis of the CARP pricing problem and its possible relaxations, including the construction of new labeling algorithms for their solution, and comprehensive computational tests on standard benchmark problems. We will show that a systematic variation of different relaxations provides a powerful approach to solve knowingly hard instances of the CARP to proven optimality.

4.1 Introduction

The capacitated arc-routing problem (CARP) is the fundamental multiple-vehicle arc-routing problem with applications in waste collection, postal delivery, winter services and more (Dror, 2000; Corberán and Prins, 2010). Chapter 2 presented a new exact solution approach based on an aggregated, non-symmetric formulation that was derived via a Dantzig-Wolfe decomposition of the well-known two-index formulation (Belenguer and Benavent, 1998). For its solution, violated valid inequalities as well as missing variables are generated dynamically. The corresponding cut-and-column-generation algorithm as a whole exploits the fact that the underlying CARP graph is sparse (exploitation of sparsity is an idea that was originally coined by Letchford and Oukil (2009)). Note that any approach using a transformation of the CARP into a node-routing problem results in dense graphs (Baldacci and Maniezzo, 2006; Longo *et al.*, 2006; Bartolini *et al.*, 2013b,a). Using the one-index formulation of the CARP, some relevant valid inequalities are computed a priori in the initial cutting phase. This provides a very fast warm-start of

the column-generation process. Due to direct use of a sparse network for fast pricing, the proposed column-generation algorithm often produces strong lower bounds in relatively short computation time for many instances from the literature. Integrated into branch-and-bound, the approach becomes a cut-first branch-and-price-second algorithm. The computation of integer solutions then benefits from the non-symmetric formulation and, in particular, from an effective branching scheme.

The contribution of this paper is the in-depth analysis of the CARP pricing problem and its possible relaxations, including the construction of new labeling algorithms for their solution, and comprehensive computational tests on standard benchmark problems. Using pricing problem relaxations is a standard technique in column generation (Lübbecke and Desrosiers, 2005; Desaulniers *et al.*, 2005) because pricing problems in routing applications are typically strongly $\mathcal{NP}$-hard elementary shortest-path problems with resource constraints (ESPPRC, Irnich and Desaulniers, 2005). In fact, many successful column-generation approaches play with the trade-off that different pricing problems relaxations offer (Irnich and Villeneuve, 2006; Baldacci *et al.*, 2011b). Stronger relaxations produce tighter lower bounds, but come at the cost of being harder to solve leading to longer computation times in the pricing subproblem. The branch-and-price approach in Chapter 2 made use of just one relaxation producing 2-loops free tours (Benavent *et al.*, 1992). This relaxation is particularly beneficial because it is compatible and at the same time indispensable for branching on followers. Actually, branching on followers and non-followers is the only effective technique known to guarantee the integrality in branch-and-price when pricing is performed on the original sparse network.

In Chapter 2 we already showed that pricing relaxations based on k-loop elimination produce better root node lower bounds. However, for these and other possible relaxations it remained unclear how integer solutions can be computed using the aforementioned branching scheme. For k-loop elimination, Chapter 3 provides an answer to this question by developing an efficient labeling algorithm for loop elimination when two task sets have to be handled (one resulting from elementarity constraints and one from branching). The paper at hand is intended to compare these and other relaxations including *ng*-route relaxations by Baldacci *et al.* (2011b) when combined with state-of-the-art pricing heuristics and acceleration techniques in a branch-and-price for the CARP. We will discuss and empirically analyze the trade-offs between hardness of pricing and strength of lower bounds for various pricing relaxations. As a result, we are able to compute new best lower bounds and optimal solutions for several knowingly hard CARP instances from the benchmark sets of Eglese and Li (1992), Brandão and Eglese (2008), and Beullens *et al.* (2003).

The remainder of this paper is structured as follows: The next section defines the CARP and briefly summarizes the cut-first branch-and-price-second approach presented in Chapter 2. Section 4.3 presents the pricing problem, and discusses well-known and also new pricing relaxations. Several acceleration techniques for solving the shortest-path subproblems via dynamic-programming labeling algorithms such as bidirectional pricing, bounding, and scaling are summarized and adapted to the new relaxations in Section 4.4. In Section 4.5, we presents comprehensive computational results and final conclusions are

drawn in Section 4.6.

4.2 Cut-First Branch-and-Price-Second for the CARP

The CARP has been introduced by Golden and Wong (1981) and studied intensively both from a heuristic and exact algorithm point of view. Heuristics and metaheuristics are essential for computing good upper bounds. Some prominent and successful approaches from the literature include approaches based on tabu search (Brandão and Eglese, 2008), genetic or memetic algorithms (Lacomme *et al.*, 2001; Fu *et al.*, 2010), guided local search (Beullens *et al.*, 2003), variable neighborhood search (Polacek *et al.*, 2008), ant colony optimization (Santos *et al.*, 2010), and many more. A survey on heuristic methods is (Prins, 2013). On the other hand, there are several approaches for computing good lower bounds. Pure polyhedral approaches to the CARP are discussed in (Letchford, 1997; Belenguer and Benavent, 1998, 2003; Ahr, 2004). At the moment, it seems that the most successful exact solution approaches are all based on a combination of cut-and-column generation. Gómez-Cabrero *et al.* (2005) and Martinelli *et al.* (2011) proposed column generation-based algorithms, where either initially computed cuts are added to the column-generation master program or a cutting-plane algorithm is applied during and after the column-generation process. Thereafter, a branch-and-bound procedure follows in (Martinelli *et al.*, 2011). Their branching scheme is not complete meaning that they can only guarantee integer deadheading flows, but route variables may remain fractional.

Complete exact methods were recently presented in (Bartolini *et al.*, 2013b,a) and Chapter 2. The first of the two methods by Bartolini *et al.* consist of computing a cascade of non-decreasing lower bounds, enumerating all routes with reduced cost smaller than the integrality gap of upper bound minus the best lower bound. Then, the master program is solved with a (general purpose) mixed integer-programming solver. Their second method also starts with lower bounding procedures based on column generation, but integer solutions are determined in a branch-cut-price fashion using a three-level hierarchical branching strategy similar to that of Chapter 2. Note that both algorithms of Bartolini *et al.* make intensive use of a transformation of the CARP into a generalized vehicle-routing problem (GVRP) so that route generation is performed on a dense graph. In contrast, the sparsity of the CARP network is heavily exploited in Chapter 2, where in the first phase a cutting-plane algorithm is applied to initialize the column-generation master program and in the second phase the branch-and-price algorithm is executed. This general approach will be explained in detail in Sections 4.2.2 and 4.2.3.

A comprehensive overview on exact CARP approaches is given in (Belenguer *et al.*, 2013) and recent surveys on both heuristic and exact approaches are (Wøhlk, 2008; Corberán and Prins, 2010).

4.2.1 Notation and Definition of the CARP

For the formal definition of the CARP, we assume an undirected and simple graph $G = (V, E)$ with node set V and edge set E. In applications, this graph G is typically *sparse*

so that $|E| \leq \Delta|V|$ holds for a small number $\Delta > 0$. A distinguished node $d \in V$ is given representing the *depot*. All edges $e \in E$ have an associated non-negative integer *demand* $q_e \geq 0$ and those with positive demand form the subset $E_R \subseteq E$ of *required edges*. Required edges have to be served exactly once. All edges $e \in E$, either required or not, can be traversed without providing service (=*deadheading*). CARP costs consist of two components, that is, *service costs* c_e^{serv} for servicing required edges e and *deadheading costs* c_e for all edges e deadheaded.

A *route* is an Eulerian subgraph (V', E') of G with $V' \subseteq V$ and $E' \subseteq E$, where $d \in V'$ holds and E' may contain copies of edges. In fact, E' is a multi-set. By definition, a Eulerian subgraph is connected and all its nodes have an even and positive node degree. A *feasible route* serves a subset $E_s \subseteq E'$ with demand $\sum_{e \in E_s} q_e$ not exceeding the *vehicle capacity* Q. It is assumed that all other edges $E_d := E' \setminus E_s$ are deadheaded (counting copies appropriately). Moreover, it must be *elementary* meaning that E_s is a simple set and does not contain copies of parallel edges. An optimal CARP solution is a cost-minimal set of feasible route such that every required edge $e \in E_R$ is serviced by exactly one route. Note that there might exist a huge number of Eulerian paths for a given Eulerian subgraph, i.e., the same feasible route might be represented by several possibilities of traversals.

Some authors define the CARP for an unlimited fleet of vehicles (Belenguer and Benavent, 2003; Longo *et al.*, 2006; Bartolini *et al.*, 2013b), others fix the number of vehicles (Belenguer and Benavent, 1998) and Chapter 2. Here, the fleet size is also fixed to the minimum number K of required vehicles (computed by solving a bin-packing problem) and we assume that each vehicle of the *homogeneous fleet* has capacity Q and is stationed at the depot d.

Throughout this paper, we use the following standard notation: Given a subset $S \subseteq V$, the *cut set* $\delta(S)$ (the set $E(S)$) is the set of edges with exactly one (both) endpoint(s) in S. The subscript $_R$ indicates the restriction to subsets of required edges so that $\delta_R(S) = \delta(S) \cap E_R$ and $E_R(S) = E(S) \cap E_R$ holds. For simplicity, the abbreviation $\delta(i)$ is used instead of $\delta(\{i\})$ (also $\delta_R(i)$ for $\delta_R(\{i\})$). Given a subset $F \subseteq E$ and any parameter or variable y, the term $y(F)$ stands for $\sum_{e \in F} y_e$.

4.2.2 Cutting-Plane Generation: First Phase

The first phase of the algorithm presented in Chapter 2 consists of the generation of a relevant set of valid inequalities that are later added to the column-generation formulation. Solving the following one-index formulation with a cutting-plane procedure, the added inequalities are those that are binding at the end.

The *one-index formulation* was first considered independently by Letchford (1997) and Belenguer and Benavent (1998). It can be used for computing lower bounds, which are known to be optimal or very tight at least for small and medium-sized instances. However, the one-index formulation is a relaxation of the CARP, since its associated integer polyhedron generally contains infeasible solutions. It uses aggregated deadheading variables $y_e \in \mathbb{Z}_+$ one for each edge $e \in E$. The attribute aggregated refers to the fact that y_e counts the deadheadings over edge e performed by all K vehicles together. The

one-index formulation reads as follows:

$$\begin{aligned}
\min \quad & c^\top y && (4.1)\\
\text{s.t.} \quad & y(\delta(S)) \geq 2K(S) - |\delta_R(S)| \quad \text{for all } \emptyset \neq S \subseteq V \setminus \{d\} && (4.2)\\
& y(\delta(S)) \geq 1 \quad \text{for all } \emptyset \neq S \subseteq V,\ |\delta_R(S)| \text{ odd} && (4.3)\\
& y \in \mathbb{Z}_+^{|E|} && (4.4)
\end{aligned}$$

The objective (4.1) minimizes the costs of all deadheadings (note that service costs are constant and therefore irrelevant for routing decisions). The capacity inequalities (4.2) require that there are at least $2K(S)$ traversals (services and deadheadings) over the cutset $\delta(S)$. Herein, $K(S)$ is the minimum number of vehicles needed to service the edges $E_R(S) \cup \delta_R(S)$. The number $K(S)$ can be approximated by $\lceil q(E_R(S) \cup \delta_R(S))/Q \rceil$ and computed exactly by solving a bin-packing problem. Furthermore, the odd-cut inequalities (4.3) ensure for each subset S with an odd number of required edges in the cut $\delta(S)$ that at least one deadheading is performed. Belenguer and Benavent (2003) introduced disjoint-path inequalities as another class of valid cuts for the CARP. The idea is to consider not only the demand of $E_R(S) \cup \delta_R(S)$ but also the demand on a path from the depot to the set S. The general form of all valid inequalities (including disjoint-path inequalities) can be written as $\sum_{e \in E} d_{es} y_e \geq b_s$ for $s \in \mathcal{S}$ where $\mathcal{S}$ is the set of all inequalities and d_{es} the coefficient of edge e in a particular cut indexed by s.

4.2.3 Branch-and-Price: Second Phase

In the second phase of the algorithm presented in Chapter 2, a restricted master program is iteratively reoptimized and variables with negative reduced costs are generated at each iteration. To obtain integer solutions a branching scheme is applied.

Master Program

The master program is derived by a Danzig-Wolfe decomposition from the two-index formulation by Belenguer and Benavent (1998) extended by additional cuts from the first phase. Because a homogeneous fleet of vehicles is assumed, an aggregation over all vehicles is applied. As a result, the column-generation formulation contains two sets of variables. On the one hand, there are variables $\lambda_r \geq 0$, one for every efficient feasible route $r \in \Omega$, where efficient means that no deadheading along a cycle in G is performed. On the other hand, variables $z_e \geq 0$ for every edge $e = \{i, j\} \in E$ indicate a deadheading along the cycle $(e, e) = (i, j, i)$.

Let $\bar{x}_{er}$ and $\bar{y}_{er}$ be the number of times a route r services and deadheads through an edge e, respectively. The linear relaxation (MP) of the extensive formulation reads then:

$$\begin{aligned}
\min \quad & \sum_{r \in \Omega} c_r \lambda_r + \sum_{e \in E} (2c_e) z_e && (4.5)\\
\text{s.t.} \quad & \sum_{r \in \Omega} \bar{x}_{er} \lambda_r = 1 \quad \text{for all } e \in E_R && (4.6)
\end{aligned}$$

$$\sum_{r\in\Omega} d_{sr}\lambda_r + \sum_{e\in E}(2d_{es})z_e \geq b_s \quad \text{for all } s \in \mathcal{S} \tag{4.7}$$

$$\mathbf{1}^\top\lambda = K \tag{4.8}$$

$$\lambda \geq \mathbf{0}, z \geq \mathbf{0} \tag{4.9}$$

The objective (4.5) consists of minimizing the costs of the routes plus the costs of deadheading along simple cycles. Each required edge must be covered by one route (4.6). Both route variables λ_r and cycle variables z_e are impacted by the additional cuts from phase one. For a specific cut $s \in \mathcal{S}$, the route $r \in \Omega$ has the coefficient $d_{sr} = \sum_{e\in E} d_{es}\bar{y}_{er}$, and the respective coefficient of the cycle variable z_e is $2d_{es}$. Thus, the general form of cuts from the one-index formulation can be transformed into the reformulated cuts (4.7). Since the number of vehicles is fixed, exactly K routes are used (4.8) and all variables are non-negative (4.9).

Note that the exact integrality condition for the integer master program (IMP) is neither $\lambda \in \mathbb{Z}_+^\Omega$ and $z \in \mathbb{Z}_+^E$ nor

$$y_e = \sum_{r\in\Omega} \bar{y}_{er}\lambda_r \in \mathbb{Z}_+. \tag{4.10}$$

The first condition is sufficient, but not necessary, because integer solution can sometimes be reconstructed from fractional λ variables (see Chapter 2). The latter conditions (4.10) are necessary, but not sufficient, see Section 4.2.3 on branching.

Pricing Problem

Because the restricted master program (RMP) is initialized with a proper subset of route variables λ_r, missing variables with negative reduced costs must be priced out. In fact, the task of the pricing problem is the generation of those variables. Let $\pi = (\pi_e)_{e\in E_R}$ be the vector of dual prices for covering constraints (4.6), $\beta = (\beta_s)$ the vector of dual prices for active valid inequalities (4.7), and μ the dual price to the generalized convexity constraint (4.8). Reduced costs for service and deadheading are defined as follows:

$$\tilde{c}_e^{serv} = c_e^{serv} - \pi_e \text{ for all } e \in E_R \qquad \text{and} \qquad \tilde{c}_e = c_e - \sum_{s\in\mathcal{S}} d_{es}\beta_s \text{ for all } e \in E. \tag{4.11}$$

With binary variables x_e for $e \in E_R$ indicating service and integer variables y_e for $e \in E$ for deadheading, the pricing problem to (π, β, μ) is:

$$\begin{aligned}
z_{PP}(\pi,\beta,\mu) \quad = \quad & \min \tilde{c}^{serv,\top}x + \tilde{c}^\top y - \mu && (4.12)\\
\text{s.t.} \quad & x(\delta_R(S)) + y(\delta(S)) \geq 2x_f \quad \text{for all } S \subseteq V\setminus\{d\},\ f \in E_R(S) && (4.13)\\
& x(\delta_R(i)) + y(\delta(i)) = 2p_i \quad \text{for all } i \in V && (4.14)\\
& q^\top x \leq Q && (4.15)\\
& p \in \mathbb{Z}_+^{|V|}, x \in \{0,1\}^{|E_R|}, y \in \mathbb{Z}_+^{|E|} && (4.16)
\end{aligned}$$

The objective (4.12) is the minimization of the reduced costs. Constraints (4.13) ensure connectivity of all required edges serviced. An even node degree is guaranteed by (4.14) using auxiliary integer variables p_i, one for each node $i \in V$. Constraint (4.15) is the capacity constraint.

Obviously, whenever deadheading gives no profit, i.e., $\tilde{c}_e \geq 0$ for all $e \in E$, it is not efficient to have cycles consisting only of deadheading. However, the two-index formulation, from which the master program and pricing problem of Chapter 2 was derived, allows deadheading cycles denoted as extended k-routes in (Belenguer and Benavent, 1998). These extended k-routes correspond to extreme rays of the polyhedron formed by (4.13)–(4.16). The variables z_e in the master program (4.5)–(4.9) model cycles $(e, e) = (i, j, i)$ for each edge $e = \{i, j\} \in E$. Additional variables in this master problem (the primal problem) correspond to inequalities in the associated dual problem. Therefore, the variables z_e give dual inequalities of the form $\sum_{s \in \mathcal{S}} d_{es}\beta_s \leq c_e$ for all $e \in E$. These dual inequalities result in a stabilization of the dual variables β_s (Ben Amor *et al.*, 2006). Moreover, the algorithmic advantage for pricing is the guarantee that the reduced costs $\tilde{c}_e$ of deadheadings over all edges are non-negative. The algorithms presented in Section 4.3 substantially rely on that property.

Note that optimal CARP routes require only the knowledge of the Eulerian subgraphs (V', E') and the partition of E' into served edges $E_s = \{e \in E : x_e = 1\}$ and deadheaded edges E_d. The pricing problem is in fact not a routing problem, since the ordering of serviced and deadheading edges is irrelevant. However, the only viable approach known to us for solving the pricing problem is to compute paths. Hence, we solve a routing problem and herewith determine an ordering of serviced and deadheading edges. We will see that this ordering is also crucial for the branching scheme presented in the next section. As pointed out earlier by Bartolini *et al.* (2013b), a feasible CARP route can then be represented by several possibilities of traversing the corresponding Eulerian subgraph.

Summarizing, the pricing problem asks for a feasible CARP route with minimum reduced cost, where reduced cost $\tilde{c}_e^{serv}$ and $\tilde{c}_e^{deadh}$ for servicing and deadheading along each edge $e \in E$ are given. Since service variables x_e are binary, no feasible CARP route can perform a service for an edge more than once. This is exactly the definition of an *elementary* CARP route. Relaxing the elementarity constraint leads to easier solvable subproblems at the cost of a generally weakened master program lower bound.

Branching

In order to obtain integer solutions, a hierarchical branching scheme was devised. It consists of three levels of branching decisions: (1) branching on node degrees, whenever a node with a non-even degree exists, (2) branching on edges with fractional edge flow, (3) branching on follower information, whenever the information if two edges are serviced consecutive is fractional. Note that the third branching decision is applicable, since the pricing problem is solved as a routing problem, where an ordering of serviced edges is determined. This decision guarantees integer route variables and can be handled by modifying the underlying pricing network. In Chapter 2 we showed that follower constraints in the branching part can be handled in the pricing problem by adding edges

that represent certain paths. On the other hand, non-follower constraints are handled by associating the same task to the corresponding edges. Combinations of several follower and non-follower constraints are more intricate to implement, but follow the same idea.

4.3 Pricing Problem Relaxations

Letchford and Oukil (2009) analyzed two mixed integer linear programming (MIP) models for solving the elementary pricing problem (4.12)–(4.16). When solved with the general purpose MIP solver CPLEX, the resulting computation times were prohibitively long. In principle, the pricing problem (4.12)–(4.16) is solvable as an ESPPRC with tasks on service edges using known labeling techniques from the literature (see Irnich and Desaulniers, 2005). However, as paths can become rather long, ESPPRC labeling still suffers from extensive computation times.

As the ESPPRC is strongly $\mathcal{NP}$-hard, different relaxations were considered in the literature. Letchford and Oukil (2009) proved that the non-elementary relaxation of the pricing problem can be solved in pseudo-polynomial time $\mathcal{O}\left(Q(|E|+|V|\log|V|)\right)$. Their labeling algorithm comprises two building blocks invoked alternately, one is similar to standard labeling approaches for extending labels along service edges and the other is a Dijkstra-like algorithm for extensions along deadheading edges. The Dijkstra steps rely on the property that deadheading edges have non-negative reduced costs (this can be assured, see Section 4.2.3).

A stronger formulation than the non-elementary SPPRC results from the 2-loop-free (=task-1-cycle-free) pricing relaxation already known for the CARP from the work of Benavent *et al.* (1992). Note that task-2-loop-free pricing in the arc routing context allows paths containing task sequences of the form (a, b, a), whereas (a, a) is forbidden. However, in the node routing context node-2-cycle-free pricing allows subpaths (i, j, k, i) and forbids (i, j, i). Both strategies have in common requiring two paths to dominate a third one (see Section 4.3.4 for further details) so that one must record, for every state, a best and a second best label having a different last task. To distinguish between arc and node routing, we will always refer to *loop* freeness in the arc-routing context. Comprehensive computational results with 2-loop-free tours were already presented in Chapter 2.

General requirements We will now outline requirements on any relaxation of the pricing problem to be used within the presented branch-and-price algorithm. In general, applying the suggested hierarchical branching scheme with branching on non-follower constraints means that any pricing problem relaxation must be able to handle two sets of tasks:

- tasks $\mathcal{T}^E$ for modeling the elementary routes
- tasks $\mathcal{T}^B$ for respecting non-follower constraints imposed by branching (2-loop-free tours)

The set $\mathcal{T}^E$ models elementary routes, and due to network modifications in the branching phase, there can be no, one or several tasks of $\mathcal{T}^E$ (forming a task sequence) on a single edge. More precisely, edges modeling deadheading have no task, the original service edges $e \in E_R$ have one task, and edges representing longer paths have a task sequence.

By introducing another set $\mathcal{T}^B$ of tasks, non-follower constraints can be handled in the pricing problem. By associating the same task of $\mathcal{T}^B$ with two different edges, it is guaranteed that any 2-loop-free path will not serve the two edges consecutively (in either direction). For tasks $\mathcal{T}^B$, there can only be no or one task per edge. Note further that any properly stronger relaxation, i.e., forbidding task loops up to a longer loop length than two, also guarantees 2-loop-free paths. However, such a relaxation is too restrictive in the sense that it would also exclude paths that are explicitly allowed in the non-follower branch, e.g., a path that contains a single 3-loop.

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. In the following, we will skip the 'task-' prefix. Consequently, 2-loop-free tours are indispensable, since the only viable branching scheme (known to us) is based on follower and non-follower constraints resulting in edges having identical tasks.

Let P be any path in G. The following attributes are associated with P in a labeling procedure:

$$
\begin{array}{rcl}
i(P) & = & \text{the end node of path } P \\
\tilde{c}(P) & = & \text{the accumulated reduced cost along } P \\
q(P) & = & \text{the accumulated load along } P \\
\mathcal{T}^E(P) & = & \text{the sequence of tasks from } \mathcal{T}^E \text{ in the ordering as serviced by } P \\
\mathcal{T}^B(P) & = & \text{the last task form } \mathcal{T}^B \text{ serviced by } P; \\
 & & \text{if } P \text{ is a pure deadheading path then } \mathcal{T}^B(P) = \cdot
\end{array}
$$

Note that we just need to keep track of the last task $\mathcal{T}^B(P)$ in any dominance algorithm, while for the tasks $\mathcal{T}^E(P)$ the sequence, a part of the sequence or a subset of the tasks might be relevant depending on the respective relaxation.

A feasible path P ending at $i = i(P)$ can be extended along an edge either deadheaded or serviced. Any deadheading extension along an edge $e = \{i, j\} \in \delta(i)$ with associated reduced cost $\tilde{c}_e$ is feasible. The resulting new path P' has the attributes of (4.17). On the other hand, a service extension along an edge $e = \{i, j\} \in \delta_R(i)$ with associated reduced cost $\tilde{c}_e^{serv}$ is feasible if $q(P) + q_e \leq Q$ holds. Moreover, in the ESPPRC case, the task sequences $\mathcal{T}^E(P)$ and $\mathcal{T}^E(i, j)$ must have no task in common, and $\mathcal{T}^B(P) \neq \mathcal{T}^B(i, j)$ needs to be fulfilled. If for one or both paths P and (i, j) there is no last task in $\mathcal{T}^B$, indicated by '$\cdot$', then the latter condition is always considered true. The resulting new path P' has the attributes of (4.18).

$$\begin{aligned} i(P') &= j \\ \tilde{c}(P') &= \tilde{c}(P) + \tilde{c}_e \\ q(P') &= q(P) \\ \mathcal{T}^E(P') &= \mathcal{T}^E(P) \\ \mathcal{T}^B(P') &= \mathcal{T}^B(P) \end{aligned} \tag{4.17}$$

$$\begin{aligned} i(P') &= j \\ \tilde{c}(P') &= \tilde{c}(P) + \tilde{c}_e^{serv} \\ q(P') &= q(P) + q_e \\ \mathcal{T}^E(P') &= (\mathcal{T}^E(P), \mathcal{T}^E(i,j)) \\ \mathcal{T}^B(P') &= \mathcal{T}^B(i,j) \end{aligned} \tag{4.18}$$

In the pure non-elementary case considered by Letchford and Oukil (2009), the attributes $\mathcal{T}^E(P)$ and $\mathcal{T}^B(P)$ are completely ignored. Then, a path P dominates another path Q if $i(P) = i(Q)$, $\tilde{c}(P) \leq \tilde{c}(Q)$, and $q(P) \leq q(Q)$ holds. The entire labeling procedure is summarized in Algorithm 5.

