CADERNOS DO LOGIS

Volume 2017, Number 6

Comparative Analysis of Capacitated Arc Routing Formulations for Branch-Cut-and-Price Algorithms

Diego Pecin, Eduardo Uchoa

September, 2017

LOGIS
Núcleo de Logística Integrada e Sistemas

Universidade Federal Fluminense

Comparative Analysis of Capacitated Arc Routing Formulations for Branch-Cut-and-Price Algorithms

Diego Pecin[a], Eduardo Uchoa[b,*]

[a] *H. Milton Stewart School of Industrial and Systems Engineering*
Georgia Institute of Technology - Atlanta - GA - USA 30332
[b] *Universidade Federal Fluminense - Departamento de Engenharia de Produção*
Rua Passo da Pátria 156, Niterói - RJ - Brasil - 24210-240

Abstract

The current best exact algorithms for the Capacitated Arc Routing Problem are based on the combination of cut and column generation. This work presents a deep theoretical investigation of the formulations behind those algorithms, classifying them and pointing similarities and differences, advantages and disadvantages. In particular, we discuss which families of cuts and branching strategies are suitable for each alternative and their pricing complexities. That analysis is used for justifying key decisions on constructing a new branch-cut-and-price algorithm, that combines several features picked from the capacitated arc routing literature with some features adapted from the most successful recent algorithms for node routing. The computational experiments show that the resulting algorithm is indeed effective and can solve almost all open instances from the classical benchmark sets.

Keywords: Arc Routing; Column Generation; Cutting Planes; Algorithmic Engineering

1. Introduction

The Capacitated Arc Routing Problem (CARP) is defined as follows. Let $G = (V, E)$ be a connected undirected graph where $V = \{0, \ldots, n\}$ is the vertex set and E is the edge set, $|E| = m$. Vertex 0 corresponds to the depot. Each edge $e \in E$ has a positive cost c_e and a non-negative integral demand d_e. The set of required edges is defined as $R = \{e \in E \,|\, d_e > 0\}$. Assume that identical vehicles with capacity Q are available at the depot. The goal is finding a minimum cost set of routes, closed walks starting and ending at the depot, that serve the demands in all required edges. Edges in a route can be traversed either serving or deadheading. The sum of the demands of the served edges in a route can not exceed Q. The number of routes may be fixed or not to a given number K. The CARP is the most classical multi-vehicle arc routing problem. It was first presented by Golden and Wong (1981) and has been used to model many situations, including street garbage collection, postal delivery, and routing of electric meter readers (Dror, 2012). In

*Corresponding author.
Email addresses: diego.pecin@isye.gatech.edu (Diego Pecin), uchoa@producao.uff.br (Eduardo Uchoa)

those applications, graph G corresponds to a road/street network and is always sparse and almost always planar (overpasses or underpasses may make it near-planar). There are cases where $R = E$, but there are also practical cases where a significant number of edges are non-required.

Many heuristic and exact algorithms have been proposed for its solution. We group the most successful exact methods into the following four categories:

1. Vehicle-indexed compact formulations, first proposed by Belenguer and Benavent (1998). Binary variables x_e^k indicate whether a vehicle k, $1 \leq k \leq K$, services required edge $e \in R$, and integer variables z_e^k counts how many times edge $e \in E$ is deadheaded by vehicle k. Computational experiments showed that such formulations suffer from symmetries and are only useful for small values of K.

2. Cutting plane algorithms based on the aggregated deadhead variables, proposed independently by Letchford (1996) and Belenguer and Benavent (1998). They use a single variable z_e, for each $e \in E$, to indicate how many times edge e is deadheaded. Known families of cuts include Odd Edge Cutset, CARP Rounded Capacity (Belenguer and Benavent, 1998) and Disjoint-Path (Belenguer and Benavent, 2003) inequalities. The resulting lower bounds are reasonably good, sometimes good enough to prove the optimality of heuristically found solutions. Moreover, since the LPs that have to solved are light, those bounds can be quickly obtained on small and medium-sized instances. Martinelli et al. (2013) proposed a dual ascent for accelerating the bound computation on large-sized instances. However, those methods can not be turned into a full exact algorithm because no complete CARP formulation over the aggregated deadhead variables is known. In fact, since it is NP-complete to check whether an integral z vector corresponds to a feasible CARP solution, it is unlikely that such formulation will be found.

3. Transformation to Capacitated Vehicle Routing Problem (CVRP). The transformation proposed by Pearn et al. (1987) turns a CARP instance into a CVRP on a complete graph with $3|R|+1$ nodes. The transformation proposed independently by Baldacci and Maniezzo (2006) and by Longo et al. (2006) produces a CVRP instance with $2|R| + 1$ nodes. The existence of sophisticated algorithms for CVRP makes the transformation quite practical: a branch-and-cut was used in Baldacci and Maniezzo (2006) and a branch-cut-and-price (BCP) in Longo et al. (2006). Actually, as will be discussed in Section 2, the BCP in Longo et al. (2006) was significantly specialized to take advantage of the CARP structure, becoming similar to other algorithms in Category 4. As will be also explained in Section 2, the transformation in Foulds et al. (2015) also leads to a BCP algorithm very similar to the one in Longo et al. (2006).

4. Algorithms based on the combination of column and cut generation. They are based on Set Partitioning Formulations (SPF) with a very large number of variables, corresponding

either to routes or to suitable route relaxations. Cuts may be added in order to tighten the linear relaxation. Those cuts may be those defined over the deadhead variables, those derived from an underlying CVRP structure or those derived from the SPF itself. Due to the stronger lower bounds, they are the current best exact algorithms for CARP.

This article is organized as follows. Section 2 contains a deep theoretical analysis of the known alternatives for building a column and cut generation algorithm for CARP, classifying them according to the underlying original formulation implicit in their pricing algorithm. That analysis fills an important gap in the literature, highlighting similarities among previously unrelated approaches, and also pointing potential advantages and disadvantages of each such formulation. This will be used for justifying the key decision of what formulation to use in the new proposed algorithm. Section 3 presents the full BCP algorithm, that also includes some of features previously only found in state-of-the-art node routing algorithms. Section 4 presents computational results, showing that 22 open instances can be now solved to optimality. In fact, not counting the recent benchmark Egl-large, only two classical benchmark CARP instances (F18 and egl-S4-A) are not solved. Section 5 contains some concluding comments. Detailed computational results are presented in the appendix.

2. CARP Formulations for Column and Cut Generation

We are going to show that all column and cut generation algorithms for CARP found in the literature can be related to one of the flow formulations presented next. All those formulations have a pseudo-polynomially large (depending on Q) number of variables and constraints. A Dantzig-Wolfe decomposition over each of those formulations lead to similar Master LP problems. However, the resulting pricing subproblems and their corresponding complexities differ. Other key properties, including the types of cuts that be