Algorithm 5: Efficient Pricing Algorithm $\mathcal{O}\left(Q \cdot (|E| + |V| \log |V|)\right)$

for $q = 0, 1, 2, \ldots, Q$ **do**
- `// Dijkstra-like extensions`
- Let $\mathcal{P}_q$ be the (sorted) set of paths P with $q(P) = q$
- `// Keep` $\mathcal{P}_q$ `always sorted w.r.t.` $\tilde{c}(P)$ `using a Fibonacci heap`
- **for** $P \in \mathcal{P}_q$ **do**
  - Extend P along deadheading edges $e = \{i, j\} \in \delta(i)$ where $i = i(P)$ using (4.17)
  - Add the new path P' to $\mathcal{P}_q$
  - Apply dominance algorithm among $Q \in \mathcal{P}_q$ with $i(Q) = i(P')$
- `// Service extensions`
- Let $\mathcal{P}_q$ be the (unsorted) set of paths P with $q(P) = q$
- **for** $P \in \mathcal{P}_q$ **do**
  - Extend P along service edges $e = \{i, j\} \in \delta_R(i)$ where $i = i(P)$ using (4.18)
  - **if** *new path* P' *is feasible* **then**
    - `// path` P' `has load` $q(P') = q + q_e > q$
    - Add the new path P' to $\mathcal{P}_{q(P')}$
    - Apply dominance algorithm among $Q \in \mathcal{P}_{q(P')}$ with $i(Q) = i(P')$

`// Filtering step`

Apply dominance algorithm at destination node d among all paths P ending at $d = i(P)$

Some remarks about Algorithm 5 seem appropriate here:

1. In the non-elementary case, dominance is trivial. The set $\{P \in \mathcal{P}_q : i(P) = i, q(P) = q\}$ for a given combination of i and q contains not more than a single path (sometimes no path). Whenever a new path P' is created with load q, it replaces the existing one, say Q, only if it is cheaper, i.e., $\tilde{c}(P') < \tilde{c}(Q)$.

   If paths are stored in arrays (index by node $i(P)$ and load $q(P)$) this dominance step needs just constant time $\mathcal{O}(1)$.

2. The use of a Fibonacci heap data structure (see Ahuja *et al.*, 1993) guarantees the worst-case complexity of $\mathcal{O}(|E| + |V| \log |V|)$ of the Dijkstra-like extensions.

3. The final filtering step is necessary, since the algorithm would otherwise output some paths that are not Pareto-optimal. Note that the dominance procedure among all paths ending at the node d requires $\mathcal{O}(Q)$ time only because paths P with $i(P) = d$ are already sorted by $q(P)$ (by using the indexing).

4.3.1 2-Loop-free Paths

The necessary modification for pricing out only 2-loop-free tours is not complicated. In this case, the tasks for non-followers $\mathcal{T}^B$ are always a subset of the tasks $\mathcal{T}^E$ so that it suffices to be 2-loop-free w.r.t. $\mathcal{T}^B$. Therefore, a path P does not record the sequence $\mathcal{T}^E(P)$, but the node $i(P)$, the cost $\tilde{c}(P)$, the load $q(P)$, and the last task $\mathcal{T}^B(P)$ serviced. A path P dominates a path Q if $i(P) = i(Q)$, $\tilde{c}(P) \leq \tilde{c}(Q)$, $q(P) \leq q(Q)$, and $\mathcal{T}^B(P) = \mathcal{T}^B(Q)$, i.e., they have the same last task. Moreover, two paths P_1 and P_2 with $\mathcal{T}^B(P_1) \neq \mathcal{T}^B(P_2)$ together dominate any other path Q if $i(P_1) = i(P_2) = i(Q)$, $\tilde{c}(P_1), \tilde{c}(P_2) \leq \tilde{c}(Q)$, $q(P_1), q(P_2) \leq q(Q)$ As a result, there are never more than two relevant paths P_1, P_2 with $i(P_1) = i(P_2)$ and $q(P_1) = q(P_2)$, one with minimum cost and one with second best cost having a different preceding task $\mathcal{T}^B(P_1) \neq \mathcal{T}^B(P_2)$. Additional algorithmic tricks for implementing 2-loop elimination can be found in (Kohl, 1995; Larsen, 1999).

4.3.2 *ng*-Route Relaxation

The *ng*-route relaxation by Baldacci *et al.* (2011b) has been successfully applied for solving several VRP variants using cut-and-column generation approaches. The relaxation is parameterized and defined by neighborhoods N_i, one for each node $i \in V$. In the CARP case, $N_i \subseteq \mathcal{T}^E$, i.e., tasks of service edges define the neighborhoods and herewith the relaxation. The principle of the *ng*-route relaxation is that the full sequence $\mathcal{T}^E(P)$ of served tasks associated with a path P is replaced by a subset $\mathcal{T}^E_{NG}(P)$ of the tasks $\mathcal{T}^E(P)$ in the sequence. It means that some of the tasks from the sequence $\mathcal{T}^E(P)$ are disregarded and also the ordering of the tasks is disregarded.

The subset $\mathcal{T}^E_{NG}(P) \subseteq \mathcal{T}^E$ is defined recursively with the extension of a path P ending at node $i = i(P)$ along an edge $e = \{i, j\} \in \delta(i)$. Any deadheading extension is allowed, and the new task set for the resulting path $P' = (P, e, j)$ is $\mathcal{T}^E_{NG}(P') = \mathcal{T}^E_{NG}(P) \cap N_j$. In contrast, the extension along the service edge is considered feasible w.r.t. $(N_i)_{i \in V}$ if and only if $\mathcal{T}^E_{NG}(P) \cap \{\mathcal{T}^E(i, j)\} = \emptyset$, and, in this case, the new path P' has the task subset $\mathcal{T}^E_{NG}(P') = (\mathcal{T}^E_{NG}(P) \cup \{\mathcal{T}^E(i, j)\}) \cap N_j$, where $\{\mathcal{T}^E(i, j)\}$ denotes the *set* of tasks in the service sequence (i, j).

The interpretation of this *ng*-route relaxation is that the neighborhoods N_i work as filters: Any task $t \in \mathcal{T}^E$ serviced along a path P is disregarded whenever $t \notin N_i$ for a node i that is visited after that service. Hence, a repeated service becomes possible then.

Dominance between two paths must consider the subset of tasks. A path P dominates another path Q if $i(P) = i(Q)$, $\tilde{c}(P) \leq \tilde{c}(Q)$, $q(P) \leq q(Q)$, and $\mathcal{T}^E_{NG}(P) \subseteq \mathcal{T}^E_{NG}(Q)$ holds. It can therefore happen that there exist $\mathcal{O}\left(2^{|N_i|}\right)$ different undominated paths P at a node $i(P)$ with identical load $q(P) = q$ for $q \in \{0, 1, 2, \ldots, Q\}$ given.

Obviously, setting all neighborhoods as large as possible, i.e., $N_i = \mathcal{T}^E$, solves the elementary case, ESPPRC, where no loops w.r.t. to any task are allowed. In the general case, however, an *ng*-route relaxation does *not* ensure that every feasible path does not contain a 2-loop w.r.t. $\mathcal{T}^B$. Therefore, the 2-loop freeness w.r.t. $\mathcal{T}^B$ has to be guaranteed additionally. Combining an *ng*-route relaxation w.r.t. $\mathcal{T}^E$ and 2-loop-free routes w.r.t. $\mathcal{T}^B$ is straightforward using both types of associated attributes. The number of different undominated paths P at a node $i(P)$ with identical load $q(P) = q$ can now grow by a factor of two, to $\mathcal{O}\left(2^{1+|N_i|}\right)$.

4.3.3 Partial Elementary

The concept of partial elementarity was presented by Desaulniers *et al.* (2008) and applied to the VRP with time windows (VRPTW). Partial elementarity is a special case of an *ng*-route relaxation where all neighborhood sets $N_i = N$ are identical for all nodes $i \in V$. Thus, elementarity w.r.t. the subset $N \subset \mathcal{T}^E$ must be ensured.

The same attribute updates and dominance rules as for *ng*-route relaxation are applied. Again 2-loop freeness w.r.t. $\mathcal{T}^B$ is not fulfilled automatically, therefore, the partial elementarity relaxation w.r.t. $\mathcal{T}^E$ and 2-loop-free routes w.r.t. $\mathcal{T}^B$ have to be combined. This increases the maximum number of different undominated paths P at the same node and with identical load to $\mathcal{O}\left(2^{1+|N|}\right)$.

4.3.4 $(k, 2)$-Loop-free Paths

It is known that solving an SPPRC with k-loop elimination is a good compromise between solving ESPPRC and SPPRC. Note that a path is k-loop-free if it does not contain a task loop of length k or smaller, e.g., for $k = 3$ no 3-loops and no 2-loops. A general labeling algorithm for k-loop-free SPPRC was first presented by Irnich and Villeneuve (2006). Applying the concept to arc routing, task-loop freeness must be enforced (we omit the prefix 'task-' in the following). In Chapter 2, computational results for solving the linear relaxation of the column-generation master program with k-loop-free pricing were presented for the CARP. Due to the incompatibility of non-follower branching with k-loop elimination for $k \geq 3$, however, no results for branch-and-price were given.

Chapter 3 present a labeling algorithm that calculates $(k, 2)$-loop-free paths, i.e., k-loop free routes w.r.t. $\mathcal{T}^E$ and 2-loop-free routes w.r.t. $\mathcal{T}^B$. They derive a new and efficient dominance rule guaranteeing a small number of labels. Their main theoretical result is that the maximum number of paths to consider at the same node and with identical load is $(k-1)!(k+1)!$. Moreover, for fixed k, the worst-case complexity of the labeling algorithm remains $\mathcal{O}\left(Q \cdot (|E| + |V| \log |V|)\right)$ as for the CARP subproblem without loop elimination (see Algorithm 5). Note that the derivation of the dominance rule is rather

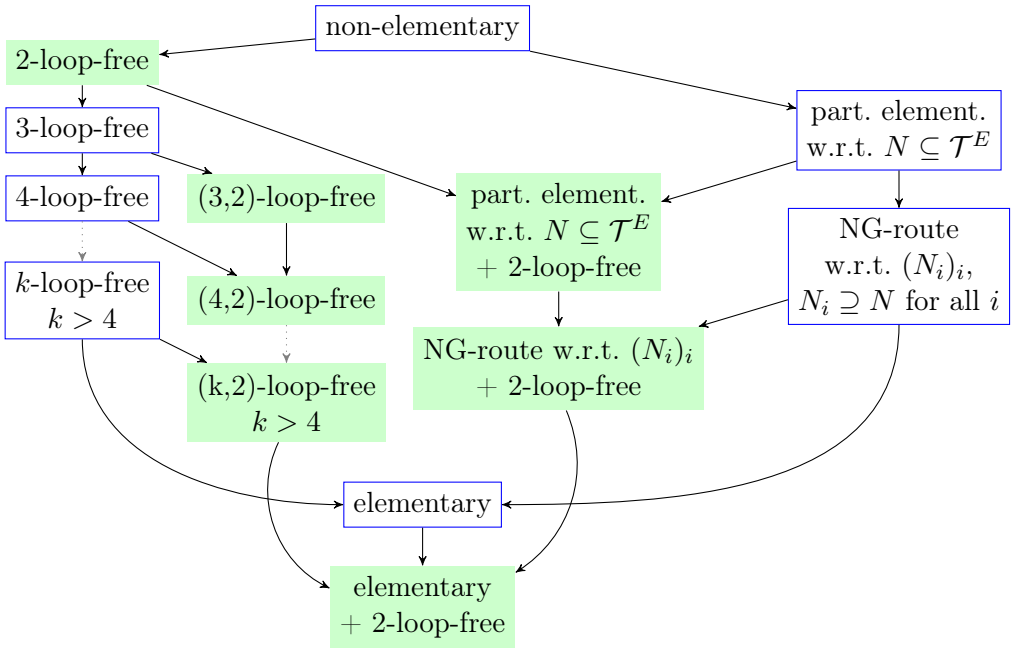


Figure 4.1: Hierarchy of Pricing Relaxations

technical. Therefore, we omit any further description and refer the interested reader to Chapter 3.

4.3.5 Hierarchy of Pricing Relaxations

All presented pricing relaxations form a hierarchy of relaxations beginning with non-elementary pricing as the weakest relaxation and ending with elementary pricing combined with 2-loop elimination as the strongest. This hierarchy is shown in Figure 4.1. An arc connecting two relaxations indicates that the tail is a stronger formulation than the head. For example, the relaxation with $(4, 2)$-loop-free routes is stronger than with 4-loop-free routes and $(3, 2)$-loop-free routes. The relaxations on the right hand side are parameterized with one or several neighborhoods N and $(N_i)_{i \in V}$ so that these boxes represent families of relaxations. Inside each family, relaxations become stronger the larger the subsets N and N_i are (comparable only in case of subset inclusions). Moreover, the ng-route relaxation is stronger than the relaxation with partial elementarity whenever $N_i \supseteq N$ holds for all nodes $i \in V$.

Shaded boxes (■) identify those relaxations that are compatible with our complete branching scheme, in particular, compatible with branching on followers and non-followers. On the other hand, framed boxes (□) represent pricing relaxations applicable only at the root node (or as long as no branching on followers and non-followers occurs).

4.4 Acceleration Techniques

To use acceleration techniques for fast pricing is essential for the effectiveness of the overall branch-and-price approach as outlined by numerous researchers. Some ideas proven useful were summarized in (Desaulniers *et al.*, 2002; Irnich and Desaulniers, 2005). In our case, to run the full exact pricing routine can be time consuming particularly for the $(k, 2)$-loop-free relaxation with larger k and the ng-route relaxations with larger neighborhoods $(N_i)_{i \in V}$. To countervail slow pricing, we implemented heuristic and exact acceleration techniques described in the following.

4.4.1 Pricing Heuristics

Letchford and Oukil (2009) developed two heuristic labeling algorithms for non-elementary pricing In their first heuristic they observed that good paths solving the pricing problem often start with deadheading beginning at the depot, followed by a continuous service part, and finish with deadheading back to the depot. The idea was that a heuristic pricer can restrict itself to assume this structure of the resulting paths. This algorithm is adapted to 2-loop elimination.

In order to eliminate 2-loops, a second type of heuristic occurs naturally. Recall that at every node and for every current load, only the best and second best labels with different predecessor tasks have to be stored. Keeping track of the best label only is the second heuristic. It is easy to adapt the same idea in case of k-loop and $(k, 2)$-loop elimination. Only if the heuristics fail, the exact pricer is invoked.

4.4.2 Bi-Directional Pricing

As pointed out by Righini and Salani (2006), when solving elementary pricing problems with DP, the number of generated states rapidly increases with the stage and the problem size. They proposed a bi-directional labeling algorithm to partially countervail this effect. It outperforms standard mono-directional pricing algorithms as proven for many node-routing applications. This technique can also be applied for all pricing relaxations discussed in Section 4.3.

Specific to the CARP is that the underlying pricing network is undirected so that forward and backward labeling are identical. Labels for both directions need to be calculated just once. Our critical and only possible resource for bounding is the load. Therefore, we extend paths P only if the current load $q(P)$ is less than or equal to $\lceil Q/2 \rceil$. Two generated labels are then combined similar to the procedure `join` presented in (Righini and Salani, 2006). The main difference is that we merge two paths with common end node, while Righini and Salani (2006) suggest merging over connecting arcs. Two specific implementation details of bidirectional labeling are considered next.

2-Loop-free Paths A special case occurs when 2-loop-free paths are generated. If the `join` procedure is implemented in a straightforward fashion, its complexity is $\mathcal{O}(|V|Q^2)$

because up to $4(Q+1)^2$ pairs of paths need to be compared at each node. For the 2-loop-free relaxation, where the number of labels at a node does not grow but is constant for increasing values of the load q, preliminary tests have shown that the `join` procedure dominates the run time. Therefore, a more efficient `join` is needed.

While the standard `join` finally guarantees the determination of all Pareto-optimal origin-destination paths, we propose a more efficient variant of `join` with complexity $\mathcal{O}(|V|Q)$, which does not guarantee the determination of the complete Pareto frontier. Instead, it is ensured that a least-cost path and all Pareto-optimal paths with load not exceeding $Q/2$ are determined. (Generally, many more Pareto-optimal paths are found.) As in the standard case, our `join` relies on the computation of a set of Pareto-optimal paths P with load $q(P) \leq \lceil Q/2 \rceil$ identified with mono-directional labeling. Then it works as follows: For every node and for every value $q = 0, 1, 2, \ldots, \lceil Q/2 \rceil$ we determine a best path $P_1^{(q)}$ and a second best path $P_2^{(q)}$ with $q(P_1^{(q)}), q(P_2^{(q)}) \leq q$, where the last task of the best and the second best path must differ. Then, to generate paths P with load $q(P) > Q/2$, a loop over all values $q = 0, 1, 2, \ldots, \lceil Q/2 \rceil$ is performed, and we merge, if feasible, combinations of the paths $P_i^{(q)}$ and $P_j^{(Q-q)}$ for $i, j \in \{1, 2\}, i + j \leq 3$ ending at the same node. This requires only $\mathcal{O}(|V|Q)$ time and space.

Note that it is non-trivial to transfer the idea to general $(k, 2)$-loop elimination for $k > 2$ because there are generally more than two paths with identical load ending at every node. Therefore, the standard `join` is used here.

ng-Route Relaxation The half-way test is a component of the `join` procedure and assures that the same path P with $q(P) > Q/2$ is not generated multiple times. In principle, this happens whenever P can be split differently into $P = (Q, R)$ with $q(Q) > Q/2$. The half-way test proposed by Righini and Salani (2006), in the node-routing context, requires that the split point is chosen as the first node on the path where the critical resource exceeds the bound. In the CARP case, consider a path $Q = (Q', e, j)$ with last edge $e \in E$ and last node j. Then, the half-way test requires that the *last edge is serviced* so that $q(Q') \leq Q/2$ and $q(Q) = q(Q') + q_e > Q/2$ holds. As a result, no path P is generated twice.

However, for the CARP and the ng-route relaxation, the half-way test is too restrictive. Again, we assume constructing the path $P = (Q, R)$ with $Q = (Q', e, j)$, i.e., last serviced edge $e \in E_R$ and last node j. The critical situation is when extending Q to another node $\ell \in V$ and when a task $e^* \in \mathcal{T}_{NG}^E(Q)$ is not contained in the neighborhood N_ℓ, i.e., $e^* \notin N_\ell$. Thus, the information that the task e^* was serviced along Q is not recorded in a label ending at node ℓ. Now consider the path $P' = (Q, e', \ell, e', j)$ where the two last extensions are deadheadings along the edge $e' = \{j, \ell\} \in E$. The path P' dominates path Q w.r.t. resources whenever the deadheading costs $\tilde{c}_{j\ell} = \tilde{c}_{e'}$ are zero. Moreover, it may properly dominate w.r.t. ng-neighbors because $e^* \notin N_\ell$. In this case, Q does not exist, but P' does not qualify as a forward path in `join` because its last edge is deadheaded.

In fact, our first implementation contained the (incorrect) half-way test, and cost-minimal paths were missing in very rare occasions. However, it happened that inconsistent bounds were computed in the branch-and-price so that this subtle detail became a

serious flaw.

Instead of applying the half-way test, we now store for every value $q = 0, 1, \ldots, Q$ a minimum reduced cost joined path and reconstruct on that basis only the Pareto-optimal paths. This is obviously a little less efficient, but the only viable approach known to us.

4.4.3 Bounding

Bounding is intended to reduce the number of states to expand in a DP approach. In the (E)SPPRC pricing context, for a partial path P at hand, the idea is to calculate a lower bound on the (reduced) cost of any completion to the destination node. If the cost of the path P plus the lower bound exceeds zero, path P can be discarded because it is useless for constructing negative reduced cost routes.

Note that in the CARP the network is fully symmetric so that forward and backward labeling is identical. Any relaxation solved with mono-directional labeling on the original network so provides lower bounds on the cost of a completion to the destination node. The hierarchy of relaxations depicted in Figure 4.1 offers numerous possibilities for pricing problem relaxations and proper relaxations of these that in combination allow bounding.

For example, 2-loop-free pricing can be used for bounding purposes in combination with any other relaxation compatible with branching. Additionally, $(\ell, 2)$-loop-free tours allow bounding for the $(k, 2)$-loop-free relaxation if $\ell < k$. Even more, in the ng-route relaxation with neighborhoods $(N_i)_{i \in V}$, smaller neighborhoods $N_i' \subset N_i$ might be used for bounding. In all cases, we implemented bounding so that the weaker relaxation provides a bounding function $f(i, q)$ defined for every node $i \in V$ and load $q \in \{0, 1, \ldots, Q\}$. The value $f(i, q)$ is a lower bound on the reduced costs of feasible paths ending at node i with not more than load q on board. When solving the stronger relaxation, any path P with $\tilde{c}(P) + f(i(P), Q - q(P)) > 0$ is identified being useless, and its label can be discarded.

4.5 Computational Results

This section reports computational results of the various pricing relaxations tested when solving the respective linear relaxation and integer formulations of the CARP. Note that Chapter 3 already contained integer results with $(k, 2)$-loop elimination, but no other relaxations, no comparisons, and no analysis of the impact of the acceleration techniques. The first benchmark set `egl` was introduced by Eglese and Li (1992) and can be downloaded from `http://www.uv.es/~belengue/carp/`. This set consists of 24 instances based on the road network of Cumbria. Group `e` consists of instances with 77 nodes and 98 edges, whereas group `s` is larger and has instances with 140 nodes and 190 edges. Each group is further split into four subsets $m \in \{1, \ldots, 4\}$, where the number of required edges increases with m. On the lowest level, each subgroup differs in the vehicle capacity, where three different sizes are assumed, indicated by $n \in \{\texttt{a},\texttt{b},\texttt{c}\}$. Within each subgroup, the instances `a` have highest capacity tending to result in less but longer routes, and instances `c` have lowest capacity resulting in more but shorter routes. Overall, instance names are coded as follows: `egl-`lm`-`n with $l \in \{\texttt{e},\texttt{s}\}$, $m \in \{1, \ldots, 4\}$, and $n \in \{\texttt{a},\texttt{b},\texttt{c}\}$.

The second benchmark set `bmcv` consisting of 100 instances is obtained from the road network of Flanders, Belgium (Beullens *et al.*, 2003). These instances range from 26 to 97 nodes and 35 to 142 edges, where only a subset of the edges is required. The instances were kindly provided by Muyldermans (2012) and comprise four subsets. The underlying graph for individual instances of subset `C` and `E` is identical, but the vehicle capacity is 300 for the `C` set and 600 for the `E` set. The same holds for the subsets of instances named `D` and `F`.

4.5.1 Computational Setup

All computations were performed on a standard PC with an Intel©Core^{TM} i7-2600 processor at 3.4 GHz with 16 GB of main memory. The algorithm was coded in C++ (MS-Visual Studio, 2010) and the callable library of CPLEX 12.2 was used to iteratively reoptimize the RMP. A hard time limit of four hours for computation has been set for the column-generation and branch-and-price algorithms.

We tested both $(k, 2)$-loop-free and ng-route relaxations with several parameter settings. Within $(k, 2)$-loop-free pricing we varied $k \in \{2, 3, 4\}$ and the relaxation used for bounding. In detail, for $(3, 2)$-loop-free pricing and ng-route relaxation we used the 2-loop-free relaxation and for $(4, 2)$-loop-free pricing we used both the 2-loop-free and $(3, 2)$-loop-free relaxation for bounding. To shorten the notation, we will skip the second entry because it is equal for all $(k, 2)$-loop-free relaxations. Therefore, in the following, k-loop is a short-cut for $(k, 2)$-loop-free pricing. In the same spirit we write 4b2-loop as a short form of $(4, 2)$-loop-free pricing with 2-loop-free bounding.

The choice of neighborhoods $(N_i)_{i \in V}$ has a great impact on the strength of the ng-route relaxation and the computational effort needed in every pricing iteration. Because there is an exponential number of possible choices, we decided to focus our analysis to the most influential parameter, which is the maximum size of a neighborhood. Here we ran the algorithms with parameters $n_{ng} \in \{3, 4, 5, 6, 7, 8, 9, 10, 12, 15\}$ meaning that all neighborhood sizes $|N_i|$ do not exceed n_{ng}, i.e., for $|N_i| \leq n_{ng}$. To indicate the (maximum) size of the neighborhoods, we write, e.g., $ng6$ whenever $|N_i| \leq 6$.

There exist several methods of determining the concrete sets N_i. Desaulniers *et al.* (2008) proposed an algorithm for partially elementary, i.e., $N_i = N$ for all $i \in V$, in which iteratively the linear relaxation of the RMP is solved. As long as the neighborhood size $|N|$ is smaller than a predefined maximal size n_{max} and there exists a task cycle in the solution, this task is added to the neighborhood N. Tasks with a large flow on cycles are chosen with priority. On the other hand, Baldacci *et al.* (2011b) use individual neighborhoods N_i for every node $i \in V$. The sets N_i are pre-computed by adding a customer j to N_i if it is among the n_{ng} nearest nodes to node i. We combine these two ideas because we dynamically generate individual neighborhoods N_i (a similar idea was presented by R. Roberti in the presentation (Baldacci *et al.*, 2011a)). The procedure is summarized in Algorithm 6.

Note that when adding new tasks to a neighborhood N_i, the resulting relaxation becomes more restrictive so that a formerly feasible route r can become infeasible. Those routes that become infeasible have to be removed from the RMP at the beginning of

Algorithm 6: Generation of Neighborhoods $(N_i)_{i \in V}$

```
Set N_i = ∅ for all i ∈ V
while do
    Solve the current linear relaxation (the RMP) for the ng-route relaxation
    defined by (N_i)_{i∈V}
    for e ∈ T^E do
        Compute the set of all elementary cycles C with positive flow f(C) > 0
        defined by task e
        for cycles C do
            if |N_i ∪ {e}| ≤ n_ng for all i ∈ V(C) then
                Add cycle C to the candidate list C
                Store with cycle C the task e = e(C), flow f(C) and its nodes V(C)
    if |C| > 0 then
        Determine cycle C ∈ C with maximum flow f(C)
        Add task e(C) to the neighborhoods N_i of all nodes i ∈ V(C)
    else
        Stop!
```

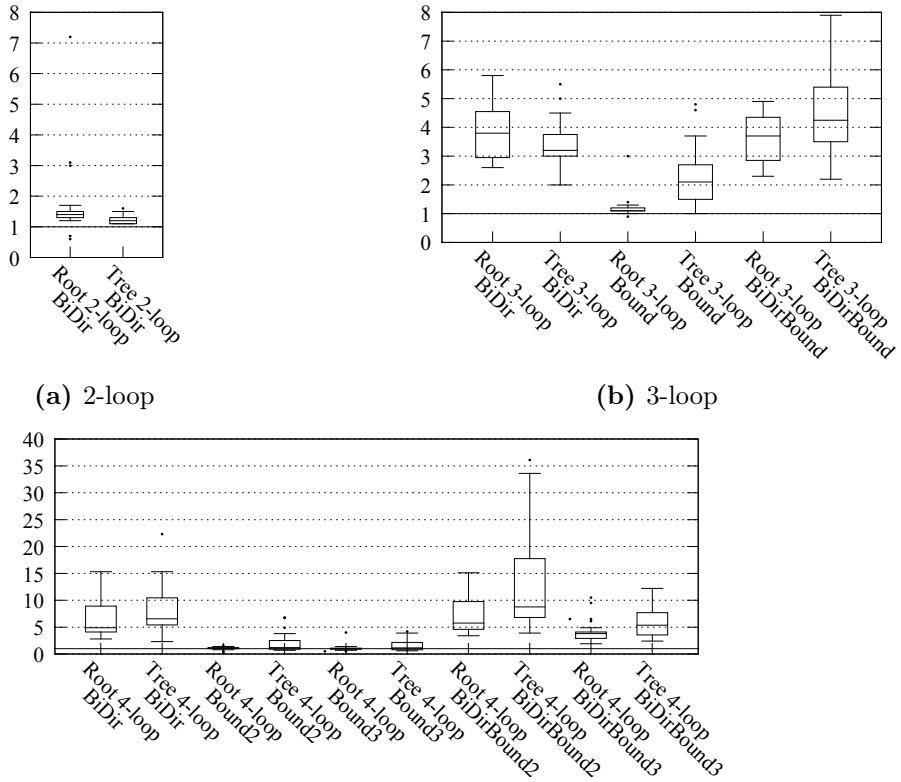
every main loop of Algorithm 6. Thus, the RMP first gets smaller, while it increases again with every newly generated route.

Finally, bidirectional labeling can be applied in every pricing algorithm. In the following, we indicate bidirectional labeling with the term 'BiDir'.

4.5.2 Impact of Acceleration Techniques

We start with analyzing the impact of the acceleration techniques presented in Section 4.4. In order to measure the improvement of bounding and bidirectional pricing for different pricing relaxations, both the root node and the full branch-and-bound tree were solved with no, one, or both techniques active. Computations were performed for all 24 `egl` instances and the different relaxations. The improvement is then calculated as the ratio of the time for pricing without acceleration and the time with one or both techniques active, respectively, for each instance. For abbreviation, we refer to the these numbers as *acceleration factors*. For not biasing the acceleration factors, we turned off all heuristic pricing procedures. Figures 4.2 and 4.3 show the resulting box-and-whisker diagrams (McGill *et al.*, 1978).

Comparing the results among the k-loop-free relaxations, bidirectional pricing has a higher impact the larger k is. For 2-loop, the only acceleration technique is bidirectional pricing, where for the linear relaxation ('Root') the median acceleration factor is 1.4 with 50% of the data lying in a very small range inside the box. Figure 4.2a shows that the acceleration factor is slightly smaller considering the overall branch-and-price tree ('Tree').



(a) 2-loop **(b)** 3-loop

(c) 4-loop

Figure 4.2: Impact of Bidirectional Pricing and/or Bounding for the $(k, 2)$-loop Relaxations: Box-and-Whisker Diagrams of the Acceleration Factors

This median increases to 3.8 and 5.1 for 3-loop and 4-loop, respectively (see Figures 4.2b and 4.2c). For these relaxations, bidirectional pricing has always an impact greater than one, nevertheless the data scatters more. For example, for the instance `e4-a` solving the root node with bidirectional pricing is about 15 times faster than with the basic 4-loop algorithm, and for the instance `s4-c` just 2.8 times faster. For indicating the spread of the data, the end of the whiskers show data that lying within the 1.5 interquartile range. Any other data is outliers and they are represented by small dots.

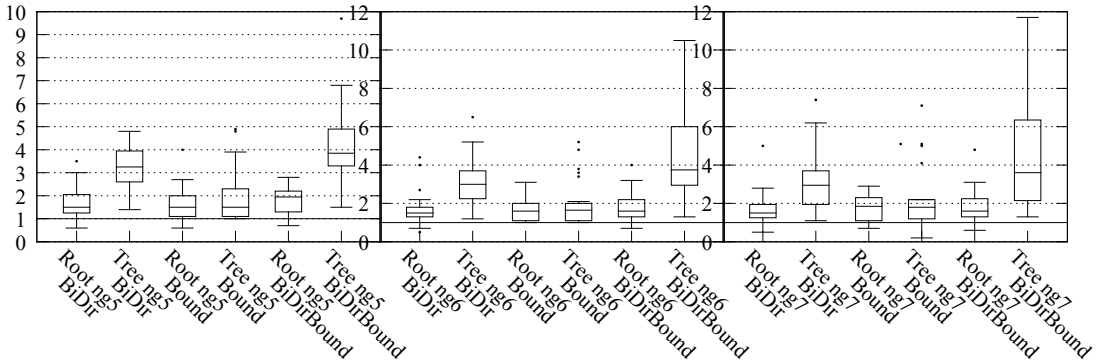
Comparing the results over the full branch-and-bound tree solely using bidirectional pricing, there is an improvement compared to the root node only for 4-loop pricing. However, combined with bounding the positive impact of using acceleration techniques is strengthened. Sometimes a speed up factor of 36 can be reached (instance `s2-c` in 4*b*2-loop pricing).

The impact of using bounding alone is very small, in particular for solving the linear relaxation ('Root'). The median within 3-loop pricing is only slightly above 1.0 and the lower whisker is ending at 1.0. There, bounding has always a small but non-negative

impact compared to 4-loop pricing. The median for bounding with 2-loop and 3-loop bounding is 1.0 and 0.9, respectively. Hence, bounding alone often results in longer computation times. Considering the whole branch-and-bound tree ('Tree'), the acceleration factors are slightly higher.

Finally, for the relaxation with 4-loop-free routes, the comparison of bounding with the 2-loop and 3-loop shows a clear winner: 2-loop-free bounding is superior to 3-loop-free bounding meaning that slightly better bounds are obtained.

The impact of bidirectional pricing and bounding is, at the root node, very similar for all tested *ng*-route relaxations (see Figures 4.3a–4.3c). The median of all acceleration



(a) ng5 **(b)** ng6 **(c)** ng7

Figure 4.3: Impact of Bidirectional Pricing and/or Bounding for the *ng*-route Relaxations: Box-and-Whisker Diagrams of the Acceleration Factors

techniques is between approximately 1.5 and 2.0, and the dispersion of the data is not as high as for the k-loop relaxations. However, except for solely bounding within $ng6$, there are instances where solving the root node takes longer than without any acceleration techniques. Similar to k-loop, considering the full branch-and-bound tree, the impact of bounding and/or bidirectional pricing is at least as good as at the root node, but often better. The only exception is bounding within the $ng7$-route relaxation: The median is approximately the same comparing the root node and the full tree, but there are instances (e.g., `e1-b` and `e2-b`) where solving the pricing problem is up to five times slower than the basic $ng7$-route algorithm. In general, combining all presented acceleration techniques for solving the branch-and-price part gives the best results. Therefore, all following computational results are presented for combined bidirectional pricing with bounding.

4.5.3 Linear Relaxation Results

The focus of the following analysis is on lower bounds obtained with the linear relaxations (at the root node). A comprehensive study for the `egl` instances and relaxations with k-loop elimination was already presented in Chapter 2. However, no acceleration techniques and no *ng*-route relaxations were considered. Therefore, we will now present lower bounds and computation times for k-loop elimination and *ng*-route relaxations with the

presented acceleration techniques activated. Table 4.1 presents aggregated results for the `egl` instances and Table 4.2 for the `bmcv` instances.

Table 4.1: Aggregated Linear Relaxation Results for `egl` Instances

	2-loop	3-loop	4b2-loop	4b3-loop	*ng*5	*ng*6	*ng*7
Minimum gap (%)	0.07	0.05	0.05	0.05	0.00	0.00	0.00
Average gap (%)	0.84	0.74	0.68	0.68	0.61	0.59	0.58
Maximum gap (%)	1.60	1.30	1.29	1.29	1.24	1.23	1.23
Minimum time (s)	9	22	21	26	63	67	65
Average time (s)	90	233	511	615	1,646	2,220	2,601
Maximum time (s)	294	837	4,151	3,660	10,507	10,016	14,306

Table 4.2: Aggregated Linear Relaxation Results for `bmcv` Instances

	2-loop	3-loop	4b2-loop	4b3-loop	*ng*5	*ng*6	*ng*7
Minimum gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Average gap (%)	0.55	0.54	0.52	0.52	0.48	0.48	0.48
Maximum gap (%)	2.79	2.69	2.69	2.69	2.39	2.37	2.37
Minimum time (s)	1	3	2	2	2	2	2
Average time (s)	20	317	426	274	1,668	2,055	2,277
Maximum time (s)	194	22,760	14,914	12,902	14,373	14,288	14,253

In preliminary computational tests we varied n_{ng} more widely including values between $n_{ng} = 3$ and $n_{ng} = 15$. We tested the relaxations inside the overall branch-and-price algorithm and counted the number of times that a specific relaxation produced the best lower bound at the time limit. It turned out that the relaxations with $n_{ng} \in \{5, 6, 7\}$ outperformed the others (except for some rare outliers). Hence, we report results for *ng*-route relaxations only for the three parameters $n_{ng} \in \{5, 6, 7\}$.

Due to the integration of 2-loop-free pricing in *ng*-route relaxations (see Section 4.3), the lower bounds obtained with any *ng*-route relaxation are always at least as good as the lower bounds with the 2-loop-free relaxation. Therefore, a stronger relaxation results in better lower bounds, i.e., smaller gaps in the best, average, and worst case. Some substantial improvements were observed, e.g., 69 units for the instances `egl-e2-c` and `egl-s1-c`. For all relaxations, the minimum gap for the `bmcv` instances is zero meaning that at the root node the gap is closed. As expected, solving the linear relaxation becomes more time consuming for both increasing values of k and n_{ng}. However, bounds alone do not provide a comprehensive assessment because, on the average, solving the root node with k-loop relaxation is significantly faster than with an *ng*-route relaxation. Detailed results with lower bound values and computation times for all instances can be found in the Appendix in Tables 4.6–4.8.

4.5.4 Integer Solution Results

Next we summarize integer results for the `egl` and `bmcv` instances. Given the time limit of four hours (14,400s) for solving each instance, we report the number of instances

solved to optimality ('Num. opt. sol.'), the number of instances where the respective relaxation produced the best lower bound among all tested relaxations ('Num best *lb*'), and the remaining gap at the end of the branch-and-price tree (using the best known upper bound *ub*). Note that the node-selection rule was best first. Aggregated results are presented in Tables 4.3 and 4.4, while detailed results for individual instances can be found in the Appendix in Tables 4.9–4.10.

Table 4.3: Aggregated Integer Results for `egl` Instances

	2-loop	3-loop	4b2-loop	4b3-loop
Num. opt. sol. (all/a/b/c)	5/4/1/0	6/4/2/0	6/4/2/0	6/4/2/0
Num. best *lb* (all/a/b/c)	7/6/1/0	6/4/2/0	7/4/2/1	6/4/2/0
Average gap (%)	0.69	0.62	0.57	0.58
Maximum gap (%)	1.12	1.09	1.09	1.10

	*ng*4	*ng*5	*ng*6	*ng*7
Num. opt. sol. (all/a/b/c)	4/3/1/0	3/2/1/0	4/2/2/0	3/1/2/0
Num. best *lb* (all/a/b/c)	6/3/2/1	6/3/2/1	13/2/5/6	13/2/4/7
Average gap (%)	0.48	0.43	0.44	0.43
Maximum gap (%)	1.04	1.06	1.06	1.07

Table 4.4: Aggregated Integer Results for `bmcv` Instances

	2-loop	3-loop	4b2-loop
Num. opt. sol. (all/C/D/E/F)	75/17/21/15/22	75/17/20/16/22	76/17/19/19/21
Num. best *lb* (all/C/D/E/F)	85/21/23/16/25	72/17/19/14/22	72/17/18/17/20
Average gap (%)	0.31	0.34	0.42
Maximum gap (%)	1.48	2.03	2.23

	4b3-loop
Num. opt. sol. (all/C/D/E/F)	76/17/19/19/21
Num. best *lb* (all/C/D/E/F)	67/17/16/15/19
Average gap (%)	0.41
Maximum gap (%)	2.26

	*ng*5	*ng*6	*ng*7
Num. opt. sol. (all/C/D/E/F)	76/18/19/19/20	75/18/19/18/20	76/18/19/19/20
Num. best *lb* (all/C/D/E/F)	71/21/15/19/16	69/22/14/18/15	68/20/12/21/15
Average gap (%)	0.37	0.39	0.41
Maximum gap (%)	2.20	2.26	2.26

For the `egl` instances, the k-loop relaxations are able to find more integer solutions, while for the `bmcv` the ng-route relaxation and the k-loop relaxations produce approximately the same number of optima. Whenever the time limit is reached, $ng6$ and $ng7$ produce the best lower bounds for the `egl` instances, and both the average and maximum gap is generally better for ng-route relaxations. In contrast, for `bmcv` instances, the 2-loop relaxation gives the best solutions both on average and with respect to the maximum

gap. However, there is the tendency that the 2-loop relaxation can solve problems of groups with higher vehicle capacity (i.e. `egl-`*lm*`-a` and `bmcv D` and `F`) better (6, 23, and 25 best lower bounds), while the best *ng*-route relaxation, i.e., *ng*7, performs worse on these instances (only 2, 12, and 15 best lower bounds). On the other hand, for instances with lower capacity, i.e., `egl-`*lm*`-c`, `bmcv C` and `E`, the 2-loop-free relaxations results in 0, 21, and 16 best lower bounds, while *ng*7 gives 7, 20, and 21 best results.

4.5.5 Strong Branching and Integer Solution Results

Strong branching is a technique where several candidates for branching are evaluated before taking the actual branching decision. For a general discussion of strong branching techniques we refer to (Achterberg *et al.*, 2005).

We tested the k-loop relaxations for $k \in \{2, 3, 4\}$ and the *ng*6 and *ng*7 relaxations with five and ten candidates on the `egl` instances. We restrict strong branching to branch-and-bound nodes at levels not exceeding ten, i.e., with not more than ten nodes between the the root node and the node under consideration. Table 4.13 in the Appendix presents detailed results for computations with strong branching for all `egl-`*lm*-*n* instances, while Table 4.5 presents aggregated information.

Table 4.5: Aggregated Integer Results with Strong Branching for `egl` Instances

	2-loop		3-loop		4b2-loop	
	sb5	sb10	sb5	sb10	sb5	sb10
Num. opt. sol. (all/a/b/c)	5/4/1/0	5/4/1/0	5/4/1/0	5/4/1/0	5/4/1/0	4/3/1/0
Num. best *lb* (all/a/b/c)	6/4/2/0	9/8/1/0	6/4/2/0	5/4/1/0	5/4/1/0	6/3/1/2
Average gap (%)	0,66	0,66	0,57	0,56	0,52	0,50
Maximum gap (%)	1,10	1,09	1,08	1,08	1,10	1,09

	4b3-loop	
	sb5	sb10
Num. opt. sol. (all/a/b/c)	4/3/1/0	4/3/1/0
Num. best *lb* (all/a/b/c)	4/3/1/0	4/3/1/0
Average gap (%)	0,53	0,53
Maximum gap (%)	1,11	1,12

	*ng*6		*ng*7	
	sb5	sb10	sb5	sb10
Num. opt. sol. (all/a/b/c)	4/2/2/0	4/2/2/0	4/3/1/0	4/2/2/0
Num. best *lb* (all/a/b/c)	10/2/6/2	8/2/4/2	8/3/1/4	7/2/3/2
Average gap (%)	0,43	0,44	0,45	0,45
Maximum gap (%)	1,07	1,09	1,08	1,08

Comparing the number of optimal solutions, the k-loop and *ng*-relaxations are able to find about the same number of integer solutions. However, similar to the results in Section 4.5.4, k-loop solves more instances of groups with higher capacity (i.e. `egl-`*lm*`-a`) to optimality. On the other hand, looking at the number of best lower bounds among all relaxation with strong branching, *ng*6 and *ng*7 with five or ten candidates perform always better, resulting also in smaller average and maximum gaps. Overall, several

lower bounds are improved compared to the integer results without strong branching (`egl-e3-b`, `egl-e4-c`, `egl-s3-a`, and `egl-s4-a`).

4.5.6 New Best Solutions for `egl` and `bmcv` Instances

Compared to the best known results from the literature several lower bounds for both data sets were improved. In addition standard `egl` and `bmcv` instances, we used a dataset of large-scale `egl` instances which was proposed by (Brandão and Eglese, 2008) and contains instances with up to 255 nodes, 375 edges and 347 or 375 required edges. Tables 4.9, 4.12 and 4.13 summarize the results for the `egl` instances, while Tables 4.10 and 4.11 present results for the `bmcv` instances. Moreover, we made additional runs for `bmcv` instances with a small gap with those relaxations that gave the best lower bounds. Here, the time limit was set to twelve hours and the results can be found in Table 4.14. In the tables, values printed in bold indicate new best solutions.

New best lower bounds were calculated for all large-scale `egl` instances and five standard `egl` instances (`egl-e3-b`, `egl-e4-c`, `egl-s3-a`, `egl-s4-a`, `egl-s4-b`). The instance `egl-e2-b` is solved to optimality for the first time. The corresponding solution is shown in Section 4.8 of the Appendix.

For the previously 16 unsolved `bmcv` instances, we obtained either better lower bounds or optimal solutions in 13 cases. In detail, `C01` and `D24` were solved to optimality for the first time. Furthermore, better lower bounds were computed for `C09`, `C11`, `C12`, `C15`, `C23`, `D21`, `E01`, `E09`, `E15`, `E18` and `E23`. Bartolini *et al.* (2013b) already mentioned that the objective value for `bmcv` instances is always a multiple of five. Using the same argument, we prove optimality for `C12` and `E15`. In the end, twelve standard `egl` instances and twelve `bmcv` instances remain open.

4.6 Conclusion

In this work, different relaxations known from the node-routing context were adapted to solve the CARP with a branch-and-price approach. The adaptation to column generation-based approaches that price out new CARP tours over the original graph is by no means trivial, but is however attractive because it offers the application of highly effective pricing procedures that exploit the sparsity of the CARP network. Exploiting sparsity results in, compared to standard node-routing problems, a more intricate branching scheme, which in turn complicates the pricing. In essence, the effective approach of Chapter 2 requires that the shortest-path pricing problem resulting from a relaxation must be able to handle two sets of tasks: One set $\mathcal{T}^E$ models elementary routes and the other set $\mathcal{T}^B$ incorporates non-follower constraints implied by the branching scheme. While for $\mathcal{T}^E$ any relaxation of elementary routes is applicable, routes must be exactly 2-loop-free regarding to tasks in $\mathcal{T}^B$.

First, we have adapted the *ng*-route relaxations (Baldacci *et al.*, 2011b) leading to combined *ng*-route 2-loop-free relaxations. These were compared with the combined $(k, 2)$-loop-free relaxations presented in Chapter 3.

Second, we integrated acceleration techniques for the heuristic and exact solution of the pricing problems. In particular, bi-directional labeling (Righini and Salani, 2006) and bounding (Baldacci *et al.*, 2009) techniques were modified to fit with all relaxations.

Third, we presented a comprehensive computational study where the performance of the acceleration techniques, the quality of the bounds (lower bounds at the root node and over time in branch-and-price), and the overall performance of different branch-and-price algorithms were analyzed. Moreover, we tried to characterize which type of relaxation and acceleration technique is best suited to solve a specific group of instances. The standard instances `egl` of Eglese and Li (1992) and `bmcv` of Beullens *et al.* (2003) were used for that purpose. In summary, reasonable parameters are $k \in \{2, 3, 4\}$ for $(k, 2)$-loop elimination and $n_{ng} \in \{5, 6, 7\}$ for the maximum size of neighborhoods in ng-route relaxations. Bounding with the 2-loop-free relaxation is generally sufficient, stronger relaxations do not pay off. For the entire branch-and-price, bi-directional labeling alone accelerates better than bounding alone, but the combination of both is often even more effective providing acceleration factors of approximately four for ng-route relaxations and $(3, 2)$-loop elimination, and factor eight for $(4, 2)$-loop elimination. The study of lower bounds provided by the linear relaxations with $(k, 2)$-loop elimination and ng-routes shows that neither relaxation outperforms the others on all instances. Concerning groups of instances, k-loop-free relaxations often work better for instances utilizing fewer vehicles, higher capacities, and relatively long routes. The opposite is true for ng-route relaxations working best when solutions comprise more vehicles with relatively shorter routes.

Overall, the relaxations with loop elimination for $k = 3$ and $k = 4$ as well as the use of the ng-route relaxations outperformed the already remarkable results with elementary routes presented by Bartolini *et al.* (2013b) and with the pure 2-loop-free relaxation presented in Chapter 2. The different branch-and-price algorithms delivered 19 new best lower bounds of the `egl` and `bmcv` benchmark sets, and improved all lower bounds for the twelve large-scale `egl` instances by Martinelli *et al.* (2013). Finally, for 29 previously open instances, one of the standard `egl` and four of the `bmcv` benchmark set are solved to optimality for the first time.

4.7 Bibliography

Achterberg, T., Koch, T., and Martin, A. (2005). Branching rules revisited. *Operations Research Letters*, **33**(1), 42–54.

Ahr, D. (2004). *Contributions to Multiple Postmen Problems*. Phd dissertation, Department of Computer Science, Heidelberg University, Heidelberg, Germany.

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., Mingozzi, A., and Roberti, R. (2011a). Dynamic ng-path relaxation. Tromso, Norway. Route Conference 2011.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011b). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Bartolini, E., Cordeau, J.-F., and Laporte, G. (2013a). An exact algorithm for the capacitated arc routing problem with deadheading demand. *Operations Research*, **61**(2), 315–327.

Bartolini, E., Cordeau, J.-F., and Laporte, G. (2013b). Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming*, **137**(1-2), 409–452.

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

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

Belenguer, J. M., Benavent, E., and Irnich, S. (2013). The capacitated arc routing problem: Exact algorithms. In Corberán and Laporte (2013), chapter 9. (In preparation.).

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

Benavent, E., Campos, V., Corberán, Á., and Mota, E. (1992). The capacitated arc routing problem: Lower bounds. *Networks*, **22**, 669–690.

Beullens, P., Muyldermans, L., Cattrysse, D., and van Oudheusden, D. (2003). A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, **147**(3), 629–643.

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, Á. and Laporte, G., editors (2013). *Arc Routing: Problems, Methods and Applications*. MOS-SIAM Series on Optimization. SIAM, Philadelphia. (In preparation.).

Corberán, Á. and Prins, C. (2010). Recent results on arc routing problems: An annotated bibliography. *Networks*, **56**(1), 50–69.

Desaulniers, G., Desrosiers, J., and Solomon, M. (2002). Accelerating strategies in column generation for vehicle routing and crew scheduling problems. In C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, Operations Research/Computer Science Interfaces Series, chapter 14, pages 309–324. Kluwer, Boston.

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.

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

Fu, H., Mei, Y., Tang, K., and Zhu, Y. (2010). Memetic algorithm with heuristic candidate list strategy for capacitated arc routing problem. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

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

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

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.

Kohl, N. (1995). *Exact methods for Time Constrained Routing and Related Scheduling Problems*. Dissertation, Department of Mathematical Modelling, Technical University of Denmark.

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.

Larsen, J. (1999). *Parallelization of the Vehicle Routing Problem with Time Windows*. Ph.D. thesis, Department of Mathematical Modelling, Technical University of Denmark.

Letchford, A. N. (1997). *Polyhedral results for some arc routing problems*. Phd dissertation, Department of Management Science, Lancaster University.

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., Poggi 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.

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

Martinelli, R., Pecin, D., Poggi de Aragão, M., and Longo, H. (2011). A branch-cut-and-price algorithm for the capacitated arc routing problem. In P. Pardalos and S. Rebennack, editors, *Experimental Algorithms*, volume 6630 of *Lecture Notes in Computer Science*, pages 315–326. Springer Berlin / Heidelberg.

Martinelli, R., Poggi de Aragão, M., and Subramanian, A. (2013). Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, **40**(8), 2145–2160.

McGill, R., Tukey, J. W., and Larsen, W. A. (1978). Variations of box plots. *The American Statistician*, **32**(1), 12–16.

Mei, Y., Tang, K., and Yao, X. (2009). A global repair operator for capacitated arc routing problem. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, **39**(3), 723–734.

Muyldermans, L. (2012). Personal Communication.

Polacek, M., Doerner, K., Hartl, R., and Maniezzo, V. (2008). A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, **14**, 405–423.

Prins, C. (2013). The capacitated arc routing problem: Heuristics. In Corberán and Laporte (2013), chapter 7. (In preparation.).

Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, **3**(3), 255–273.

Santos, L., Coutinho-Rodrigues, J., and Current, J. R. (2010). An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B*, **44**(2), 246 – 266.

Tang, K., Mei, Y., and Yao, X. (2009). Memetic algorithm with extended neighborhood search for capacitated arc routing problems. *Evolutionary Computation, IEEE Transactions on*, **13**(5), 1151–1166.

Wøhlk, S. (2008). A decade of capacitated arc routing. In R. Sharda, S. Voß, B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 29–48. Springer.

4.8 Appendix

Tables

Linear Relaxation Results The Tables 4.6–4.8 present the linear relaxation results for the `egl` and `bmcv` instances. The meaning of the table entries are as follows:

instance	name of the instance (for `egl` instances the prefix `egl-` is omitted for the sake of brevity)
ub_{best} or $\underline{opt}$	the best known upper bound (not underlined) or the optimum (underlined)
lb	lower bound provided by the respective linear relaxation (rounded up to the next integer)
gap	absolute gap, i.e., the difference $ub_{best} - lb$ or $opt - lb$
time	computation time in seconds (rounded up to the next integer)

Table 4.6: Linear Relaxation Results for `egl` Instances

instance	ub_{best} or opt	2-loop			3-loop			4b2-loop			4b3-loop			*ng5*			*ng6*			*ng7*		
		lb	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time
`e1-a`	3548	3545	3	41	3546	2	75	3546	2	111	3546	2	322	3548	0	261	3548	0	269	3548	0	262
`e1-b`	4498	4464	34	13	4465	33	44	4467	31	36	4467	31	40	4470	28	70	4474	24	98	4474	24	83
`e1-c`	5595	5523	72	10	5528	67	28	5532	63	21	5532	63	26	5544	51	135	5542	53	187	5545	50	143
`e2-a`	5018	4996	22	24	4996	22	91	4999	19	317	4999	19	419	5000	18	1054	5001	17	2341	5001	17	2975
`e2-b`	6317	6273	44	19	6280	37	65	6283	34	86	6283	34	83	6292	25	259	6299	18	475	6299	18	621
`e2-c`	8335	8202	133	9	8227	108	22	8263	72	37	8263	72	29	8271	64	70	8271	64	67	8271	64	78
`e3-a`	5898	5894	4	32	5895	3	181	5895	3	300	5895	3	332	5896	2	1069	5896	2	1235	5896	2	4401
`e3-b`	7775	7684	91	20	7699	76	57	7704	71	82	7704	71	94	7712	63	301	7712	63	408	7712	63	385
`e3-c`	10292	10145	147	9	10176	116	25	10182	110	32	10182	110	39	10184	108	63	10184	108	70	10184	108	65
`e4-a`	6444	6389	55	39	6389	55	234	6389	55	265	6389	55	377	6392	52	743	6392	52	1477	6392	52	1351
`e4-b`	8961	8852	109	18	8862	99	59	8865	96	78	8865	96	112	8876	85	176	8881	80	240	8882	79	207
`e4-c`	11529	11411	118	12	11438	91	40	11463	66	40	11463	66	74	11466	63	92	11467	62	106	11467	62	93
`s1-a`	5018	5011	7	234	5012	6	565	5013	5	1180	5013	5	1380	5015	3	5360	5015	3	8030	5015	3	14306
`s1-b`	6388	6370	18	219	6373	15	837	6376	12	1292	6376	12	1453	6377	11	4897	6378	10	10016	6377	11	6259
`s1-c`	8518	8418	100	76	8457	61	147	8468	50	123	8468	50	208	8478	40	2516	8487	31	3048	8487	31	2806
`s2-a`	9884	9791	93	238	9795	89	539	9795	89	936	9795	89	1666	9799	85	3154	9800	84	3124	9800	84	3169
`s2-b`	13100	12949	151	108	12955	145	238	12960	140	302	12960	140	535	12971	129	1276	12976	124	1554	12975	125	1733
`s2-c`	16425	16314	111	105	16332	93	131	16338	87	139	16338	87	193	16357	68	495	16358	67	544	16358	67	523
`s3-a`	10220	10144	76	294	10145	75	779	10145	75	4151	10145	75	3660	9799	85	10507	10151	69	9391	10151	69	12799
`s3-b`	13682	13598	84	168	13604	78	263	13605	77	566	13605	77	686	12971	129	1755	13620	62	2414	13620	62	2790
`s3-c`	17188	17058	130	67	17089	99	122	17090	98	133	17090	98	18	17112	76	359	17112	76	449	17113	75	370
`s4-a`	12268	12126	142	162	12129	139	500	12129	139	1353	12129	139	1620	12136	132	3323	12138	130	5285	12138	130	4726
`s4-b`	16283	16066	217	107	16071	212	285	16073	210	413	16073	210	768	16081	202	1038	16082	201	1943	16082	201	1661
`s4-c`	20481	20340	141	139	20362	119	265	20375	106	280	20375	106	457	20391	90	531	20392	89	498	20394	87	627

Table 4.7: Linear Relaxation Results for `bmcv` Instances, Subsets `C` and `E`

instance	ub_{best} or $\underline{opt}$	2-loop			3-loop			4b2-loop			4b3-loop			$ng5$			$ng6$			$ng7$		
		lb	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time
C01	4150	4086	64	19	4086	64	46	4089	61	53	4089	61	73	4097	53	80	4097	53	84	4098	52	91
C02	3135	3135	0	4	3135	0	10	3135	0	15	3135	0	28	3135	0	18	3135	0	19	3135	0	19
C03	2575	2529	46	3	2542	33	8	2546	29	10	2546	29	13	2549	26	25	2549	26	25	2549	26	27
C04	3510	3474	36	11	3474	36	24	3474	36	35	3474	36	43	3476	34	190	3476	34	270	3476	34	340
C05	5365	5320	45	4	5321	44	12	5323	42	13	5323	42	15	5323	42	21	5324	41	26	5324	41	29
C06	2535	2508	27	2	2509	26	7	2509	26	11	2509	26	12	2510	25	14	2510	25	16	2510	25	16
C07	4075	4019	56	4	4018	57	10	4019	56	16	4019	56	17	4020	55	33	4020	55	33	4020	55	37
C08	4090	4025	65	14	4026	64	29	4028	62	32	4028	62	45	4028	62	41	4028	62	43	4028	62	46
C09	5260	5219	41	18	5219	41	45	5220	40	53	5220	40	69	5223	37	80	5223	37	86	5223	37	87
C10	4700	4606	94	4	4614	86	8	4616	84	9	4616	84	12	4619	81	15	4619	81	16	4619	81	16
C11	4635	4571	64	27	4571	64	73	4571	64	105	4571	64	138	4572	63	92	4573	62	115	4573	62	112
C12	4240	4175	65	10	4175	65	27	4175	65	52	4175	65	51	4176	64	55	4176	64	57	4176	64	59
C13	2955	2907	48	2	2909	46	5	2909	46	7	2909	46	11	2910	45	8	2910	45	9	2910	45	8
C14	4030	3982	48	8	3985	45	15	3985	45	18	3985	45	28	3991	39	34	3991	39	35	3991	39	35
C15	4940	4887	53	67	4888	52	123	4890	50	172	4890	50	216	4892	48	186	4892	48	198	4892	48	204
C16	1475	1470	5	2	1470	5	8	1470	5	58	1470	5	51	1470	5	12	1470	5	12	1470	5	12
C17	3555	3547	8	2	3548	7	6	3550	5	7	3550	5	9	3550	5	8	3550	5	7	3550	5	7
C18	5620	5557	63	59	5557	63	167	5557	63	241	5557	63	366	5557	63	664	5557	63	769	5557	63	1343
C19	3115	3074	41	12	3076	39	28	3076	39	34	3076	39	51	3089	26	147	3089	26	159	3089	26	182
C20	2120	2120	0	3	2120	0	8	2120	0	17	2120	0	33	2120	0	58	2120	0	66	2120	0	80
C21	3970	3956	14	11	3955	15	45	3955	15	117	3955	15	147	3957	13	185	3957	14	247	3957	13	265
C22	2245	2245	0	8	2245	0	22	2245	0	29	2245	0	40	2245	0	39	2245	0	45	2245	0	38
C23	4085	4032	53	27	4031	54	91	4032	53	128	4032	53	182	4039	46	483	4040	45	948	4041	44	2170
C24	3400	3377	23	24	3379	21	47	3379	21	89	3379	21	100	3380	20	129	3380	20	127	3380	20	145
C25	2310	2310	0	2	2310	0	4	2310	0	5	2310	0	4	2310	0	8	2310	0	8	2310	0	8
E01	4910	4857	53	20	4857	53	56	4858	52	73	4858	52	131	4858	52	70	4858	52	73	4858	52	75
E02	3990	3960	30	6	3960	30	17	3965	25	28	3965	25	34	3966	24	43	3966	24	47	3966	24	54
E03	2015	2015	0	3	2015	0	9	2015	0	20	2015	0	23	2015	0	14	2015	0	14	2015	0	14
E04	4155	4128	27	15	4128	28	41	4130	25	61	4130	25	73	4132	23	161	4132	23	313	4132	23	326
E05	4585	4562	23	8	4567	18	18	4572	13	21	4572	13	29	4577	8	32	4577	8	32	4577	8	32
E06	2055	2055	0	3	2055	0	5	2055	0	10	2055	0	12	2055	0	9	2055	0	9	2055	0	9
E07	4155	4068	87	6	4072	83	19	4078	77	25	4078	77	22	4084	71	63	4085	70	75	4085	70	80
E08	4710	4671	39	6	4674	36	16	4679	31	16	4679	31	19	4679	31	49	4679	31	49	4679	31	47
E09	5820	5771	49	129	5771	49	258	5772	48	288	5772	48	345	5774	46	374	5774	46	404	5774	46	460
E10	3605	3605	0	2	3605	0	8	3605	0	8	3605	0	9	3605	0	14	3605	0	13	3605	0	14
E11	4655	4630	25	21	4630	25	48	4630	25	88	4630	25	87	4632	23	98	4632	23	95	4632	23	108
E12	4180	4109	71	18	4110	70	40	4111	69	67	4111	69	58	4128	52	59	4128	52	60	4128	52	62
E13	3345	3309	36	5	3310	35	12	3311	34	12	3311	34	16	3312	33	15	3312	33	15	3312	33	16
E14	4115	4091	24	4	4090	25	10	4091	24	18	4091	24	24	4090	25	17	4090	25	16	4090	25	20
E15	4205	4182	23	45	4181	24	130	4182	23	241	4182	23	349	4185	20	703	4184	21	1875	4185	20	1689
E16	3775	3747	28	16	3749	26	32	3751	24	38	3751	24	53	3759	16	75	3760	15	72	3760	15	80
E17	2740	2740	0	2	2740	0	4	2740	0	4	2740	0	6	2740	0	10	2740	0	10	2740	0	10
E18	3835	3825	10	31	3825	10	61	3825	10	108	3825	10	139	3825	10	346	3826	10	1013	3826	9	1108
E19	3235	3204	31	40	3204	31	105	3205	30	150	3205	30	189	3212	23	524	3210	25	547	3212	23	832
E20	2825	2789	36	6	2792	33	21	2793	32	35	2793	32	33	2795	30	85	2796	29	114	2796	29	146
E21	3730	3725	5	14	3727	3	27	3727	3	37	3727	3	54	3727	3	132	3727	3	206	3727	3	291
E22	2470	2461	9	7	2465	5	13	2465	5	15	2465	5	18	2466	4	92	2466	4	91	2466	4	132
E23	3710	3684	26	194	3685	25	265	3686	24	449	3686	24	420	3689	21	1297	3689	21	1528	3689	21	1795
E24	4020	3992	28	25	3992	28	53	3996	24	67	3996	24	89	4001	19	158	4002	18	194	4002	18	220
E25	1615	1615	0	1	1615	0	3	1615	0	2	1615	0	2	1615	0	2	1615	0	2	1615	0	2

Table 4.8: Linear Relaxation Results for `bmcv` Instances, Subsets `D` and `F`

instance	ub_{best} or $\underline{opt}$	2-loop			3-loop			4b2-loop			4b3-loop			$ng5$			$ng6$			$ng7$		
		lb	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time	*lb*	gap	time
D01	3215	3215	0	18	3215	0	131	3215	0	480	3215	0	618	3215	0	613	3215	0	1163	3215	0	3097
D02	2520	2520	0	3	2520	0	14	2520	0	36	2520	0	25	2520	0	130	2520	0	192	2520	0	157
D03	2065	2065	0	7	2065	0	29	2065	0	142	2065	0	152	2065	0	374	2065	0	378	2065	0	432
D04	2785	2785	0	13	2785	0	81	2785	0	118	2785	0	56	2785	0	1074	2785	0	3141	2785	0	3264
D05	3935	3935	0	7	3935	0	24	3935	0	20	3935	0	33	3935	0	267	3935	0	357	3935	0	380
D06	2125	2125	0	6	2125	0	30	2125	0	107	2125	0	28	2125	0	305	2125	0	1502	2125	0	2345
D07	3115	3028	87	7	3031	84	34	3034	81	76	3034	81	65	3044	71	149	3046	69	185	3047	68	302
D08	3045	2982	63	13	2982	63	71	2982	63	163	2982	63	184	2983	62	196	2983	62	210	2989	56	973
D09	4120	4120	0	14	4120	0	98	4120	0	68	4120	0	129	4120	0	1332	4120	0	1779	4120	0	2517
D10	3340	3332	8	8	3332	9	39	3332	8	190	3332	8	331	3332	8	638	3332	8	467	3332	8	510
D11	3745	3745	0	19	3745	0	223	3745	0	134	3745	0	442	3745	0	3654	3745	0	5702	3745	0	7546
D12	3310	3310	0	21	3310	0	51	3310	0	51	3310	0	105	3310	0	550	3310	0	555	3310	0	509
D13	2535	2535	0	3	2535	0	14	2535	0	41	2535	0	58	2535	0	158	2535	0	146	2535	0	172
D14	3280	3272	8	16	3272	8	65	3272	8	93	3272	8	149	3272	8	941	3272	8	2009	3272	8	3393
D15	3990	3990	0	61	3990	0	525	-	-	-	3990	0	1955	3990	0	12964	3990	0	11232	3990	0	13578
D16	1060	1060	0	2	1060	0	11	1060	0	36	1060	0	54	1060	0	6362	1060	0	6342	1060	0	6332
D17	2620	2620	0	2	2620	0	7	2620	0	10	2620	0	13	2620	0	34	2620	0	34	2620	0	52
D18	4165	4165	0	163	-	-	-	4165	0	4753	-	-	-	4165	0	12988	4165	0	12566	4165	0	13484
D19	2400	2372	28	21	2372	28	114	2372	28	322	2372	28	234	2374	26	5503	2372	28	11085	2372	28	9077
D20	1870	1870	0	3	1870	0	22	1870	0	48	1870	0	113	1870	0	1219	1870	0	4440	1870	0	4687
D21	3050	2967	83	14	2968	82	173	2968	82	296	2968	82	433	2977	73	4547	2978	72	8785	2978	72	8672
D22	1865	1865	0	5	1865	0	52	1865	0	162	1865	0	75	1865	0	509	1865	0	570	1865	0	4311
D23	3130	3111	19	50	3110	20	2432	3110	20	12301	3110	20	12902	3115	15	14165	3113	17	7259	3113	17	7436
D24	2710	2666	44	21	2666	44	139	2667	43	301	2667	43	356	2668	42	9595	2666	44	14288	2665	45	5796
D25	1815	1815	0	2	1815	0	13	1815	0	25	1815	0	30	1815	0	238	1815	0	196	1815	0	424
F01	4040	4040	0	22	4040	0	90	4040	0	96	4040	0	197	4040	0	2056	4040	0	2468	4040	0	4164
F02	3300	3300	0	7	3300	0	36	3300	0	36	3300	0	67	3300	0	334	3300	0	569	3300	0	778
F03	1665	1665	0	3	1665	0	20	1665	0	35	1665	0	35	1665	0	336	1665	0	386	1665	0	669
F04	3485	3476	9	15	3476	9	69	3476	9	101	3476	9	130	3476	9	4160	3476	9	10239	3476	9	14253
F05	3605	3605	0	9	3605	0	39	3605	0	43	3605	0	101	3605	0	324	3605	0	359	3605	0	772
F06	1875	1875	0	5	1875	0	9	1875	0	13	1875	0	18	1875	0	307	1875	0	337	1875	0	401
F07	3335	3335	0	7	3335	0	29	3335	0	44	3335	0	62	3335	0	139	3335	0	205	3335	0	218
F08	3705	3690	15	11	3690	15	28	3691	14	64	3691	14	110	3691	14	166	3691	14	161	3691	14	189
F09	4730	4730	0	39	4730	0	285	4730	0	392	4730	0	514	4730	0	7971	4730	0	10819	4730	0	13926
F10	2925	2925	0	2	2925	0	11	2925	0	11	2925	0	14	2925	0	52	2925	0	71	2925	0	69
F11	3835	3835	0	24	3835	0	230	3835	0	283	3835	0	355	3835	0	7966	3835	0	10318	3835	0	14164
F12	3395	3386	9	44	3386	9	238	3386	9	564	3386	9	552	3386	9	571	3386	9	586	3386	9	607
F13	2855	2855	0	2	2855	0	7	2855	0	12	2855	0	22	2855	0	161	2855	0	254	2855	0	254
F14	3330	3330	0	11	3330	0	51	3330	0	92	3330	0	84	3330	0	298	3330	0	415	3330	0	701
F15	3560	3560	0	50	3560	0	114	3560	0	227	3560	0	208	3560	0	13269	3560	0	13720	3560	0	10980
F16	2725	2725	0	8	2725	0	29	2725	0	35	2725	0	50	2725	0	471	2725	0	972	2725	0	990
F17	2055	2055	0	2	2055	0	7	2055	0	14	2055	0	12	2055	0	30	2055	0	30	2055	0	30
F18	3075	3062	13	49	3062	14	189	3062	14	549	3062	14	424	3062	13	7053	3062	13	6769	3062	13	10977
F19	2525	2488	37	106	2488	37	408	2488	37	874	2488	37	796	2489	36	14373	2488	37	14211	2488	37	13580
F20	2445	2445	0	8	2445	0	48	2445	0	56	2445	0	58	2445	0	319	2445	0	757	2445	0	978
F21	2930	2930	0	12	2930	0	80	2930	0	69	2930	0	112	2930	0	1235	2930	0	4513	2930	0	3750
F22	2075	2075	0	5	2075	0	11	2075	0	13	2075	0	16	2075	0	341	2075	0	574	2075	0	670
F23	3005	2989	16	25	2988	17	167	2988	17	532	2988	17	303	2988	17	12553	2988	17	13703	2988	17	13069
F24	3210	3210	0	29	3210	0	181	3210	0	313	3210	0	354	3210	0	6722	3210	0	8095	3210	0	9017
F25	1390	1390	0	1	1390	0	3	1390	0	5	1390	0	6	1390	0	66	1390	0	74	1390	0	66

Integer Solution Results The Tables 4.9–4.11 present the integer results for the `egl` and `bmcv` instances. The meaning of the table entries are as follows:

instance	name of the instance (for `egl` instances the prefix `egl-` is omitted for the sake of brevity)
ub_{best} or $\underline{opt}$	the best known upper bound (not underlined) or the optimum (underlined)
lb^{tree}	lower bound provided by the branch-and-price algorithm within the time limit of 4 hours (rounded up to the next integer) 'OPT' indicates that the instance is solved to proven optimality within 4 hours $lb^{tree} = opt$ indicates that the gap was closed, but no integer optimal solution was computed within the time limit
lb^{best}_{own}	best lower bound over all relaxations tested here
Num. lb^{best}_{own}	number of instances for which the respective relaxation provided the best lower bound lb^{best}_{own}

Lower bounds written in **bold** indicate that that this bound is a new best bound exceeding the best known lower bounds from the literature. The upper bounds $ub = 11529$ for the instance `egl-e4-c` and $ub = 4650$ for the `bmcv` instance `E11` (written in **bold** also) result from new best integer solutions found with branch-and-price.

Table 4.9: Integer Results for `egl` Instances

instance	ub_{best} or $\underline{opt}$	2-loop lb^{tree}	3b2-loop lb^{tree}	4b2-loop lb^{tree}	4b3-loop lb^{tree}	$ng4$ lb^{tree}	$ng5$ lb^{tree}	$ng6$ lb^{tree}	$ng7$ lb^{tree}	lb^{best}_{own}
e1-a	$\underline{3548}$	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
e1-b	$\underline{4498}$	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
e1-c	$\underline{5595}$	5545	5551	5555	5554	5560	5571	5570	5572	5572
e2-a	$\underline{5018}$	OPT	OPT	OPT	OPT	OPT	5018	5018	5012	OPT
e2-b	$\underline{6317}$	6301	6301	6306	6305	6308	6311	**OPT**	**OPT**	**OPT**
e2-c	$\underline{8335}$	8242	8269	8303	8302	8300	8304	8315	8317	8317
e3-a	$\underline{5898}$	OPT	OPT	OPT	OPT	OPT	OPT	OPT	5898	OPT
e3-b	7775	7730	7735	7732	7733	7734	**7741**	7737	7740	**7741**
e3-c	10292	10191	10220	10226	10225	10226	10228	10228	10229	10229
e4-a	6444	6408	6405	6399	6399	6398	6399	6399	6398	6408
e4-b	8961	8892	8899	8900	8897	8905	8908	8913	8910	8913
e4-c	11529	11456	11488	11502	11499	11499	11500	11502	11502	11502
s1-a	$\underline{5018}$	OPT	OPT	OPT	OPT	5018	5018	5018	5015	OPT
s1-b	$\underline{6388}$	6386	OPT	OPT	OPT	6384	6384	6385	6383	OPT
s1-c	$\underline{8518}$	8440	8476	8500	8499	8501	8504	8509	8507	8509
s2-a	9884	9805	9806	9804	9803	9807	9806	9806	9808	9808
s2-b	13100	12970	12978	12982	12980	12991	12991	12994	12994	12994
s2-c	$\underline{16425}$	16351	16377	16380	16379	16393	16392	16393	16393	16393
s3-a	10220	10160	10154	10150	10149	10153	10153	10154	10152	10160
s3-b	13682	13630	13629	13627	13625	13637	13640	13644	13640	13644
s3-c	$\underline{17188}$	17096	17122	17125	17123	17138	17143	17142	17141	17143
s4-a	12268	12149	12147	12142	12141	12150	**12151**	**12151**	12150	**12151**
s4-b	16283	16104	16106	16105	16104	**16113**	16111	16111	16108	**16113**
s4-c	20481	20374	20397	20406	20405	20418	20420	20422	20423	20423
Num. lb^{best}_{own}		7	6	7	6	6	6	12	11	

Table 4.10: Integer Results for `bmcv` Instances, Subsets `C` and `E`

instance	ub_{best} or $\underline{opt}$	2-loop lb^{tree}	3b2-loop lb^{tree}	4b2-loop lb^{tree}	4b3-loop lb^{tree}	ng5 lb^{tree}	ng6 lb^{tree}	ng7 lb^{tree}	lb^{best}_{own} lb^{best}_{own}
C01	4150	4144	4140	4140	4138	4143	**4145**	4144	**4145**
C02	3135	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C03	2575	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C04	3510	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C05	5365	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C06	2535	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C07	4075	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C08	4090	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C09	5260	5244	5242	5242	5241	**5245**	**5245**	**5245**	**5245**
C10	4700	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C11	4635	4608	4608	4607	4604	4609	**4611**	4609	**4611**
C12	4240	4234	4231	4226	4225	4233	4232	4232	4234
C13	2955	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C14	4030	4010	4021	4024	4019	OPT	OPT	OPT	OPT
C15	4940	4918	4915	4916	4914	4918	4918	4918	4918
C16	1475	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C17	3555	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C18	5620	5570	5568	5563	5562	5564	5562	5562	5570
C19	3115	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C20	2120	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C21	3970	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C22	2245	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C23	4085	4073	4072	4069	4070	4073	4068	4058	4073
C24	3400	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
C25	2310	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
Num lb^{best}_{own}		21	17	17	17	21	22	20	
E01	4910	4898	4896	4896	4893	4898	4897	4897	4898
E02	3990	3971	3985	OPT	OPT	OPT	OPT	OPT	OPT
E03	2015	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
E04	4155	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
E05	4585	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
E06	2055	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
E07	4155	4137	4149	OPT	OPT	OPT	OPT	OPT	OPT
E08	4710	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
E09	5820	5802	5800	5798	5797	5802	5802	5802	5802
E10	3605	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
E11	4650	4650	OPT	4650	4650	4650	OPT	OPT	OPT
E12	4180	4167	4169	4170	4166	4178	4177	4179	4179
E13	3345	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
E14	4115	4108	OPT	OPT	OPT	OPT	4111	OPT	OPT
E15	4205	4199	4196	4194	4192	4197	4192	4193	4199
E16	3775	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
E17	2740	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
E18	3835	3825	3825	3825	3825	3826	3831	**3832**	**3832**
E19	3235	OPT	OPT	OPT	3235	3235	3235	3235	OPT
E20	2825	2815	2820	OPT	OPT	OPT	OPT	OPT	OPT
E21	3730	3730	3730	3730	3730	3730	**OPT**	**OPT**	**OPT**
E22	2470	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
E23	3710	3704	3703	3699	3697	**3707**	3704	3701	**3707**
E24	4020	OPT	4020	OPT	4020	OPT	OPT	OPT	OPT
E25	1615	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
Num lb^{best}_{own}		16	14	17	15	19	18	21	

Table 4.11: Integer Results for `bmcv` Instances, Subsets `D` and `F`

instance	ub_{best} or $\underline{opt}$	2-loop lb^{tree}	3b2-loop lb^{tree}	4b2-loop lb^{tree}	4b3-loop lb^{tree}	ng5 lb^{tree}	ng6 lb^{tree}	ng7 lb^{tree}	lb^{best}_{own} lb^{best}_{own}
D01	3215	OPT	OPT	OPT	3215	OPT	OPT	OPT	OPT
D02	2520	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
D03	2065	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
D04	2785	OPT	OPT	OPT	OPT	OPT	2785	2785	OPT
D05	3935	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
D06	2125	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
D07	3115	3108	3102	3098	3092	3098	3090	3082	3108
D08	3045	OPT	3041	3027	3022	3030	3027	3004	OPT
D09	4120	OPT	OPT	OPT	OPT	OPT	OPT	4120	OPT
D10	3340	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
D11	3745	3745	OPT	OPT	3745	3745	3745	3745	OPT
D12	3310	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
D13	2535	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
D14	3280	3280	OPT	OPT	OPT	OPT	OPT	OPT	OPT
D15	3990	OPT	OPT	-	3990	3990	3990	3990	OPT
D16	1060	OPT	OPT	OPT	OPT	OPT	OPT	1060	OPT
D17	2620	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
D18	4165	OPT	-	4165	-	4165	4165	4165	OPT
D19	2400	OPT	OPT	OPT	OPT	2376	2373	2373	OPT
D20	1870	OPT	OPT	OPT	OPT	1870	1870	1870	OPT
D21	3050	3005	2988	2982	2980	2983	2981	2981	3005
D22	1865	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
D23	3130	3126	3114	3111	3111	3115	3113	3113	3126
D24	2710	2704	2691	2679	2669	2669	2666	2666	2704
D25	1815	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
Num lb^{best}_{own}		23	19	18	16	15	14	12	
F01	4040	OPT	OPT	OPT	OPT	OPT	OPT	4040	OPT
F02	3300	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F03	1665	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F04	3485	OPT	OPT	OPT	3485	3483	3477	3476	OPT
F05	3605	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F06	1875	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F07	3335	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F08	3705	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F09	4730	OPT	OPT	4730	4730	4730	4730	4730	OPT
F10	2925	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F11	3835	OPT	OPT	OPT	OPT	3835	3835	3835	OPT
F12	3395	OPT	3395	3392	3392	3392	3390	3390	OPT
F13	2855	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F14	3330	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F15	3560	OPT	3560	3560	OPT	3560	3560	3560	OPT
F16	2725	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F17	2055	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F18	3075	3065	3065	3065	3065	3062	3062	3062	3065
F19	2525	2515	2515	2514	2511	2489	2489	2488	2515
F20	2445	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F21	2930	OPT	OPT	OPT	2930	OPT	2930	OPT	OPT
F22	2075	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
F23	3005	3003	2998	2994	2996	2989	2989	2989	3003
F24	3210	OPT	OPT	OPT	OPT	3210	3210	3210	OPT
F25	1390	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
Num lb^{best}_{own}		25	22	20	19	16	15	15	

The Table 4.12 presents the integer results for the large-scale `egl` instances. The meaning of the table entries are as follows:

instance	name of the instance
ub_{best}	the best known upper bound At the time of writing the best upper bounds ub were computed by Martinelli *et al.* (2013).
lb^{tree}	lower bound provided by the branch-and-price algorithm within the time limit of 10 hours (rounded up to the next integer)

Lower bounds written in **bold** indicate that that this bound is a new best bound exceeding the best known lower bounds from the literature.

Table 4.12: Integer Results for Large-Scale `egl` Instances

instance		2-loop	2-loop scaling 50
	ub_{best}	lb^{tree}	lb^{tree}
`egl-g1-a`	1,004,864	974,383	**976,907**
`egl-g1-b`	1,129,937	1,092,760	**1,093,884**
`egl-g1-c`	1,262,888	1,211,590	**1,212,151**
`egl-g1-d`	1,398,958	1,341,370	**1,341,918**
`egl-g1-e`	1,543,804	1,481,500	**1,482,176**
`egl-g2-a`	1,115,339	**1,069,536**	1,067,262
`egl-g2-b`	1,226,645	1,184,230	**1,185,221**
`egl-g2-c`	1,371,004	1,308,960	**1,311,339**
`egl-g2-d`	1,509,990	1,445,870	**1,446,680**
`egl-g2-e`	1,659,217	1,580,030	**1,581,459**

The Table 4.13 presents the integer results for strong branching using the standard `egl` instances. The meaning of the table entries are as follows:

instance	name of the instance
ub_{best} or $\underline{opt}$	the best known upper bound (not underlined) or the optimum (underlined)
lb^{tree}	lower bound provided by the branch-and-price algorithm within the time limit of 4 hours (rounded up to the next integer) 'OPT' indicates that the instance is solved to proven optimality within 4 hours $lb^{tree} = opt$ indicates that the gap was closed, but no integer optimal solution was computed within the time limit
ub_{own}^{best}	best lower bound computed in this analysis

Table 4.13: Integer Solutions with Strong Branching for `eg1` Instances

instance	ub_{best} or *opt*	2-loop	2-loop sb5	2-loop sb10	3-loop	3-loop sb5	3-loop sb10	4b2-loop	4b2-loop sb5	4b2-loop sb10	ng6	ng6 sb5	ng6 sb10	ng7	ng7 sb5	ng7 sb10	lb^{best}_{own}
e1-a	3548	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
e1-b	4498	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
e1-c	5595	5545	5546	5544	5551	5551	5551	5555	5555	5554	5570	5573	5571	5572	5570	5569	5573
e2-a	5018	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	5018	5017	5016	5012	OPT	5017	OPT
e2-b	6317	6301	6301	6301	6301	6301	6301	6306	6306	6305	OPT	**OPT**	**OPT**	OPT	6317	**OPT**	**OPT**
e2-c	8335	8242	8256	8262	8269	8274	8279	8303	8307	8309	8315	8318	8319	8317	8317	8319	8319
e3-a	5898	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	5898	OPT	OPT	OPT
e3-b	7775	7730	7738	7738	7735	7742	7743	7732	7736	7738	7737	7742	**7744**	7740	7738	7737	**7744**
e3-c	10292	10191	10196	10202	10220	10224	10228	10226	10229	10236	10229	10234	10236	10229	10231	10234	10236
e4-a	6444	6408	6408	6408	6405	6404	6402	6399	6397	6395	6399	6398	6398	6398	6396	6397	6408
e4-b	8961	8892	8897	8896	8899	8912	8915	8900	8911	8911	8910	8919	8919	8910	8914	8918	8919
e4-c	**11529**	11456	11458	11461	11488	11493	11494	11502	11506	**11512**	11501	11504	11504	11501	11503	11502	**11512**
s1-a	5018	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	5018	5018	5018	5018	5015	5014	5013	OPT
s1-b	6388	6386	6382	6380	OPT	6387	6386	OPT	6386	6384	6382	6381	6382	6383	6377	6378	6387
s1-c	8518	8440	8439	8438	8476	8477	8482	8500	8497	8496	8507	8505	8504	8507	8505	8505	8505
s2-a	9884	9805	9807	9812	9806	9805	9805	9804	9803	9802	9806	9806	9806	9808	9804	9805	9812
s2-b	13100	12970	12972	12972	12978	12978	12978	12982	12981	12981	12994	12993	12992	12994	12992	12993	12993
s2-c	16425	16351	16352	16352	16377	16376	16376	16380	16380	16379	16393	16391	16389	16393	16392	16390	16392
s3-a	10220	10160	10163	**10165**	10154	10152	10153	10150	10148	10148	10154	10153	10153	10152	10152	10151	**10165**
s3-b	13682	13630	13630	13630	13629	13628	13627	13627	13626	13624	13642	13640	13638	13640	13638	13637	13640
s3-c	17188	17096	17098	17098	17122	17123	17122	17125	17125	17125	17143	17139	17137	17141	17141	17139	17141
s4-a	12268	12149	12150	**12153**	12147	12145	12146	12142	12139	12137	12150	12150	12148	12150	12146	12147	**12153**
s4-b	16283	16104	16104	16106	16106	16107	16107	16105	16106	16106	16111	**16109**	16106	16108	16107	16107	**16109**
s4-c	20481	20374	20377	20376	20397	20398	20397	20406	20408	20406	20423	20421	20418	20423	20423	20418	20423

Finally, the Table 4.14 presents the integer results for `bmcv` instances with twelve hour computation time. The meaning of the table entries are as follows:

instance	name of the instance
relaxation(s)	the relaxation used that provided the corresponding lower bound lb^{tree}
lb^{tree}	lower bound provided by the branch-and-price algorithm within the time limit of 12 hours (rounded up to the next integer) 'OPT' indicates that the instance is solved to proven optimality within 12 hours in that case the objective value is given in parentheses
remark	indicates if either a new best lower bound is provided or optimality is proven by matching with an upper bound from the literature Note, that objective values are multiples of five

Table 4.14: Integer Solutions with long computation time for `bmcv` Instances

instance	relaxation(s)	lb^{tree}	remark
C01	ng6	**OPT (4150)**	
C11	ng6	**4617**	new best lb
C12	2-loop, ng5	**4239**	opt proven by matching with ub
C15	ng7	**4923**	new best lb
C23	ng5	**4078**	new best lb
E01	2-loop, ng5	**4903**	new best lb
E09	ng6, ng7	**5809**	new best lb
E15	2-loop	**4202**	opt proven by matching with ub
D21	2-loop	**3011**	new best lb
D24	2-loop	**OPT (2710)**	

Best Known Lower and Upper Bounds

The Tables 4.15–4.17 list the best known lower and upper bounds for the standard and large-scale `egl` instances and the `bmcv` instances. The meaning of the table entries are as follows:

instance	name of the instance
lb_{best}	the best known lower bound
ub_{best}	the best known upper bound
opt	cost of an optimal solution
own	a bound or proof of optimality provided using results of the paper at hand

Note: if an instance is solved to optimality, we do not give a lower bound.

At the time of writing this paper, twelve of the standard and all twelve large-scale `egl` instances remain unsolved. For the `bmcv` benchmark set, seven `C`, two `D`, three `E`, and two `F` instances are open.

Table 4.15: Best Known Bounds for the `egl` Instances

instance	lb_{best}	computed by	ub_{best}	computed by	*opt*	proved by
`egl-e1-a`			3548	Lacomme *et al.* (2001)	3548	Longo *et al.* (2006)
`egl-e1-b`			4498	Lacomme *et al.* (2001)	4498	Baldacci and Maniezzo (2006)
`egl-e1-c`			5595	Lacomme *et al.* (2001)	5595	Bartolini *et al.* (2013b)
`egl-e2-a`			5018	Lacomme *et al.* (2001)	5018	Baldacci and Maniezzo (2006)
`egl-e2-b`			6317	Brandão and Eglese (2008)	6317	own
`egl-e2-c`			8335	Brandão and Eglese (2008)	8335	Bartolini *et al.* (2013b)
`egl-e3-a`			5898	Lacomme *et al.* (2001)	5898	Longo *et al.* (2006)
`egl-e3-b`	7744	own	7775	Polacek *et al.* (2008)		
`egl-e3-c`	10244	Bartolini *et al.* (2013b)	10292	Polacek *et al.* (2008)		
`egl-e4-a`	6408	Chapter 2	6444	Santos *et al.* (2010)		
`egl-e4-b`	8935	Bartolini *et al.* (2013b)	8961	Bartolini *et al.* (2013b)		
`egl-e4-c`	11512	own	11529	Chapter 3		
`egl-s1-a`			5018	Lacomme *et al.* (2001)	5018	Baldacci and Maniezzo (2006)
`egl-s1-b`			6388	Brandão and Eglese (2008)	6388	Bartolini *et al.* (2013b)
`egl-s1-c`			8518	Lacomme *et al.* (2001)	8518	Bartolini *et al.* (2013b)
`egl-s2-a`	9825	Bartolini *et al.* (2013b)	9884	Santos *et al.* (2010)		
`egl-s2-b`	13017	Bartolini *et al.* (2013b)	13100	Brandão and Eglese (2008)		
`egl-s2-c`			16425	Brandão and Eglese (2008)	16425	Bartolini *et al.* (2013b)
`egl-s3-a`	10165	own	10220	Santos *et al.* (2010)		
`egl-s3-b`	13648	Bartolini *et al.* (2013b)	13682	Polacek *et al.* (2008)		
`egl-s3-c`			17188	Bartolini *et al.* (2013b)	17188	Bartolini *et al.* (2013b)
`egl-s4-a`	12153	own	12268	Santos *et al.* (2010)		
`egl-s4-b`	16113	own	16283	Fu *et al.* (2010)		
`egl-s4-c`	20430	Bartolini *et al.* (2013b)	20481	Bartolini *et al.* (2013b)		
`egl-g1-a`	976907	own	1049708	Martinelli *et al.* (2013)		
`egl-g1-b`	1093884	own	1140692	Martinelli *et al.* (2013)		
`egl-g1-c`	1212151	own	1282270	Martinelli *et al.* (2013)		
`egl-g1-d`	1341918	own	1420126	Martinelli *et al.* (2013)		
`egl-g1-e`	1482176	own	1583133	Martinelli *et al.* (2013)		
`egl-g2-a`	1067262	own	1129229	Martinelli *et al.* (2013)		
`egl-g2-b`	1185221	own	1255907	Martinelli *et al.* (2013)		
`egl-g2-c`	1311339	own	1417145	Martinelli *et al.* (2013)		
`egl-g2-d`	1446680	own	1516103	Martinelli *et al.* (2013)		
`egl-g2-e`	1581459	own	1701681	Martinelli *et al.* (2013)		

Table 4.16: Best Known Bounds for the `bmcv` Instances, Subsets `C` and `E`

instance	lb_{best}	computed by	ub_{best}	computed by	*opt*	proved by
C01			4150	Beullens *et al.* (2003)	4150	own
C02			3135	Beullens *et al.* (2003)	3135	Beullens *et al.* (2003)
C03			2575	Beullens *et al.* (2003)	2575	Bartolini *et al.* (2013b)
C04			3510	Beullens *et al.* (2003)	3510	own
C05			5365	Brandão and Eglese (2008)	5365	Bartolini *et al.* (2013b)
C06			2535	Beullens *et al.* (2003)	2535	Bartolini *et al.* (2013b)
C07			4075	Beullens *et al.* (2003)	4075	Bartolini *et al.* (2013b)
C08			4090	Beullens *et al.* (2003)	4090	Bartolini *et al.* (2013b)
C09	5245	own	5260	Brandão and Eglese (2008)		
C10			4700	Brandão and Eglese (2008)	4700	Bartolini *et al.* (2013b)
C11	4617	own	4630	Mei *et al.* (2009)		
C12			4240	Beullens *et al.* (2003)	4240	own
C13			2955	Beullens *et al.* (2003)	2955	Bartolini *et al.* (2013b)
C14			4030	Beullens *et al.* (2003)	4030	Bartolini *et al.* (2013b)
C15	4923	own	4940	Beullens *et al.* (2003)		
C16			1475	Beullens *et al.* (2003)	1475	Bartolini *et al.* (2013b)
C17			3555	Beullens *et al.* (2003)	3555	Bartolini *et al.* (2013b)
C18	5580	Bartolini *et al.* (2013b)	5620	Santos *et al.* (2010)		
C19			3115	Beullens *et al.* (2003)	3115	Chapter 3
C20			2120	Beullens *et al.* (2003)	2120	Beullens *et al.* (2003)
C21			3970	Beullens *et al.* (2003)	3970	Chapter 3
C22			2245	Beullens *et al.* (2003)	2245	Beullens *et al.* (2003)
C23	4078	own	4085	Beullens *et al.* (2003)		
C24			3400	Beullens *et al.* (2003)	3400	Chapter 3
C25			2310	Beullens *et al.* (2003)	2310	Beullens *et al.* (2003)
E01	4903	own	4910	Brandão and Eglese (2008)		
E02			3990	Beullens *et al.* (2003)	3990	Bartolini *et al.* (2013b)
E03			2015	Beullens *et al.* (2003)	2015	Beullens *et al.* (2003)
E04			4155	Beullens *et al.* (2003)	4155	Bartolini *et al.* (2013b)
E05			4585	Brandão and Eglese (2008)	4585	Bartolini *et al.* (2013b)
E06			2055	Beullens *et al.* (2003)	2055	Beullens *et al.* (2003)
E07			4155	Beullens *et al.* (2003)	4155	Bartolini *et al.* (2013b)
E08			4710	Beullens *et al.* (2003)	4710	Bartolini *et al.* (2013b)
E09	5809	own	5820	Tang *et al.* (2009)		
E10			3605	Beullens *et al.* (2003)	3605	Beullens *et al.* (2003)
E11			4650	Chapter 3	4650	Chapter 3
E12			4180	Bartolini *et al.* (2013b)	4180	Bartolini *et al.* (2013b)
E13			3345	Beullens *et al.* (2003)	3345	Bartolini *et al.* (2013b)
E14			4115	Beullens *et al.* (2003)	4115	Bartolini *et al.* (2013b)
E15			4205	Santos *et al.* (2010)	4205	own
E16			3775	Beullens *et al.* (2003)	3775	Chapter 3
E17			2740	Beullens *et al.* (2003)	2740	Beullens *et al.* (2003)
E18			3835	Beullens *et al.* (2003)	3835	Chapter 3
E19			3235	Beullens *et al.* (2003)	3235	Chapter 3
E20			2825	Beullens *et al.* (2003)	2825	Chapter 3
E21			3730	Beullens *et al.* (2003)	3730	Bartolini *et al.* (2013b)
E22			2470	Beullens *et al.* (2003)	2470	Bartolini *et al.* (2013b)
E23			3710	Beullens *et al.* (2003)	3710	own
E24			4020	Beullens *et al.* (2003)	4020	Chapter 3
E25			1615	Beullens *et al.* (2003)	1615	Beullens *et al.* (2003)

Table 4.17: Best Known Bounds for the `bmcv` Instances, Subsets `D` and `F`

instance	lb_{best}	computed by	ub_{best}	computed by	*opt*	proved by
D01			3215	Beullens *et al.* (2003)	3215	Beullens *et al.* (2003)
D02			2520	Beullens *et al.* (2003)	2520	Beullens *et al.* (2003)
D03			2065	Beullens *et al.* (2003)	2065	Beullens *et al.* (2003)
D04			2785	Beullens *et al.* (2003)	2785	Beullens *et al.* (2003)
D05			3935	Beullens *et al.* (2003)	3935	Beullens *et al.* (2003)
D06			2125	Beullens *et al.* (2003)	2125	Beullens *et al.* (2003)
D07			3115	Beullens *et al.* (2003)	3115	Bartolini *et al.* (2013b)
D08			3045	Beullens *et al.* (2003)	3045	Chapter 3
D09			4120	Beullens *et al.* (2003)	4120	Beullens *et al.* (2003)
D10			3340	Beullens *et al.* (2003)	3340	Bartolini *et al.* (2013b)
D11			3745	Tang *et al.* (2009)	3745	Beullens *et al.* (2003)
D12			3310	Beullens *et al.* (2003)	3310	Beullens *et al.* (2003)
D13			2535	Beullens *et al.* (2003)	2535	Beullens *et al.* (2003)
D14			3280	Beullens *et al.* (2003)	3280	Chapter 3
D15			3990	Beullens *et al.* (2003)	3990	Beullens *et al.* (2003)
D16			1060	Beullens *et al.* (2003)	1060	Beullens *et al.* (2003)
D17			2620	Beullens *et al.* (2003)	2620	Beullens *et al.* (2003)
D18			4165	Beullens *et al.* (2003)	4165	Beullens *et al.* (2003)
D19			2400	Beullens *et al.* (2003)	2400	Chapter 3
D20			1870	Beullens *et al.* (2003)	1870	Beullens *et al.* (2003)
D21	3011	own	3050	Beullens *et al.* (2003)		
D22			1865	Beullens *et al.* (2003)	1865	Beullens *et al.* (2003)
D23			3130	Beullens *et al.* (2003)	3130	Chapter 3
D24			2710	Beullens *et al.* (2003)	2710	own
D25			1815	Beullens *et al.* (2003)	1815	Beullens *et al.* (2003)
F01			4040	Beullens *et al.* (2003)	4040	Beullens *et al.* (2003)
F02			3300	Beullens *et al.* (2003)	3300	Beullens *et al.* (2003)
F03			1665	Beullens *et al.* (2003)	1665	Beullens *et al.* (2003)
F04			3485	Beullens *et al.* (2003)	3485	Chapter 3
F05			3605	Beullens *et al.* (2003)	3605	Beullens *et al.* (2003)
F06			1875	Beullens *et al.* (2003)	1875	Beullens *et al.* (2003)
F07			3335	Beullens *et al.* (2003)	3335	Beullens *et al.* (2003)
F08			3705	Beullens *et al.* (2003)	3705	Chapter 3
F09			4730	Beullens *et al.* (2003)	4730	Beullens *et al.* (2003)
F10			2925	Beullens *et al.* (2003)	2925	Beullens *et al.* (2003)
F11			3835	Beullens *et al.* (2003)	3835	Beullens *et al.* (2003)
F12			3395	Beullens *et al.* (2003)	3395	Chapter 3
F13			2855	Beullens *et al.* (2003)	2855	Beullens *et al.* (2003)
F14			3330	Beullens *et al.* (2003)	3330	Beullens *et al.* (2003)
F15			3560	Beullens *et al.* (2003)	3560	Beullens *et al.* (2003)
F16			2725	Beullens *et al.* (2003)	2725	Beullens *et al.* (2003)
F17			2055	Beullens *et al.* (2003)	2055	Beullens *et al.* (2003)
F18	3065	Bartolini *et al.* (2013b)	3075	Beullens *et al.* (2003)		
F19	2515	Chapter 3	2525	Beullens *et al.* (2003)		
F20			2445	Beullens *et al.* (2003)	2445	Beullens *et al.* (2003)
F21			2930	Beullens *et al.* (2003)	2930	Beullens *et al.* (2003)
F22			2075	Beullens *et al.* (2003)	2075	Beullens *et al.* (2003)
F23			3005	Beullens *et al.* (2003)	3005	Chapter 3
F24			3210	Beullens *et al.* (2003)	3210	Beullens *et al.* (2003)
F25			1390	Beullens *et al.* (2003)	1390	Beullens *et al.* (2003)

Integer Solutions

In this section, new integer solutions are given. Note that in the following '=' indicates a service and '–' a deadheading. The terms *ub* and *opt* show the cost of the presented solution. 'load' is the demand served by the respective route.

New Best Known Solutions

`egl-e2-b` *opt* = 6317

veh 1	1–2–4–69–59–44=43–42=57=58=59–69–4–2–1 load 195
veh 2	1–2–4–5–7–8–9–10–11–12–76=20–18=15=17–15–14=13=16=12=11–10=9=8–7–5–4–2–1 load 199
veh 3	1–2–4–69=58=60–67–56=55–56–42–41–35–32=34–32=35–41–42–57–58–69–4–2–1 load 200
veh 4	1–2–4–5–7–8–9–10–11–12–76–20=19=18–72=73=74–73=71–72=18–20–76–12–11–10–9–8–7–5–4–2–1 load 200
veh 5	1–2–4–69–59–44–46–47–49–51–21–22–75=23=26–23=31=32=33=36–33–37–39–35=41–42–57–58–69–4–2–1 load 199
veh 6	1–2–4–69–58–60–62=63=65–63=64–63–62–66=68–66=62=60=61–60–58–69–4=2–1 load 200
veh 7	1=2=3–2–4=5–4–2–1 load 102
veh 8	1–2–4–69–59–44=45–46–47=49–50=19=21=51=53=52=54–52=50=49–47=48–47=46=44=59–69–4–2–1 load 197
veh 9	1–2–4–69–59–44–46–47–49=51–21=22–24=25=26=27–26=28–29=25=75=22=24=53–52–50–49–47–46–44–59–69–4–2–1 load 199
veh 10	1–2–4–5=7=8–9–10–11=59=69=4–2–1 load 188

`C01` *opt* = 4150

veh 1	40-44-46-47-38=36=37=1-5-4=19=42-40 load 300
veh 2	40-44-45-51-53-54-55-31-30-29=24=23=25=2-3=35=34-36-38-47-46-44-40 load 300
veh 3	40-44-45-63=12=11=65-66=64=67-64=68=65=66=20-43-40 load 300
veh 4	40=42=41=39=38=47=46=45=63=13-63-45=44-40 load 265
veh 5	40-44-45=51=48=49=50=52=53=14-15=56=54=52-50=32=33=34=36-38-39=42-40 load 295
veh 6	40-44-45-51=53=54=55=58=59=60=16=61=59-60=57=56-54-52-50-49-48=47-46=44-40 load 300
veh 7	40=44=43=20-43=40 load 135
veh 8	40-44-45-51-53-54-56-57=58-59-61=62=28-27=26=22-23=21-23=22=24=33-32-49-48-47-46-44-40 load 300
veh 9	40-44-46-47-48-49=32=30-31=28=27=29=30=31=55-54-53-51-45-44-40 load 295

`D24` *opt* = 2710

veh 1	49-7=10-13=12-5=3=30-3=29=53-55=61=59=58=57=60=56-1=2=54-55=28-62=61=60=54=55=53=4=3-4=2=6=21-6=5-12-13=48-47-49 load 585
veh 2	49-7=16=17=18=20=19=11=13=10=11=12=5=52=51-52=42=41=40=44=46=39=41-43=31-43=41-40=39=47-49 load 565
veh 3	49=47=46=45=66=68=8-9=50=14-50=15=9=8=7=49 load 315
veh 4	49-47=48=42=43=44=32-33=38=37=34-37=26=27=25=24=76-70-75-74-63=38=64-38=65=69=32-69=67=45-46-47-49 load 575

`E21` *opt* = 3730

veh 1	25–22–24–29–31–34=36=37–36=38=39–38=40=35=34–31–29–24–22–25 load 300
veh 2	25–26=28=14=15–14=16–18=20–18=17=19–17=16=18=21=23=22–25 load 295
veh 3	25–26–28–12–53=52=51–52=13=47=33=13=53–12=28–26–25 load 300
veh 4	25–26–28–12=53=54=55=52–55=56=57–50=9–50=57=10=56=54=11=14=22–25 load 300
veh 5	25=22=21–23=24=29=30–29=27=28–27=26=25 load 230
veh 6	25–22=24–29=31=32=42–32=33–13=12=11–12–28–26–25 load 300
veh 7	25–22–24–29–31–34–36–38=7–8–6–5–1=46=40–41=42=43=44=45=46=3–2=44–43=45=41=40–35=36–34=31–29–24–22–25 load 300

4.9 Bibliography

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

Bartolini, E., Cordeau, J.-F., and Laporte, G. (2013b). Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming*, **137**(1-2), 409–452.

Beullens, P., Muyldermans, L., Cattrysse, D., and van Oudheusden, D. (2003). A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, **147**(3), 629–643.

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

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

Fu, H., Mei, Y., Tang, K., and Zhu, Y. (2010). Memetic algorithm with heuristic candidate list strategy for capacitated arc routing problem. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

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.

Longo, H., Poggi 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.

Martinelli, R., Pecin, D., Poggi de Aragão, M., and Longo, H. (2011). A branch-cut-and-price algorithm for the capacitated arc routing problem. In P. Pardalos and S. Rebennack, editors, *Experimental Algorithms*, volume 6630 of *Lecture Notes in Computer Science*, pages 315–326. Springer Berlin / Heidelberg.

Martinelli, R., Poggi de Aragão, M., and Subramanian, A. (2013). Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, **40**(8), 2145–2160.

Mei, Y., Tang, K., and Yao, X. (2009). A global repair operator for capacitated arc routing problem. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, **39**(3), 723–734.

Polacek, M., Doerner, K., Hartl, R., and Maniezzo, V. (2008). A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, **14**, 405–423.

Santos, L., Coutinho-Rodrigues, J., and Current, J. R. (2010). An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B*, **44**(2), 246 – 266.

Tang, K., Mei, Y., and Yao, X. (2009). Memetic algorithm with extended neighborhood search for capacitated arc routing problems. *Evolutionary Computation, IEEE Transactions on*, **13**(5), 1151–1166.

Chapter 5

Park and Loop Delivery Problems

Claudia Bode

Abstract

Combining arc routing problems with problems of determining locations for depots results in location arc routing problems (LARP). Several ways of connecting depots and assigning routes to the depots are presented in the literature. This paper addresses the different types of LARP and classifies the literature according to these types. Park and loop is one of them, where routes have to be assigned to the depots in such a way that the start and end point of the routes is the same depot. Extending this problem by also allowing a service to be performed with a large transfer vehicle that connects the depots is called park and loop with curbline routes. For these two types of LARP, four mixed integer formulations are presented. The two models, each representing one of the types, differ in how they formulate feasible routes connecting the depots. While the first model uses generalized subtour elimination constraints, the second one uses flow variables. The quality of the formulations is tested in a computational study.

5.1 Introduction

Since their introduction by Golden and Wong (1981), (mixed) capacitated arc routing problems ((M)CARP) have received increasing attention over the last decades (see surveys (Dror, 2000; Corberán and Prins, 2010)). An optimal CARP solution consists of a set of cost-minimal feasible routes for a fleet of vehicles such that every edge with positive demand is serviced by exactly one vehicle and the capacity of each vehicle is met. Each route starts and ends at a specific depot node. In the mixed case, the underlying network is represented as a mixed graph in order to achieve more realistic models of real world problems (Belenguer *et al.*, 2006). Some authors relax the standard assumption of the CARP that a fixed depot location is required: This leads to a location arc routing problem (LARP) (Levy and Bodin, 1989).

This problem consists of simultaneously determining locations of depots and obtaining routes connected to the depots. Unfortunately, the same term LARP is used by several authors even though the ways of connecting routes to the depots and connecting the depots to each other differ.

Therefore, the first contribution of this paper is to classify the literature into the categories as defined by Bodin and Levy (2000) for mail delivery application. The second

contribution is to present a total of four mathematical formulations for the park and loop (PAL) problem, which is a variant of the LARP, and the combined park and loop with curbline routes (CurbPAL) problem, which is an extension of the PAL problem. Characteristic of the PAL problem is the combination of the routing of a transfer vehicle that takes the postman from the depot to parking lots and the routing of smaller delivery tours starting and ending at each parking lot. In the PAL problem, service is only allowed within routes starting at the parking lots. The extended variant – the CurbPAL problem – can be formulated to also allow service while the transfer vehicle connecting the parking lots is being driven. So far, the formulations presented in the literature either (*i*) do not describe the connections between the parking lots or (*ii*) do not have delivery routes starting and ending at the same parking lots. In this work, these points are integrated into a mathematical formulation for the first time. Two models for each problem (PAL and CurbPAL), based on flow formulations of Gouveia *et al.* (2010) and Hashemi Doulabi and Seifi (2013) are presented. These two models differ in the way they handle infeasible subtours of the transfer vehicle. The first model forbids such subtours by subsequently adding generalized subtour elimination constraints; the second model uses additional flow variables to ensure connectivity.

The rest of the paper is structured as follows: The description of different types of LARP and their classification into the categories of Bodin and Levy (2000) are presented in Section 5.2. The four models (two for the PAL and two for the extended PAL problem) are introduced in Section 5.3. In Section 5.4, computational results for the presented models are shown, and a conclusion is drawn in Section 5.5.

5.2 Classification of Location Arc Routing Problems

It is well known that the locations of depots have an influence on the costs of routes assigned to them (Salhi and Rand, 1989). Therefore, it is advantageous to simultaneously solve the problem of locating the depots and the routing problem, which is then called a location routing problem (LRP). The question is how to locate depots and assign routes to those depots in order to minimize the overall costs, which consist of depot and vehicle costs. The types of routes can be further divided into routes requiring a service at nodes (still called LRP) and routes performing a service along edges or arcs (LARP).

While the LRP is addressed extensively, only a few articles target the arc routing case. A survey on applications for mail delivery, winter gritting, and garbage collection problems can be found in (Assad and Golden, 1995). The first two papers studying the LARP adressed mail delivery for the Canada Post Corporation (Roy and Rousseau, 1989) and the United States Postal Service (Levy and Bodin, 1989). Later, Assad and Golden (1995) and Bodin and Levy (2000) observed that different ways of designing delivery routes exist, and hence of assigning routes to the located depots. In the following, we will present delivery types suggested by Bodin and Levy (2000) and classify articles on the LARP into these types.

Curbline and Dismount Route When delivering mail in a curbline route, the postman

starts at the depot and moves along streets where mail has to be delivered. He can fill mailboxes directly from his vehicle without getting out. The working time of the postman is completed when he returns to the depot. A similar mode is the dismount route, but in contrast to the curbline mode the postman has to leave his vehicle to post the mail. No parking lots have to be opened and only one type of vehicle is involved. This delivery mode can be modeled as a standard CARP, without the inclusion of any location aspects. Several exact and heuristic methods have been presented for this kind of problem and are surveyed in (Belenguer *et al.*, 2013) and (Prins, 2013).

Relay Box Routes The second delivery mode is called relay box routes, where a postman starts the delivery route with a full handcart at some point. He delivers mail until his handcart is empty, then walks to a relay box to refill his handcart and to continue his route. Subsequent refillings can be done at the same or at different relay boxes. There are no constraints on where the last delivery ends. The transfer from the depot to the starting point and from the end point back to the depot is done by some means of transportation determined by the postal service. Disregarding this transfer transport, only one type of vehicle is involved. The main aspect, as stated in (Roy and Rousseau, 1989), is that of tour length restrictions for the morning and afternoon tours, which can be seen as capacity restrictions of a specific tour (Assad and Golden, 1995). Additionally, Bodin and Levy (2000) mention that the mail delivery from the depot to the relay boxes is not considered simultaneously with the mail delivery (to customers), but can be modeled as a node routing problem with time windows. However, combining these two aspects without time windows and assuming that the locations of the relay boxes are not given in advance, the problem is to find these locations in order to meet the load capacity restrictions of the postman and to obtain his walking route and the filling route of the relay boxes simultaneously. Such problems can be modeled as LARPs.

There are some papers that can be classified in this problem category: Ghiani *et al.* (2001) consider the CARP with intermediate facilities. Applications are waste management problems, where one vehicle collects garbage and whenever the vehicle capacity is met, it has to be unloaded at a dump site or incinerator. Another application is road gritting, where intermediate facilities represent located boxes of salt, sand or chemicals. What these problems have in common is that a vehicle starts at the depot collecting or delivering goods and the vehicle has to be unloaded or replenished at the intermediate facilities, which represent relay boxes. After servicing all required edges, it returns to the depot. However, no routes for filling up the relay boxes are considered. Amaya *et al.* (2007, 2010) consider the mail delivery to the relay boxes and the routing of the postman simultaneously. They look at a road marking problem where a fleet of vehicles starts at the depot, marks required roads, and returns to the depot. A second type of vehicle also starts at the depot but moves to specific positions (the relay boxes) to replenish the first vehicle type with color for marking the roads. Salazar-Aguilar *et al.* (2013) extend

this problem to a synchronized arc and node routing problem by including time aspects. They consider the locations of refilling and the routes for refilling and marking simultaneously.

Park and Loop A third delivery mode is the park and loop mode. There, several types of vehicles for traversing streets and mail delivery are available. In the standard problem as proposed in (Bodin and Levy, 2000), a park and loop route consists of driving a transfer vehicle from the depot to a parking lot, getting off and walking to deliver mail to a set of street segments and returning to the parking lot. In the following, these loops are referred to as walking loops. Further, either the handcart is filled again with mail and a new walking loop starts, or the postman drives the vehicle to the next parking lot. When all mail has been distributed, the postman drives back to the depot. Possible extensions can be the use of several vehicle types for mail delivery within the walking loops. The main difference to the relay box mode is that the postman has to bring all mail for delivery by himself and he has to return to the parking lot at the end of each delivery route. Determining the locations of parking lots and obtaining the corresponding walking loops can be modeled as an LARP.

Levy and Bodin (1989) considered the delivery problem of the United States Postal Service, where each delivery loop has to start and end at the same depot. The presented heuristic follows the location-allocation-routing principle, where the location of depots and the allocation of street segments to a depot are evaluated according to small deadheading time, balanced workload of each partition, and minimal number of located depots. Amberg *et al.* (2000) transformed the problem into an arc-constraint minimum spanning tree with multiple centers and developed a tabu search algorithm to solve the problem. In contrast to the former approach, a heterogeneous fleet of vehicles is available to deliver mail. Several objective functions, such as minimizing the deadheading time, balancing the workload, and accounting for customer priority, are evaluated. Recently, Hashemi Doulabi and Seifi (2013) presented a flow formulation of the multi depot problem with homogeneous vehicles, where a predefined number of depots can be opened. The objective function takes into account service and deadheading costs, hiring costs for each vehicle in use, and dumping costs for open depots. To analyze the performance of their simulated annealing approach, a lower bounding model was presented.

What is common to all approaches considered so far are the missing interconnections of the depots by a transfer vehicle, which is part of the delivery mode proposed by Bodin and Levy (2000). Nevertheless, these problems can be seen as LARPs with park and loop characteristics because all vehicles of the walking loops have to return to their starting depots.

Combined Park and Loop with Curbline Mode The last delivery mode that Bodin and Levy (2000) mentioned is a combination of park and loop routes and curbline routes. There, the postman is able to deliver mail both when driving the transfer vehicle to a parking lot or when performing a walking loop starting and ending at a parking

lot. So far, no model has been presented in the literature that considers this kind of problem. As park and loop routes are part of the problem, again, an LARP model appears to be convenient.

Combined Relay Box Routes with Curbline Mode The combination of relay box routes and curbline routes is another possible delivery mode. Del Pia and Filippi (2006) consider a garbage collection problem with two types of collection vehicles. The capacity and the underlying street network differ for the smaller satellite vehicles and the larger compactor trucks. Both types of vehicles are able to collect garbage and whenever satellites and trucks meet at the same node at the same time, the smaller vehicle can dump its load into the container of the larger truck. The problem considered is that of simultaneously determining the dump locations and the routes for each vehicle.

5.3 Mail Delivery Routes with Park and Loop Characteristics

As pointed out in the previous section, no formulations representing the full problem specifications defined by Bodin and Levy (2000) for both the pure park and loop delivery mode and the combination with curbline delivery exist. Especially the aspect of connecting the parking lots by a transfer vehicle is omitted. This section will introduce four mixed integer programs, two modeling the park and loop mode and two formulations for the combined park and loop with curbline mode. The presented models are based on the multi-depot LARP defined in (Hashemi Doulabi and Seifi, 2013). They extend the flow formulation of the single-depot MCARP of Gouveia *et al.* (2010). However, the opened parking lots are not connected in their approach. The formulations shown below extend their model in order to integrate a transfer route starting and ending at a specific main-depot and visiting all open parking lots. For the first two formulations, the postman is not allowed to perform any service on edges or arcs while driving the transfer vehicle. In order to model combined park and loop and curbline routes, a service is also allowed with the transfer vehicle for the second two models by further modifications.

Some notation valid for all mathematical formulations is presented first. The (combined) park and loop model considered in this paper is defined on a mixed graph $G = (V, E \cup A')$ with node set V, edge set E, and arc set A'. With each edge or arc (i, j), a non-negative demand $q_{ij} \geq 0$ is associated. The set of required edges or arcs, i.e., $q_{ij} > 0$, is indicated by a subscript $\cdot_R$. A subset of nodes $D \subseteq V$ defines the possible locations of parking lots. The maximum number of parking lots to be used (= opend) is D_{max}. Whenever a parking lot $d \in D$ is open, costs OC_d occur. At each parking lot $d \in D$, a heterogeneous fleet K of $|K|$ different vehicles with capacity Q^k $k \in K$ is available to perform the service. The index l links a vehicle k to a parking lot $d \in D$. The set of link-indices l is defined as $L := \{1, \ldots, D_{max}\}$. Each vehicle k selected for servicing creates hiring costs λ^k. The transfer vehicle is an additional vehicle indexed by $k = 0$ and is stationed at a specific depot for the transfer vehicle the main-depot

$d_M \in D$. Whenever traversing an edge or arc (i, j) with vehicle $k \in K \cup \{0\}$, traversing costs c_{ij}^k occur. Service costs $c_{ij}^{serv,k}$ are associated with required edges or arcs (i, j) when the service is performed with vehicle $k \in K$. If it is also allowed to perform a service with the transfer vehicle, service costs $c_{ij}^{serv,0}$ occur.

If the multi-depot LARP is based on flow variables, then some modifications of the graph are necessary. Each edge $\{i, j\}$ is replaced by two opposite arcs (i, j) and (j, i). Therefore, the directed graph $G = (V, A'')$ consists of the node set V and the new arc set $A'' = A' \cup \{(i, j), (i, j) | \{i, j\} \in E\}$. To be able to use a parking lot also as an intermediate node of a walking loop, each node that represents a potential parking lot is duplicated and the set of duplicated parking lots is denoted by D'. New non-required arcs (d, d') and (d', d) with zero deadheading costs are added. The final graph representing the underlying network is then given by $G = (V \cup D', A)$ with $A = A'' \cup \{(d, d'), (d', d) | d \in D\}$.

Throughout the paper we will use the following standard notation. For a set $S \subseteq V$, we denote by $\delta(S)$ the set of edges with exactly one endpoint in S and by $\delta^+(S)$ and $\delta^-(S)$ the cut set of arcs leaving and entering set S, respectively. Similarly to the notation of required edges and arcs, we denote by $\delta_R(S) = E_R \cap \delta(S)$ and $\delta_R^*(S) = A_R \cap \delta^*(S)$ with $* \in \{+, -\}$ the set of required edges or arcs in the cut. To further shorten the notation, we will write $\delta(i)$ ($\delta^*(i)$ with $* \in \{+, -\}$) instead of $\delta(\{i\})$ ($\delta^*(\{i\})$ with $* \in \{+, -\}$).

5.3.1 Park and Loop Models

A park and loop solution consists of a set of opened parking lots, a transfer route connecting these parking lots and a set of tours starting and ending at each parking lot, while respecting the capacity restriction of each vehicle. Inspired by the simple cycle problem (Fischetti *et al.*, 2004) and the directed version of the prize collecting traveling salesman problem (Balas, 1989), the first of our models uses standard generalized subtour elimination constraints (SEC) for connecting the parking lots. However, because of the exponential number of constraints, the problem cannot be explicitly wrote down for realistic sized instances. The model must then be solved with cutting plane procedures to identify violated generalized SEC, which can easily be adapted to the park and loop case. The second of our models uses a flow formulation to connect the open parking lots, similarly to the single depot flow formulation of Gavish and Graves (1978) for the traveling salesman problem. The advantage of this formulation is the linear number of constraints needed to eliminate subtours. However, additional flow variables have to be introduced into the formulation.

Park and Loop Model with Generalized SEC

To give a mathematical formulation, some further notation is needed: Let SP_{ij} be the cost of the shortest deadheading path between i and j for the transfer vehicle. The costs of the shortest path are determined by summing up the corresponding traversing costs. Variables $x_{ij}^{k,l}$ take value 1 if arc (i, j) is serviced by vehicle k that is linked to a parking lot by label l, and 0 otherwise. Variables $y_{ij}^{k,l}$ count the number of times an arc (i, j) is deadheaded by vehicle k linked to a parking lot by l. In order to include the hiring

cost of vehicles that perform a service, variables DC_d are introduced. For each potential location of a parking lot, a variable z_d exists and takes value 1 if the parking lot is used, and 0 otherwise. The tour of a vehicle k that stops at parking lot l is indicated by variables T_d^l. These variables take value 1 if node d is selected as a parking lot and label l is assigned to this place. Flow variables $f_{ij}^{k,l}$ for every arc (i,j) are used to ensure the connectivity of the tour of vehicle k linked to a parking lot by l within the walking loops. Finally, variables r_{ij} define the route of the transfer vehicle and take value 1 if parking lot i is connected with parking lot j. The (PAL) model then reads:

$$\text{(PAL)}\ \min \sum_{\forall l \in L} \left(\sum_{k \in K} \left(\sum_{(i,j) \in A_R} c_{ij}^{serv,k} x_{ij}^{k,l} + \sum_{(i,j) \in A} c_{ij}^{k} y_{ij}^{k,l} \right) \right) + \sum_{i \in D'} \sum_{j \in D' \setminus \{i\}} SP_{ij} r_{ij} + \sum_{d' \in D'} (DC_{d'} + OC_{d'})\, z_{d'} \tag{5.1}$$

$$\text{s.t.} \sum_{j \in \delta^+(i)} y_{ij}^{k,l} + \sum_{j \in \delta_R^+(i)} x_{ij}^{k,l} = \sum_{j \in \delta^-(i)} y_{ji}^{k,l} + \sum_{j \in \delta_R^-(i)} x_{ji}^{k,l} \quad \forall i \in V, \quad \forall k \in K, \quad \forall l \in L \tag{5.2}$$

$$\sum_{\forall l \in L} \sum_{k \in K} x_{ij}^{k,l} = 1 \qquad \forall (i,j) \in A'_R$$

$$\sum_{\forall l \in L} \sum_{k \in K} (x_{ij}^{k,l} + x_{ji}^{k,l}) = 1 \quad \forall \{i,j\} \in E_R \tag{5.3}$$

$$y_{d'd}^{k,l} \le T_{d'}^{l} \qquad \forall d' \in D', \quad \forall k \in K, \quad \forall l \in L \tag{5.4}$$

$$\sum_{j \in \delta^-(i)} f_{ji}^{k,l} - \sum_{j \in \delta^+(i)} f_{ij}^{k,l} = \sum_{j \in \delta_R^-(i)} q_{ji} x_{ji}^{k,l} \quad \forall i \in V, \quad \forall k \in K, \quad \forall l \in L \tag{5.5}$$

$$\sum_{(i,j) \in A_R} q_{ij} x_{ij}^{k,l} - M(1 - T_d^l) \le f_{d'd}^{k,l} \le \sum_{(i,j) \in A_R} q_{ij} x_{ij}^{k,l} + M(1 - T_d^l) \quad \forall d' \in D', \quad \forall k \in K, \quad \forall l \in L \tag{5.6}$$

$$f_{d'd}^{k,l} \le M T_d^l \qquad \forall d' \in D', \quad \forall k \in K, \quad \forall l \in L \tag{5.7}$$

$$f_{dd'}^{k,l} = 0 \qquad \forall d' \in D', \quad \forall k \in K, \quad \forall l \in L \tag{5.8}$$

$$f_{ij}^{k,l} \le Q^k (x_{ij}^{k,l} + y_{ij}^{k,l}) \qquad \forall (i,j) \in A, \quad \forall k \in K, \quad \forall l \in L \tag{5.9}$$

$$DC_{d'} \ge \lambda \Big(\sum_{k \in K} y_{d'd}^{k,l} \Big) - M(1 - T_{d'}^{l}) \quad \forall d' \in D', \quad \forall l \in L \tag{5.10}$$

$$\sum_{d' \in D'} z_{d'} \le D_{max} \tag{5.11}$$

$$\sum_{\forall l \in L} T_{d'}^{l} = z_{d'} \qquad \forall d' \in D' \tag{5.12}$$

$$\sum_{d' \in D'} T^l_{d'} \leq 1 \qquad \forall l \in L \tag{5.13}$$

$$x^{k,l}_{ij} \leq \sum_{d' \in D'} T^l_{d'} \qquad \forall (i,j) \in A_R, \quad \forall k \in K, \quad \forall l \in L \tag{5.14}$$

$$y^{k,l}_{ij} \leq M \sum_{d' \in D'} T^l_{d'} \qquad \forall (i,j) \in A, \quad \forall k \in K, \quad \forall l \in L \tag{5.15}$$

$$\sum_{j \in D' \setminus \{i\}} r_{ij} = z_i \quad \forall i \in D' \setminus \{d_M\}, \qquad \sum_{i \in D' \setminus \{j\}} r_{ij} = z_j \quad \forall j \in D' \setminus \{d_M\} \tag{5.16}$$

$$\sum_{i \in S} \sum_{j \notin S} r_{ij} \geq z_h \qquad \forall S \subseteq D',\, d_M \in S,\, h \in D' \setminus S \tag{5.17}$$

$$\begin{aligned} & x^{k,l} \in \{0,1\}^{|A_R|}, \quad y^{k,l} \in \mathbb{Z}_+^{|A|} \quad \forall k \in K, \quad \forall l \in L \\ & f^{k,l}_{ij} \geq 0 \quad \forall (i,j) \in A, \quad \forall k \in K, \quad \forall l \in L \\ & z \in \{0,1\}^{|D'|}, \quad T^l \in \{0,1\}^{|D'|} \quad \forall l \in L, \\ & r_{ij} \in \{0,1\} \quad \forall i,j \in D'. \end{aligned} \tag{5.18}$$

The objective (5.1) seeks to minimize the sum over service and deadheading costs of the walking loops, the connection of parking lots by the transfer vehicle, and the costs for opening a depot and hiring a vehicle. Constraints (5.2)–(5.15) define feasible routes of the service vehicles $k \in K$, while constraints (5.16) and (5.17) ensure a feasible transfer tour. The connectivity of service tours at each node is ensured by equations (5.2) and the service of each required arc and edge by exactly one vehicle is guaranteed by equations (5.3). Inequalities (5.4) ensure a maximum of one traversal from the dummy parking lot to the original one if d is an opened parking lot. Constraints (5.5)–(5.8) are flow conservation constraints. Together with the coupling constraint (5.9) they ensure the elimination of infeasible subtours.

- Equations (5.5) ensure that the difference between inflow and outflow of node i is exactly the demand delivered on arcs entering node i. This type of constraint is also known as a generalized flow conservation constraint.
- Inequalities (5.6) ensure that if the parking lot d is opened, the flow leaving this node by vehicle k is equal to the demand delivered by the same vehicle linked to the parking lot by l.
- Inequalities (5.7) ensure that the outgoing flow from parking lot d is zero if d is not open.
- Equations (5.8) ensure that there is no flow left when finishing the tour by returning to the dummy parking lot.
- Inequalities (5.9) are upper bounds on the flow of an arc (i,j) and couple flow variables with traversing and servicing variables.

Next, inequalities (5.10) calculate the total hiring costs of tours assigned to parking lot d. Constraints (5.11)–(5.15) restrict the number of open parking lots, the number of labels assigned to a parking lot and the vehicles assigned to a parking lot by a label l. More precisely:

- Inequality (5.11) ensures at maximum D_{max} parking lots are open.
- Equalities (5.12) links the lth loop to a specific parking lot d.
- Inequalities (5.13) ensure that no more than one depot d is linked to a label l.
- Inequalities (5.14) and (5.15) ensure that a tour of vehicle k linked to parking lot d by l is constructed only if there is a link between d and l.

The last two sets of restrictions define a feasible tour for the transfer vehicle. Equalities (5.16) are generalized node degree constraints and state that exactly one arc has to leave (enter) node i if i is selected as a parking lot. If d_M is the only open parking lot, the transfer vehicle does not need to connect d_M to other parking lots. Therefore, no such constraint exists for the main-depot d_M. Constraints (5.17) eliminate subtours of the transfer vehicle including an open parking lot h disconnected from the main-depot d_M. This version of the generalized subtour elimination constraints stated here assumes that the main-depot d_M is always a possible parking lot.

Additional constraints, as presented in (Hashemi Doulabi and Seifi, 2013) and (Gouveia *et al.*, 2010), can be added to strengthen the formulation:

$$f_{ij}^{kl} \leq q_{ij} x_{ij}^{kl} \qquad \forall (i,j) \in A_R, \quad \forall k \in K, \quad \forall l \in L \tag{5.19}$$

$$f_{ij}^{kl} \leq \left(y_{ij}^{kl} - 1\right) \min_{(i^*j^*)} q_{i^*j^*} \qquad \forall (i,j) \in A \setminus A_R, \quad \forall k \in K, \quad \forall l \in L \tag{5.20}$$

$$\sum_{d \in D} dT_d^l \geq \sum_{d \in D} dT_d^{l+1} \qquad \forall l = 1, \ldots, D_{max} - 1. \tag{5.21}$$

Constraints (5.19) and (5.20) impose lower bounds on the flow variables and constraints (5.21) break symmetric solutions regarding the link to a parking lot.

Park and Loop Model with Flow Formulation

In order to formulate flow constraints for the transfer vehicle, new variables s_{ij} are necessary to model the flow for the transfer vehicle. The flow formulation (PAL-flow) of

the park and loop model then reads:

$$\text{(PAL-flow)} \quad \min \sum_{\forall l \in L} \left(\sum_{k \in K} \left(\sum_{(i,j) \in A_R} c_{ij}^{serv,k} x_{ij}^{k,l} + \sum_{(ij) \in A} c_{ij}^{k} y_{ij}^{k,l} \right) \right)$$
$$+ \sum_{i \in D'} \sum_{j \in D' \setminus \{i\}} SP_{ij} r_{ij} + \sum_{d' \in D'} \left(DC_{d'} + OC_{d'} \right) z_{d'} \tag{5.22}$$

$$\text{s.t. constraints } (5.2) - (5.15) \text{ of PAL} \tag{5.23}$$

$$\sum_{j \in D' \setminus \{i\}} r_{ij} = z_i \quad \forall i \in D' \setminus \{d_M\}, \qquad \sum_{i \in D' \setminus \{j\}} r_{ij} = z_j \quad \forall j \in D' \setminus \{d_M\} \tag{5.24}$$

$$\sum_{j \in D} s_{ji} - \sum_{j \in D} s_{ij} = z_i \quad \forall i \in D \setminus \{d_M\} \tag{5.25}$$

$$\sum_{j \in D} s_{d_M j} = |D| - 1 \tag{5.26}$$

$$(|D| - 1) r_{ij} - s_{ij} \geq 0 \quad \forall i, j \in D \text{ and } i \neq j \tag{5.27}$$

$$x^{k,l} \in \{0,1\}^{|A_R|}, \quad y^{k,l} \in \mathbb{Z}_+^{|A|} \quad \forall k \in K, \quad \forall l \in L$$

$$f_{ij}^{k,l} \geq 0 \quad \forall (i,j) \in A, \quad \forall k \in K, \quad \forall l \in L$$

$$z \in \{0,1\}^{|D'|}, \quad T^l \in \{0,1\}^{|D'|} \quad \forall l \in L$$

$$r_{ij} \in \{0,1\} \quad \forall i, j \in D', \quad s_{ij} \geq 0 \quad \forall (i,j) \in A. \tag{5.28}$$

The objective (5.22) and constraints (5.23) are the same as in the PAL model before. Constraints (5.24)–(5.27) model the tour of the transfer vehicle. Instead of adding generalized subtour elimination constraints (see (5.17)), a flow formulation is now used. Whenever node i is selected as an open parking lot, the transfer vehicle has to enter and leave that node, which is stated by constraints (5.24). Flow conservation is guaranteed by constraints (5.25) and (5.26). The first set of constraints (5.25) states that whenever node i is a parking lot, one unit of flow has to be absorbed by that node. The second constraint states that exactly $|D| - 1$ units of flow leave the main-depot d_M. If less than $|D|$ parking lots are open (either $D_{max} < |D|$ or because of optimality), the remaining flow is taken by the loop variable $s_{d_M d_M}$. Constraints (5.27) link the flow variables s_{ij} with the transfer variables r_{ij}. The same constraints (5.19)–(5.21) as for the PAL model can be added.

5.3.2 Combined Park and Loop and Curbline Models

This paragraph will introduce two mixed integer formulations for the combined park and loop and curbline delivery mode. In contrast to the models of the previous paragraph, the postman is now also allowed to deliver mail with the transfer vehicle. Again, the two models differ in the way they handle infeasible subtours of the transfer vehicle. The first formulation follows the idea of the sparse formulation presented by Belenguer

and Benavent (1998) for the CARP. Additional balance constraints resulting from the mixed network and constraints ensuring the connectivity of tour of the transfer vehicle are added. The computational results of Belenguer and Benavent (1998) show that the formulation work well for small-sized CARP instances. The second formulation uses flow variables in the same spirit as presented in (Gouveia *et al.*, 2010). Their computation results show that this formulation is sometimes able to improve the lower bounds obtained by the aggregated version of the spare formulation for the CARP.

Combined Park and Loop and Curbline Routes with Generalized SEC

In order to formulate the combined model, new variables u_e, u_a, w_e, w_a, and p_i are introduced. Variables u_e and u_a take value 1 if the transfer vehicle services edge e or arc a. Variables w_e and w_a count the number of times an edge e or arc a is traversed without being serviced by the transfer vehicle. Auxiliary variables p_i for every node $i \in V$ are needed to ensure even node degrees. Note that the transfer vehicle is indexed by $k = 0$ and the set of (required) arcs of the original network is indicated by A' (A'_R). A mathematical formulation for the combined delivery mode is then:

(CurbPAL)

$$\min \sum_{\forall l \in L} \left(\sum_{k \in K} \left(\sum_{(i,j) \in A_R} c_{ij}^{serv,k} x_{ij}^{k,l} + \sum_{(ij) \in A} c_{ij}^{k} y_{ij}^{k,l} \right) \right) + \sum_{e \in E_R} c_e^{serv,0} u_e + \sum_{a \in A'_R} c_a^{serv,0} u_a + \sum_{e \in E} c_e^0 w_e + \sum_{a \in A'} c_a^0 w_a + \sum_{d \in D} (DC_d + OC_d)\, z_d \tag{5.29}$$

$$\text{s.t. constraints (5.2) and (5.4) – (5.15) of PAL} \tag{5.30}$$

$$\sum_{\forall l \in L} \sum_{k \in K} x_{ij}^{k,l} + u_a = 1 \qquad \forall a = (i,j) \in A'_R,$$

$$\sum_{\forall l \in L} \sum_{k \in K} (x_{ij}^{k,l} + x_{ji}^{k,l}) + u_e = 1 \qquad \forall e = \{i,j\} \in E_R \tag{5.31}$$

$$\sum_{e \in \delta_R(d_M)} u_e + \sum_{a \in \delta_R^+(d_M)} u_a + \sum_{e \in \delta(d_M)} w_e + \sum_{a \in \delta^+(d_M)} w_a + z_{d_M} \geq 1 \tag{5.32}$$

$$\sum_{e \in E_R} u_e + \sum_{a \in A_R} u_a \leq Q^0 \tag{5.33}$$

$$\sum_{e \in \delta_R(i)} u_e + \sum_{a \in \delta_R^+(i) \cup \delta_R^-(i)} u_a + \sum_{e \in \delta(i)} w_e + \sum_{a \in \delta^+(i) \cup \delta^-(i)} w_a = 2p_i \quad \forall i \in V \tag{5.34}$$

$$- \sum_{a \in \delta_R^+(S)} u_a - \sum_{a \in \delta^+(S)} w_a + \sum_{a \in \delta_R^-(S)} u_a + \sum_{a \in \delta^-(S)} w_a$$
$$+ \sum_{e \in \delta_R(S)} u + \sum_{e \in \delta(S)} w_a \geq 0 \qquad \forall S \subseteq V \tag{5.35}$$

$$\sum_{e \in \delta_R(i)} u_e + \sum_{a \in \delta_R^+(i) \cup \delta_R^-(i)} u_a + \sum_{e \in \delta(i)} w_e + \sum_{a \in \delta^+(i) \cup \delta^-(i)} w_a \geq \begin{cases} 2u_e, & e \in E_R(S) \\ 2u_a, & a \in A'_R(S) \\ 2w_e, & e \in E(S) \\ 2w_a, & a \in A'(S) \\ 2z_i, & i \in D' \cap S \end{cases}$$
$$\forall S \subseteq V \setminus \{d_M\} \tag{5.36}$$

$$x^{k,l} \in \{0,1\}^{|A_R|}, \quad y^{k,l} \in \mathbb{Z}_+^{|A|}, \quad f^{k,l} \geq 0, \quad \forall k \in K, \forall l \in L,$$
$$u \in \{0,1\}^{|A'_R \cup E_R|}, \quad w \in \mathbb{Z}_+^{|A' \cup E|}, \quad p \in \mathbb{Z}_+^{|V|}$$
$$z \in \{0,1\}^{|D|}, \quad T^l \in \{0,1\}^{|D|} \quad \forall l \in L. \tag{5.37}$$

The objective (5.29) calculates the costs that occur when either a vehicle of the smaller delivery routes starting at a parking lot or the transfer vehicle services or traverses an edge or arc. Again, hiring costs for vehicles in use and fixed costs for opening a parking lot are taken into account. Constraints (5.30) are the same as for the pure park and loop delivery mode and model feasible routes for vehicles of the walking loops. The service constraints (5.31) take into account that it is now possible to service an edge or arc with the transfer vehicle. A feasible route for the transfer vehicle is defined by constraints (5.32)–(5.36). In detail:

- Constraint (5.32) ensures that if the main-depot d_M is not chosen as an open parking lot, at least one outgoing edge or arc of the transfer vehicle has to be used.
- Constraint (5.33) ensures the capacity restriction of the transfer vehicle.
- Constraints (5.34) ensure that for every node $i \in V$ the sum of incoming and outgoing edges and arcs is even.
- Constraints (5.35) are balanced set constraints and ensure that for every subset of nodes the difference between incoming and outgoing arcs can be compensated by edges.
- Subtour elimination constraints (5.36) ensure that there are no disconnected arcs or edges and no disconnected open parking lots from the main-depot d_M.

As before, constraints for bounding the flow of vehicles in the walking loop (5.19), (5.20) and symmetry breaking constraints (5.21) can be added to strengthen the formulation.

Combined Park and Loop and Curbline Routes with Flow Formulation

The second formulation of the combined park and loop and curbline routes uses again a flow formulation to describe a feasible route of the transfer vehicle. For each arc $(i,j) \in A'$, there is a variable w_{ij} counting the number of traversals of the transfer vehicle without servicing. Whenever the arc (i,j) is required, the binary variable u_{ij} exists taking value 1 if (i,j) is serviced by the transfer vehicle, and 0 otherwise. Two service and deadheading variables u_{ij}, u_{ji} and w_{ij}, w_{ji}, respectively, exist for each edge $\{i,j\}$ representing both directions in which the edge can be traversed. Similar, flow variables s_{ij} exist, one for each arc (i,j) and two for each edge $\{i,j\}$. The flow formulation of the combined PAL with curbline problem then reads:

(CurbPAL-flow)

$$\min \sum_{\forall l \in L} \left(\sum_{k \in K} \left(\sum_{(i,j) \in A_R} c_{ij}^{serv,k} x_{ij}^{k,l} + \sum_{(ij) \in A} c_{ij}^{k} y_{ij}^{k,l} \right) \right)$$
$$+ \sum_{(i,j) \in A_R} c_{ij}^{serv,0} u_{ij} + \sum_{(i,j) \in A} c_{ij}^{0} w_{ij} + \sum_{d \in D} (DC_d + OC_d)\, z_d \tag{5.38}$$

s.t. constraints (5.2) and (5.4) – (5.15) of PAL (5.39)

$$\sum_{\forall l \in L} \sum_{k \in K} x_{ij}^{k,l} + u_{ij} = 1 \qquad \forall (i,j) \in A'_R,$$

$$\sum_{\forall l \in L} \sum_{k \in K} (x_{ij}^{k,l} + x_{ji}^{k,l}) + (u_{ij} + u_{ji}) = 1 \qquad \forall \{i,j\} \in E_R \tag{5.40}$$

$$\sum_{j \in \delta^+(i)} w_{ij} + \sum_{\delta_R^+(i)} u_{ij} = \sum_{\delta^-(i)} w_{ji} + \sum_{\delta_R^-(i)} u_{ji} \qquad \forall i \in V \tag{5.41}$$

$$\sum_{j \in \delta^-(i)} s_{ji} - \sum_{j \in \delta^+(i)} s_{ij} = \sum_{j \in \delta_R^-(i)} q_{ji} u_{ji} \qquad \forall i \in V \tag{5.42}$$

$$\sum_{j \in \delta^+(d_M)} s_{d_M j} = \sum_{(i,j) \in A_R} q_{ij} u_{ij} \tag{5.43}$$

$$s_{ij} \le Q^0 (u_{ij} + w_{ij}) \qquad \forall (i,j) \in A \tag{5.44}$$

$$\sum_{j \in \delta^+(d')} u_{d'j} + \sum_{j \in \delta^+(d')} w_{d'j} \ge z_{d'} \qquad \forall d' \in D' \tag{5.45}$$

$$x^{k,l} \in \{0,1\}^{|A_R|}, \quad y^{k,l} \in \mathbb{Z}_+^{|A|}, \quad f^{k,l} \ge 0, \quad \forall k \in K,\ \forall l \in L,$$
$$u \in \{0,1\}^{|A'_R \cup E_R|}, \quad w \in \mathbb{Z}_+^{|A' \cup E|}, \quad s \ge 0$$
$$z \in \{0,1\}^{|D|}, \quad T^l \in \{0,1\}^{|D|} \quad \forall l \in L. \tag{5.46}$$

The objective (5.38) and constraints (5.39) are the same as in the PAL model of the previous section. The service constraints (5.40) take into account that it is possible to service an edge or arc with the transfer vehicle. Because of the flow formulation, there are two variables associated with every edge, one for each possible traversing direction.

A feasible route for the transfer vehicle is defined by constraints (5.41)–(5.45), which are formulated with flow variables. In detail:

- Constraints (5.41) ensure the continuity of the transfer vehicle's tour.
- Constraints (5.42) ensure that the difference between incoming and outgoing flow of the transfer vehicle at node i is exactly the demand absorbed by the demand of arcs entering node i and serviced with the transfer vehicle. These generalized flow conservation constraints are similar to constraints (5.5) of the PAL model.
- Constraint (5.43) ensures that the outgoing flow of the main-depot d_M equals the demand delivery by the transfer vehicle.
- Constraints (5.44) are upper bounds on the flow variables.
- Constraints (5.45) ensure that the transfer vehicle reaches every open parking lot.

As before, constraints for bounding the flow of vehicles in the walking loop (5.19), (5.20) and symmetry breaking constraints (5.21) can be added to strengthen the formulation.

5.4 Computational Results

This section reports computational results for the four models presented for the park and loop problem and the combination with curbline delivery. To test the quality of the new formulations, we randomly generated a benchmark set of 39 instances on mixed graphs with both required and non-required edges and arcs. For instances `Pal1` to `Pal25`, the size of the underlying network increases for both nodes and links. There are three types of vehicles available, where the first type always represents the transfer vehicle. It is assumed that there exists just one transfer vehicle. Instances `Pal26` to `Pal39` model problems with two or three districts that are connected by arcs and edges. These instances are derived by combining some of the first problems and additional arcs and edges to ensure a strongly connected graph. Again, three different vehicle types are given, but more vehicles per type are available. Service and transfer costs are provided for every link and every vehicle type. About 20% of the nodes represent possible parking lots.

All computations were performed on a standard PC with an Intel©Core$^{\text{TM}}$ i7-2600 processor at 3.4 GHz with 16 GB of main memory. The four models for the (combined) park and loop problem were introduced to CPLEX through the callable C++ API of CPLEX 12.2 and the cutting plane algorithm was coded in C++ (MS Visual Studio, 2010). For separating violated SEC in the (Curb)PAL model, we follow the approach of Benavent *et al.* (2000) and compute a minimum cut separating the main-depot d_M and a parking lot h or an edge or arc traversed or serviced by the transfer vehicle. Separating violated balanced set constraints is a bit more intricate to implement. The separation procedure we follow is described in (Nobert and Picard, 1996). A hard time limit of four hours has been set for CPLEX to solve the model. We also use CPLEX for finding an upper bound and applied the feasibility pump heuristic with an emphasis on finding a feasible solution.

Tables 5.1–5.4 report results for the linear relaxation and for the branch-and-cut on all tested benchmark sets. The entries of the header of all tables have the following meaning:

$D_{max} = 2,3$	maximum number of open parking lots
(Curbline+) PAL/PAL-flow	lower bounds and computation times for the PAL model (5.1)–(5.17), the PAL-flow model (5.22)–(5.27) the combined park and loop with curbline mode CurbPAL model (5.29)–(5.36) and the CurbPAL-flow model (5.38)–(5.44)
instance	name of the instance
$\|V\|,\|A \cup E\|,\|A_R\|,\|E_R\|,P_{total}$	characteristics of the instance: $\|V\|$ number of nodes, $\|A \cup E\|$ number of links, $\|A_R\|$ number of required arcs, $\|E_R\|$ number of required edges, P_{total} total number of vehicles at each parking lot For the sake of brevity, this information is omitted in the reporting integer results.
lb	lower bound computed by linear relaxation and at termination of branch-and-cut
time	computation time in seconds; if the time limit is reached, it is indicated by *4h*
lb_{best}	best lower bound obtained with either the PAL or PAL-flow formulation or CurbPAL or CurbPAL-flow formulation for the combined problem
ub	best upper bound reported by CPLEX

The following additional information is given for the respective model:

Num lb_{best}	number of instances when the model provided the best lower bound lb_{best}
Num opt	number of obtained integer solutions
avg time	average computation time for linear relaxation and branch-and-cut

5.4.1 Linear Relaxation Results

We will start with the analysis of the linear relaxation results for both delivery modes obtained at the end of the root node. Tables 5.1 and 5.2 show the results for the 39 `Pal` instances.

Comparing the values of the lower bounds for the pure park and loop delivery mode,

Table 5.1: Lower bounds for `Pal` instances with park and loop mode

						$D_{max} = 2$					$D_{max} = 3$				
						PAL		PAL-flow			PAL		PAL-flow		
instance	$\|N\|$	$\|A \cup E\|$	$\|A_R\|$	$\|E_R\|$	P_{total}	lb	time	lb	time	lb_{best}	lb	time	lb	time	lb_{best}
Pal1	9	14	6	6	6	403	0.0	403	0.1	403	403	0.1	403	0.2	403
Pal2	9	17	9	5	6	470	0.3	453	0.2	470	470	0.4	468	0.3	470
Pal3	9	13	5	7	6	317	1.9	316	0.2	317	328	3.8	326	0.2	328
Pal4	10	15	7	9	6	279	0.1	288	0.2	288	279	0.6	279	0.2	279
Pal5	9	17	8	8	8	431	1.3	434	0.4	434	431	3.9	423	0.6	431
Pal6	36	53	38	33	6	1711	7.0	1720	7.1	1720	1714	15.9	1708	14.1	1714
Pal7	36	46	32	32	6	1913	8.5	1904	5.6	1913	1903	13.8	1905	8.4	1905
Pal8	36	46	32	34	6	1763	16.5	1764	9.0	1764	1754	72.2	1747	47.1	1754
Pal9	36	48	34	32	6	1573	7.6	1568	7.0	1573	1555	21.2	1561	17.4	1561
Pal10	36	60	45	30	6	1839	7.9	1858	7.1	1858	1836	13.2	1835	7.5	1836
Pal11	37	66	20	18	6	1236	6.5	1232	4.1	1236	1226	16.7	1214	7.4	1226
Pal12	36	66	21	18	6	1138	2.7	1151	2.6	1151	1136	9.0	1137	6.8	1137
Pal13	36	63	19	17	6	1146	7.0	1148	4.4	1148	1138	14.4	1138	9.3	1138
Pal14	37	69	21	21	6	1082	6.8	1086	5.2	1086	1088	21.3	1074	9.4	1088
Pal15	36	68	21	18	8	1311	8.3	1316	4.6	1316	1301	12.9	1301	7.4	1301
Pal16	81	111	85	81	6	4314	238.2	4319	85.2	4319	4268	371.4	4254	110.6	4268
Pal17	81	102	77	82	6	4320	184.3	4317	105.6	4320	4284	319.5	4260	77.8	4284
Pal18	81	105	81	77	6	4655	112.4	4614	39.5	4655	4622	595.9	4621	179.6	4622
Pal19	81	112	87	76	6	3963	108.4	3939	50.5	3963	3930	126.9	3924	85.7	3930
Pal20	81	105	81	77	6	4156	345.9	4124	61.0	4156	4114	390.3	4107	55.2	4114
Pal21	81	140	44	44	6	4958	54.5	4949	21.9	4958	4946	315.3	4946	87.1	4946
Pal22	81	143	46	43	6	2823	105.7	2822	68.4	2823	2817	474.8	2816	237.5	2817
Pal23	81	140	43	46	6	2783	133.9	2787	77.2	2787	2757	470.3	2759	304.8	2759
Pal24	81	142	46	44	6	3195	185.0	3203	56.8	3203	3169	170.6	3181	187.8	3181
Pal25	81	136	40	48	6	2489	55.7	2515	72.0	2515	2474	347.1	2472	171.6	2474
Pal26	18	27	15	11	7	830	4.1	834	2.9	834	830	4.9	829	3.7	830
Pal27	18	24	12	13	6	777	3.9	777	0.6	777	766	4.5	768	1.5	768
Pal28	18	26	14	13	5	803	4.1	812	2.5	812	796	4.8	780	2.7	796
Pal29	18	30	16	14	6	679	4.6	679	2.9	679	667	4.3	668	3.1	668
Pal30	18	28	16	15	8	763	4.8	766	2.7	766	760	5.9	762	4.4	762
Pal31	27	47	25	19	12	1187	12.0	1185	7.1	1187	1185	25.9	1185	13.5	1185
Pal32	27	40	23	19	11	1251	9.0	1255	6.1	1255	1238	26.8	1237	8.1	1238
Pal33	54	74	50	49	13	2271	43.2	2272	196.4	2272	2262	81.3	2262	283.2	2262
Pal34	27	43	23	23	11	951	8.4	946	3.6	951	947	10.0	946	6.1	947
Pal35	54	78	53	50	13	2500	101.6	2472	53.7	2500	2462	143.4	2466	112.0	2466
Pal36	54	72	48	48	16	2377	752.4	2381	934.7	2381	2365	1458.4	2360	1314.7	2365
Pal37	54	71	45	48	16	2364	167.4	2353	101.4	2364	2350	686.3	2376	1465.2	2376
Pal38	27	44	24	23	8	1159	14.2	1161	10.2	1161	1157	32.4	1158	13.7	1158
Pal39	54	88	61	45	6	2538	13.3	2534	6.3	2538	2534	59.1	2527	9.0	2534
Num lb_{best}						19		23			28		18		
avg time							70.5		52.0			162.8		125.0	

Table 5.2: Lower bounds for `Pal` instances with combined park and loop with curbline mode

						$Dmax = 2$					$Dmax = 3$				
						CurbPAL		CurbPAL-flow			CurbPAL		CurbPAL-flow		
instance	$\|N\|$	$\|A \cup E\|$	$\|A_R\|$	$\|E_R\|$	P_{total}	lb	time	lb	time	lb_{best}	lb	time	lb	time	lb_{best}
Pal1	9	14	6	6	6	395	0.5	395	0.2	395	395	1.6	395	0.1	395
Pal2	9	17	9	5	6	433	0.7	454	0.3	454	427	0.8	453	0.3	453
Pal3	9	13	5	7	6	307	0.4	317	0.2	317	312	2.0	317	0.2	317
Pal4	10	15	7	9	6	286	1.6	280	0.8	286	274	0.7	273	0.5	274
Pal5	9	17	8	8	8	408	0.4	418	0.4	418	405	4.6	408	0.5	408
Pal6	36	53	38	33	6	1663	68.7	1665	9.4	1665	1659	193.4	1671	9.0	1671
Pal7	36	46	32	32	6	1856	44.2	1867	5.9	1867	1855	114.8	1850	9.2	1855
Pal8	36	46	32	34	6	1689	56.9	1693	7.0	1693	1673	145.5	1687	8.8	1687
Pal9	36	48	34	32	6	1529	27.2	1550	7.4	1550	1530	86.3	1544	6.4	1544
Pal10	36	60	45	30	6	1792	44.8	1801	6.4	1801	1786	116.3	1815	10.8	1815
Pal11	37	66	20	18	6	1192	27.2	1191	5.3	1192	1170	34.5	1178	6.1	1178
Pal12	36	66	21	18	6	1111	14.9	1131	5.9	1131	1107	31.4	1113	7.1	1113
Pal13	36	63	19	17	6	1126	30.1	1135	7.8	1135	1123	53.7	1129	14.3	1129
Pal14	37	69	21	21	6	1046	36.1	1039	5.7	1046	1042	70.5	1036	6.2	1042
Pal15	36	68	21	18	8	1275	24.6	1274	4.7	1275	1275	40.7	1284	8.8	1284
Pal16	81	111	85	81	6	4124	3116.0	4135	35.4	4135	4101	6632.5	0	0.0	4101
Pal17	81	102	77	82	6	4180	2241.4	4217	181.5	4217	4154	6922.4	4198	82.9	4198
Pal18	81	105	81	77	6	4530	2241.0	4541	154.8	4541	4511	4540.4	4535	423.4	4535
Pal19	81	112	87	76	6	3849	2419.7	3864	47.1	3864	3829	4922.7	3837	91.4	3837
Pal20	81	105	81	77	6	4025	2421.3	4061	152.1	4061	4008	4660.3	4027	180.3	4027
Pal21	81	140	44	44	6	4862	872.8	4888	13.5	4888	4858	2253.5	4858	38.7	4858
Pal22	81	143	46	43	6	2693	1055.1	2724	51.4	2724	2687	2902.2	2700	94.4	2700
Pal23	81	140	43	46	6	2677	1758.7	2698	246.6	2698	2647	3469.8	2655	321.6	2655
Pal24	81	142	46	44	6	3077	909.3	3098	0.0	3077	3069	2036.3	3072	134.5	3072
Pal25	81	136	40	48	6	2426	1071.1	2439	66.6	2439	2402	2667.3	2426	231.0	2426
Pal26	18	27	15	11	7	555	4.5	555	2.5	555	555	5.3	555	2.6	555
Pal27	18	24	12	13	6	569	4.4	569	1.0	569	568	5.4	569	1.6	569
Pal28	18	26	14	13	5	558	5.2	552	2.6	558	558	7.2	555	2.7	558
Pal29	18	30	16	14	6	548	5.0	550	2.5	550	545	6.6	545	3.0	545
Pal30	18	28	16	15	8	616	6.6	620	3.2	620	616	9.5	620	3.7	620
Pal31	27	47	25	19	12	888	12.2	890	4.0	890	889	29.5	890	5.7	890
Pal32	27	40	23	19	11	906	19.2	915	1.9	915	898	28.4	901	5.3	901
Pal33	54	74	50	49	13	1895	941.1	1892	17.1	1895	1880	1864.7	1888	28.8	1888
Pal34	27	43	23	23	11	811	12.9	816	3.9	816	811	23.2	810	5.5	811
Pal35	54	78	53	50	13	2026	978.0	2031	34.5	2031	2026	2366.4	2026	34.0	2026
Pal36	54	72	48	48	16	1929	998.4	1940	94.6	1940	1932	2672.5	1935	81.0	1935
Pal37	54	71	45	48	16	1850	1430.6	1858	19.4	1858	1851	1802.8	1851	101.9	1851
Pal38	27	44	24	23	8	938	19.3	937	4.1	938	937	36.3	937	5.0	937
Pal39	54	88	61	45	6	2109	169.6	2112	8.9	2112	2109	435.3	2108	11.4	2109
Num lb_{best}						10		32			13		33		
avg time							592.1		33.0			1312.7		51.9	

the performance of a formulation seems to depend on the problem size. If only two parking lots can be opened, the number of best lower bounds lb_{best} is slightly higher for the PAL-flow model than for the PAL model. However, allowing only three open parking lots, the MIP size increases by $\mathcal{O}(|A|P_{total})$ both in variables and constraints. Then, the PAL model formulated with generalized SEC results more often in better lower bounds than the flow formulation PAL-flow. Overall, the computation time for solving the root node is on average higher for the PAL model than for the PAL-flow model.

Comparing the linear relaxation results for the combined park and loop with curbline mode in Table 5.2, we see that the PAL-flow model clearly outperforms the PAL model. For both $D_{max} = 2$ and $D_{max} = 3$, higher lower bounds are obtained with the PAL-flow formulation in 31 and 32 out of 39 cases, respectively. Also, the average computation time is by factor 18 and 25, respectively, drastically smaller than the average computation time of the root node with the PAL model.

The combined park and loop with curbline mode is a more complex problem than the pure park and loop mode, as additional decisions on whether or not to service an edge or arc with the transfer vehicle are required. Increasing computation times for the root node of the combined problem modeled with the CurbPAL formulation support this. Surprisingly, solving the root node of the CurbPAL-flow formulation is much faster than the PAL-flow formulation of the pure park and loop mode.

5.4.2 Integer Results

In Tables 5.3 and 5.4, we report results at the end of the branch-and-bound tree. Again, the flow formulation of the PAL problem clearly outperforms the formulation with the generalized SEC. As reported in Table 5.3, the PAL-flow model results in 38 out of 39 cases in a better lower bound. The only exceptions are instances `Pal35` for $D_{max} = 2$ and `Pal36` for $D_{max} = 3$. Also, the number of obtained optimal integer solutions is much higher for the flow formulation (31 and 26, respectively). The advantage of the PAL-flow model is also shown by the smaller average computation times (about factor two times faster). Similar results are obtained for the combined park and loop and curbline mode. There, all 39 instances for both $D_{max} = 2$ and $D_{max} = 3$ are solved best with the flow formulation. All integer solutions found with the CurbPAL-flow model are also found with the CurbPAL model, but not vice versa.

5.5 Conclusion

Extending the standard MCARP by a location aspect results in a combined location and arc routing problem (LARP). In this paper, we have considered the different mail delivery modes classified by Bodin and Levy (2000) and have assigned the existing literature on LARPs to these categories. We also presented mathematical formulations for two of the delivery modes, and showed computational results for all formulations. There are several possibilities for how locations are connected among themselves and with delivery routes.

Table 5.3: Integer results for `Pal` instances with park and loop mode

	$D_{max} = 2$							$D_{max} = 3$						
	PAL			PAL-flow				PAL			PAL-flow			
instance	lb	ub	time	lb	ub	time	lb_{best}	lb	ub	time	lb	ub	time	lb_{best}
Pal1	opt		0.1	opt		0.1	403	opt		0.3	opt		0.1	403
Pal2	opt		0.5	opt		0.2	472	opt		0.8	opt		0.7	472
Pal3	opt		2.7	opt		0.1	328	opt		3.7	opt		0.1	328
Pal4	opt		3.7	opt		0.5	303	opt		24.8	opt		0.7	303
Pal5	opt		6.9	opt		1.3	444	opt		14.1	opt		1.3	444
Pal6	1734	1740	*4h*	opt		35.8	1740	1725	1740	*4h*	opt		149.4	1740
Pal7	opt		1753.3	opt		28.6	1944	opt		8109.3	opt		75.4	1944
Pal8	opt		102.6	opt		18.3	1770	opt		924.0	opt		80.5	1770
Pal9	opt		99.4	opt		14.6	1586	opt		6791.4	opt		31.0	1586
Pal10	1867	1883	*4h*	opt		46.3	1883	1849	1883	*4h*	opt		316.4	1883
Pal11	opt		880.7	opt		22.8	1258	opt		9273.1	opt		57.8	1258
Pal12	opt		1908.8	opt		35.1	1186	opt		11346.5	opt		103.9	1186
Pal13	opt		237.6	opt		43.9	1186	opt		7415.7	opt		292.8	1186
Pal14	opt		10341.2	opt		90.7	1132	1105	1136	*4h*	opt		368.5	1132
Pal15	opt		131.6	opt		21.0	1328	opt		620.9	opt		71.9	1328
Pal16	4328	4345	*4h*	opt		5311.6	4344	4268	4668	*4h*	4316	4511	*4h*	4316
Pal17	4334	4420	*4h*	opt		4771.6	4361	4292	4549	*4h*	4343	4371	*4h*	4343
Pal18	4660	4806	*4h*	4700	4726	*4h*	4700	4623	5224	*4h*	4668	4879	*4h*	4668
Pal19	3972	4004	*4h*	opt		6944.7	3999	3934	4269	*4h*	3987	3999	*4h*	3987
Pal20	4162	4333	*4h*	4179	4297	*4h*	4179	4120	4426	*4h*	4163	4318	*4h*	4163
Pal21	5005	5006	*4h*	opt		302.7	5006	4958	5017	*4h*	opt		1838.6	5006
Pal22	2828	3074	*4h*	opt		7515.2	2870	2820	2892	*4h*	2838	2924	*4h*	2838
Pal23	2802	2833	*4h*	opt		4879.8	2821	2769	2822	*4h*	opt		11538.7	2821
Pal24	3199	3291	*4h*	3262	3279	*4h*	3262	3171	3687	*4h*	3215	3291	*4h*	3215
Pal25	2503	2617	*4h*	opt		379.3	2559	2489	2683	*4h*	opt		8757.8	2559
Pal26	opt		351.0	opt		16.5	899	opt		3769.8	opt		24.2	899
Pal27	opt		8.8	opt		2.1	805	opt		179.2	opt		4.8	805
Pal28	opt		283.9	opt		12.5	848	841	848	*4h*	opt		114.6	848
Pal29	opt		10.1	opt		6.3	682	opt		51.6	opt		2.8	682
Pal30	opt		860.5	opt		33.6	776	775	776	*4h*	opt		40.5	776
Pal31	1203	1307	*4h*	1253	1266	*4h*	1253	1188	1295	*4h*	1224	1347	*4h*	1224
Pal32	1274	1299	*4h*	opt		2305.8	1299	1243	1300	*4h*	opt		6951.5	1299
Pal33	2271		*4h*	2293	2377	*4h*	2293	2262	2659	*4h*	2280		*4h*	2280
Pal34	984	1009	*4h*	opt		1762.5	1009	972	1021	*4h*	opt		3060.7	1009
Pal35	2500		*4h*	2483	2684	*4h*	2500	2468		*4h*	2472	2639	*4h*	2472
Pal36	2378		*4h*	2394		*4h*	2394	2370		*4h*	2366		*4h*	2370
Pal37	2365		*4h*	2369	3182	*4h*	2369	2351	3356	*4h*	2362		*4h*	2362
Pal38	1175	1184	*4h*	opt		155.0	1184	1174	1184	*4h*	opt		1326.5	1184
Pal39	2549	2566	*4h*	opt		6964.7	2564	2541	2570	*4h*	2557	2564	*4h*	2557
Num lb_{best}	19			38				16			38			
Num opt	18			31				15			26			
avg time			8189.3			4023.7				10105.8			5702.9	

Table 5.4: Integer results for `Pal` instances with combined park and loop with curbline mode

	$D_{max} = 2$							$D_{max} = 3$						
	CurbPAL			CurbPAL-Flow				CurbPAL			CurbPAL-Flow			
instance	lb	ub	time	lb	ub	time	lb_{best}	lb	ub	time	lb	ub	time	lb_{best}
Pal1	opt		0,4	opt		0,5	395	opt		1,2	opt		0,1	395
Pal2	opt		10.9	opt		8.1	454	opt		14.6	opt		0.6	454
Pal3	opt		1.8	opt		0.3	317	opt		2.0	opt		0.1	317
Pal4	opt		10.3	opt		11.6	298	opt		88.8	opt		6.8	298
Pal5	opt		126.7	opt		4.8	431	opt		167.9	opt		189.7	431
Pal6	1663	1735	*4h*	1689	1690	*4h*	1689	1654	1910	*4h*	1676	1690	*4h*	1676
Pal7	opt		796.2	opt		40.2	1874	1854	1878	*4h*	opt		136.8	1874
Pal8	1707	1735	*4h*	opt		7057.4	1727	1677	2033	*4h*	opt		2648.1	1727
Pal9	1539	1563	*4h*	opt		113.3	1561	1528	1561	*4h*	opt		3169.9	1561
Pal10	1798	1874	*4h*	1824	1830	*4h*	1824	1785		*4h*	opt		1528.2	1830
Pal11	1204	1213	*4h*	opt		890.0	1213	1191	1216	*4h*	opt		3473.5	1213
Pal12	1138	1149	*4h*	opt		87.7	1149	1126	1152	*4h*	opt		311.2	1149
Pal13	opt		10042.7	opt		6507.3	1179	1147	1179	*4h*	1167	1179	*4h*	1167
Pal14	1063	1114	*4h*	1075	1112	*4h*	1075	1047	1141	*4h*	1064	1109	*4h*	1064
Pal15	1285	1302	*4h*	opt		92.0	1302	1278	1304	*4h*	opt		33.0	1302
Pal16	4122		*4h*	4166	4332	*4h*	4166	4106		*4h*	4144		*4h*	4144
Pal17	4172		*4h*	4235	4445	*4h*	4235	4157		*4h*	4207		*4h*	4207
Pal18	4532		*4h*	4577	4833	*4h*	4577	4514		*4h*	4553	4764	*4h*	4553
Pal19	3836		*4h*	3887	3950	*4h*	3887	3835		*4h*	3856	4138	*4h*	3856
Pal20	4036		*4h*	4073	4243	*4h*	4073	4010		*4h*	4040		*4h*	4040
Pal21	4881	4903	*4h*	opt		220.0	4902	4869	4936	*4h*	4894	4902	*4h*	4894
Pal22	2694		*4h*	2730	2793	*4h*	2730	2688		*4h*	2718		*4h*	2718
Pal23	2682		*4h*	2715	2924	*4h*	2715	2659		*4h*	2687	3008	*4h*	2687
Pal24	3088	3796	*4h*	3120	3229	*4h*	3120	3069		*4h*	3098	3420	*4h*	3098
Pal25	2436		*4h*	2464	2593	*4h*	2464	2399		*4h*	2448	2616	*4h*	2448
Pal26	opt		4.9	opt		2.9	555	opt		4.9	opt		2.8	555
Pal27	opt		5.8	opt		2.1	569	opt		10.8	opt		1.7	569
Pal28	opt		15.7	opt		9.4	561	opt		39.6	opt		7.1	560
Pal29	opt		4.2	opt		2.9	550	opt		70.3	opt		7.3	550
Pal30	opt		29.3	opt		4.2	621	opt		106.8	opt		5.0	621
Pal31	opt		96.8	opt		12.9	900	893	900	*4h*	opt		30.6	900
Pal32	opt		205.5	opt		3.8	915	911	915	*4h*	opt		30.8	915
Pal33	1884		*4h*	1901	1942	*4h*	1901	1879		*4h*	1894	2107	*4h*	1894
Pal34	825	842	*4h*	opt		4977.1	839	814	838	*4h*	826	832	*4h*	826
Pal35	2026	2292	*4h*	2039	2164	*4h*	2039	2026		*4h*	2033	2070	*4h*	2033
Pal36	1939		*4h*	1956	1971	*4h*	1956	1931		*4h*	1947		*4h*	1947
Pal37	1851		*4h*	1871	1930	*4h*	1871	1845	2102	*4h*	1862		*4h*	1862
Pal38	opt		620.7	opt		95.2	945	939	945	*4h*	opt		113.6	945
Pal39	2111		*4h*	opt		5799.1	2125	2104		*4h*	2119	2141	*4h*	2119
Num lb_{best}	15			39				10			39			
Num opt	15			23				10			20			
avg time			9168.5			6572.9				10720.7			7315.3	

First, the existing literature concerned with location arc routing was reviewed to distinguish between these possibilities. Considering the mail delivery context, these extensions were described by the delivery modes Curbline and Dismount Route, Relay Box Route, Park and Loop, Combined Park and Loop with Curbline Mode and Combined Relay Box with Curbline Mode.

Second, when the postman drives a transfer vehicle that does not allow a service to be performed, the corresponding delivery mode is called park and loop mode otherwise it is called combined park and loop with curbline mode. For each of these two delivery modes, we have proposed two mathematical formulations. The two formulations differ in how they model feasible routes of the transfer vehicle. In particular, the first model uses generalized subtour elimination constraints. This has the advantage of starting with a small problem and adding only relevant constraints. The second model uses a flow formulation to model feasible routes of the transfer vehicle. The advantage here is that only a linear number of constraints is needed to describe feasible routes. Therefore, the whole model can be stated at once. As far as we know, this is the first time that a mathematical formulation both for the park and loop and the combined park and loop with curbline delivery has been presented.

Third, we have provided computational results, where the performance of the different formulations is analyzed. The results show that the flow formulations of both problem types outperform the formulations with generalized subtour elimination constraints. Both better bounds and lower computation times are obtained with the flow formulations.

5.6 Bibliography

Amaya, C.-A., Langevin, A., and Trépanier, M. (2007). The capacitated arc routing problem with refill points. *Operations Research Letters*, **35**(1), 45–53.

Amaya, C.-A., Langevin, A., and Trépanier, M. (2010). A heuristic method for the capacitated arc routing problem with refill points and multiple loads. *Journal of the Operational Research Society*, **61**(7), 1095–1103.

Amberg, A., Domschke, W., and Voss, S. (2000). Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees. *European Journal of Operational Research*, **124**, 360–376.

Assad, A. A. and Golden, B. L. (1995). Chapter 5 arc routing methods and applications. In C. M. M.O. Ball, T.L. Magnanti and G. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, pages 375 – 483. Elsevier.

Balas, E. (1989). The prize collecting traveling salesman problem. *Networks*, **19**(6), 621–636.

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

Belenguer, J. M., Benavent, E., Lacomme, P., and Prins, C. (2006). Lower and upper bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, **33**(12), 3363–3383.

Belenguer, J. M., Benavent, E., and Irnich, S. (2013). The capacitated arc routing problem: Exact algorithms. In Corberán and Laporte (2013), chapter 9. (In preparation.).

Benavent, E., Ángel Corberán, and Sanchis, J. (2000). Linear programming based methods for solving arc routing problems. In Dror (2000), chapter 7, pages 231–275.

Bodin, L. and Levy, L. (2000). Scheduling of local delivery carrier routes for the United States Postal Service. In Dror (2000), chapter 11, pages 419–442.

Corberán, Á. and Laporte, G., editors (2013). *Arc Routing: Problems, Methods and Applications*. MOS-SIAM Series on Optimization. SIAM, Philadelphia. (In preparation.).

Del Pia, A. and Filippi, C. (2006). A variable neighborhood descent algorithm for a real waste collection problem with mobile depots. *International Transactions in Operational Research*, **13**(2), 125–141.

Fischetti, M., Salazar-González, J.-J., and Toth, P. (2004). The generalized traveling salesman and orienteering problems. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, Combinatorial Optimization, chapter 12, pages 609–662. Springer US.

Gavish, B. and Graves, S. C. (1978). The travelling salesman problem and related problems. Technical report, Massachusetts Institute of Technology, Operations Research Center.

Ghiani, G., Improta, G., and Laporte, G. (2001). The capacitated arc routing problem with intermediate facilities. *Networks*, **37**(3), 134–143.

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

Gouveia, L., Mourão, M. C., and Pinto, L. S. (2010). Lower bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, **37**, 692–699.

Hashemi Doulabi, S. H. and Seifi, A. (2013). Lower and upper bounds for location-arc routing problems with vehicle capacity constraints. *European Journal of Operational Research*, **224**(1), 189–208.

Levy, L. and Bodin, L. (1989). The arc oriented location routing problem. *Information Systems and Operations Research*, **27**(1), 74–94.

Nobert, Y. and Picard, J. (1996). An optimal algorithm for the mixed chinese postman problem. *Networks*, **27**, 95–108.

Prins, C. (2013). The capacitated arc routing problem: Heuristics. In Corberán and Laporte (2013), chapter 7. (In preparation.).

Roy, S. and Rousseau, J. (1989). The Capacitated Canadian Postman Problem. *Information Systems and Operations Research*, **27**, 58–73.

Salazar-Aguilar, M. A., Langevin, A., and Laporte, G. (2013). The synchronized arc and node routing problem: application to road marking. *Computers & Operations Research*.

Salhi, S. and Rand, G. (1989). The effect of ignoring routes when location depots. *European Journal of Operational Research*, **39**, 150–156.

Chapter 6

Conclusion

The goal of the thesis was to develop a full exact method for the CARP that, for the first time, exploits the original sparse network. Second, a Park and Loop delivery problem was formulated which is a more complex and realistic problem than the basic CARP .

Development of a branch-and-price algorithm. Solution methods for the CARP presented in the literature so far do either transform the problem into a node routing problem, which results in dense graphs or do not guarantee integer solutions. The proposed branch-and-price algorithm in Chapter 2 exploits the sparsity of the underlying network which enables efficient pricing and integer solutions are obtained due to an effective branching scheme, which does not change the structure of the pricing network. Advantageous for the pricing computation is also the fact that there are no negative reduced costs on deadheading edges by adding dual-optimal inequalities. Therefore, Dijkstra's algorithm can be used. An aggregation over vehicles in the master problem eliminates symmetry, but still allows the reconstruction of individual vehicle routes. Due to the branching on followers, where the consecutively service of two required edges is enforced or forbidden, integer routes are obtained. In the CARP case, those edges do not need to be connected directly, but a deadheading path might be in between. 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.

Labeling algorithms for the pricing relaxations. Using pricing problem relaxations is a standard technique in column generation because pricing problems in routing applications are typically strongly $\mathcal{NP}$-hard elementary shortest-path problems with resource constraints. However, replacing the weaker 2-loop free pricing with stronger relaxations at first glance seems incompatible with the branch-and-price algorithm. The suggested branching rule on (non-)followers implies 2-loop elimination constraints on a new task set and k-loop elimination for $k \geq 3$ on another task set. Therefore, a new dynamic programming labeling algorithm was proposed in Chapter 3 for handling combined task-(k_1, k_2)-loop elimination (with $k_1, k_2 \geq 2$) in SPPRC for situations where loops with respect to two different task sets must be avoided. However, the crucial part for an effective labeling algorithm is the definition of a dominance relation. Applying straightforward approaches the possible number of labels grows exponentially. We presented update procedures for the attributes and effective dominance rules that can guarantee that, for fixed k_1 and k_2, the worst-case computational complexity for solving standard SPPRC and SPPRC with combined task-(k_1, k_2)-loop elimination is identical.

ng-route relaxation is another relaxation known from the node-routing that was adapted

to solve the CARP with a branch-and-price approach. In Chapter 4 we analyzed the impact of integrated acceleration techniques for the heuristic and exact solution of the pricing problems, such as bounding and bidirectional pricing. Loop elimination for $k = 3$ and $k = 4$ as well as the use of the ng-route revealed the superiority of the presented branch-and-price algorithm.

Embedding CARP in a more complex context. Different mail delivery modes proposed in Bodin and Levy (2000) were analyzed in Chapter 5 and the existing literature were classified to these categories. Extending the standard CARP by a location aspect results in a combined location and arc routing problem. One possibility of combining these two aspects is called park and loop mode. Different mathematical formulations for two park and loop problems were proposed and computational results show the performance of each formulation.

Bibliography

Achterberg, T., Koch, T., and Martin, A. (2005). Branching rules revisited. *Operations Research Letters*, **33**(1), 42–54.

Ahr, D. (2004). *Contributions to Multiple Postmen Problems*. Phd dissertation, Department of Computer Science, Heidelberg University, Heidelberg, Germany.

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

Amaya, C.-A., Langevin, A., and Trépanier, M. (2007). The capacitated arc routing problem with refill points. *Operations Research Letters*, **35**(1), 45–53.

Amaya, C.-A., Langevin, A., and Trépanier, M. (2010). A heuristic method for the capacitated arc routing problem with refill points and multiple loads. *Journal of the Operational Research Society*, **61**(7), 1095–1103.

Amberg, A., Domschke, W., and Voss, S. (2000). Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees. *European Journal of Operational Research*, **124**, 360–376.

Assad, A. A. and Golden, B. L. (1995). Chapter 5 arc routing methods and applications. In C. M. M.O. Ball, T.L. Magnanti and G. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, pages 375 – 483. Elsevier.

Balas, E. (1989). The prize collecting traveling salesman problem. *Networks*, **19**(6), 621–636.

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.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011a). Dynamic ng-path relaxation. Tromso, Norway. Route Conference 2011.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011b). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011c). New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*.

Bartolini, E., Cordeau, J.-F., and Laporte, G. (2013a). An exact algorithm for the capacitated arc routing problem with deadheading demand. *Operations Research*, **61**(2), 315–327.

Bartolini, E., Cordeau, J.-F., and Laporte, G. (2013b). Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming*, **137**(1-2), 409–452.

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

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

Belenguer, J. M., Benavent, E., Lacomme, P., and Prins, C. (2006). Lower and upper bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, **33**(12), 3363–3383.

Belenguer, J. M., Benavent, E., and Irnich, S. (2013). The capacitated arc routing problem: Exact algorithms. In Corberán and Laporte (2013), chapter 9. (In preparation.).

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

Benavent, E., Campos, V., Corberán, Á., and Mota, E. (1992). The capacitated arc routing problem: Lower bounds. *Networks*, **22**, 669–690.

Benavent, E., Ángel Corberán, and Sanchis, J. (2000). Linear programming based methods for solving arc routing problems. In Dror (2000), chapter 7, pages 231–275.

Bentley, J. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, **23**(4), 214–229.

Beullens, P., Muyldermans, L., Cattrysse, D., and van Oudheusden, D. (2003). A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, **147**(3), 629–643.

Bode, C. (2013). Lower bounds for park and loop delivery problems. Technical Report LM-2013-02, Chair of Logistics Management, Mainz School of Management and Economics, Johannes Gutenberg University, Mainz, Germany. Available at `http://logistik.bwl.uni-mainz.de/158.php`.

Bode, C. and Irnich, S. (2012). Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research*, **60**(5), 1167–1182.

Bode, C. and Irnich, S. (2013a). In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. accepted by Transportation Science.

Bode, C. and Irnich, S. (2013b). The shortest-path problem with resource constraints with $(k, 2)$-loop elimination and its application to the capacitated arc-routing problem. Technical Report LM-2013-01, Chair of Logistics Management, Johannes Gutenberg University Mainz, Mainz, Germany. Available online at `http://logistik.bwl.uni-mainz.de/158.php`.

Bodin, L. and Levy, L. (2000). Scheduling of local delivery carrier routes for the United States Postal Service. In Dror (2000), chapter 11, pages 419–442.

Boland, N., Dethridge, J., and Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, **34**(1), 58–68.

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, Á. and Laporte, G., editors (2013). *Arc Routing: Problems, Methods and Applications*. MOS-SIAM Series on Optimization. SIAM, Philadelphia. (In preparation.).

Corberán, Á. and Prins, C. (2010). Recent results on arc routing problems: An annotated bibliography. *Networks*, **56**(1), 50–69.

Del Pia, A. and Filippi, C. (2006). A variable neighborhood descent algorithm for a real waste collection problem with mobile depots. *International Transactions in Operational Research*, **13**(2), 125–141.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., and Villeneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 3, pages 57–93. Kluwer Academic Publisher, Boston, Dordrecht, London.

Desaulniers, G., Desrosiers, J., and Solomon, M. (2002). Accelerating strategies in column generation for vehicle routing and crew scheduling problems. In C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, Operations Research/Computer Science Interfaces Series, chapter 14, pages 309–324. Kluwer, Boston.

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. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, **42**(5), 977–978.

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.

Fischetti, M., Salazar-González, J.-J., and Toth, P. (2004). The generalized traveling salesman and orienteering problems. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, Combinatorial Optimization, chapter 12, pages 609–662. Springer US.

Fu, H., Mei, Y., Tang, K., and Zhu, Y. (2010). Memetic algorithm with heuristic candidate list strategy for capacitated arc routing problem. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

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.

Gavish, B. and Graves, S. C. (1978). The travelling salesman problem and related problems. Technical report, Massachusetts Institute of Technology, Operations Research Center.

Ghiani, G., Improta, G., and Laporte, G. (2001). The capacitated arc routing problem with intermediate facilities. *Networks*, **37**(3), 134–143.

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

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

Gouveia, L., Mourão, M. C., and Pinto, L. S. (2010). Lower bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, **37**, 692–699.

Hashemi Doulabi, S. H. and Seifi, A. (2013). Lower and upper bounds for location-arc routing problems with vehicle capacity constraints. *European Journal of Operational Research*, **224**(1), 189–208.

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. (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

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.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**(2), 497–511.

Kohl, N. (1995). *Exact methods for Time Constrained Routing and Related Scheduling Problems*. Dissertation, Department of Mathematical Modelling, Technical University of Denmark.

Kohl, N., Desrosiers, J., Madsen, O., Solomon, M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, **33**(1), 101–116.

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.

Larsen, J. (1999). *Parallelization of the Vehicle Routing Problem with Time Windows*. Ph.D. thesis, Department of Mathematical Modelling, Technical University of Denmark.

Letchford, A. N. (1997). *Polyhedral results for some arc routing problems*. Phd dissertation, Department of Management Science, Lancaster University.

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.

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

Levy, L. and Bodin, L. (1989). The arc oriented location routing problem. *Information Systems and Operations Research*, **27**(1), 74–94.

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

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

Martinelli, R., Pecin, D., Poggi de Aragão, M., and Longo, H. (2011). A branch-cut-and-price algorithm for the capacitated arc routing problem. In P. Pardalos and S. Rebennack, editors, *Experimental Algorithms*, volume 6630 of *Lecture Notes in Computer Science*, pages 315–326. Springer Berlin / Heidelberg.

Martinelli, R., Poggi de Aragão, M., and Subramanian, A. (2013). Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, **40**(8), 2145–2160.

McGill, R., Tukey, J. W., and Larsen, W. A. (1978). Variations of box plots. *The American Statistician*, **32**(1), 12–16.

Mei, Y., Tang, K., and Yao, X. (2009). A global repair operator for capacitated arc routing problem. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, **39**(3), 723–734.

Mingozzi, A., Bianco, L., and Ricciardelli, S. (1997). Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research*, **45**(3), 365–377.

Muyldermans, L. (2012). Personal Communication.

Nasiri, S. D. and Letchford, A. N. (2013). Pricing routines for vehicle routing with time windows on road networks. Technical Report The Management School, Lancaster University.

Nobert, Y. and Picard, J. (1996). An optimal algorithm for the mixed chinese postman problem. *Networks*, **27**, 95–108.

Pandi, R. and Muralidharan, B. (1995). A capacitated general routing problem on mixed networks. *Computers & Operations Research*, **22**(5), 465–478.

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

Polacek, M., Doerner, K., Hartl, R., and Maniezzo, V. (2008). A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, **14**, 405–423.

Prins, C. (2013). The capacitated arc routing problem: Heuristics. In Corberán and Laporte (2013), chapter 7. (In preparation.).

Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, **3**(3), 255–273.

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, **51**(3), 155–170.

Roy, S. and Rousseau, J. (1989). The Capacitated Canadian Postman Problem. *Information Systems and Operations Research*, **27**, 58–73.

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.

Salazar-Aguilar, M. A., Langevin, A., and Laporte, G. (2013). The synchronized arc and node routing problem: application to road marking. *Computers & Operations Research.*

Salhi, S. and Rand, G. (1989). The effect of ignoring routes when location depots. *European Journal of Operational Research*, **39**, 150–156.

Santos, L., Coutinho-Rodrigues, J., and Current, J. R. (2010). An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B*, **44**(2), 246 – 266.

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

Tang, K., Mei, Y., and Yao, X. (2009). Memetic algorithm with extended neighborhood search for capacitated arc routing problems. *Evolutionary Computation, IEEE Transactions on*, **13**(5), 1151–1166.

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

Wøhlk, S. (2008). A decade of capacitated arc routing. In R. Sharda, S. Voß, B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 29–48. Springer. 10.1007/978-0-387-77778-8_2.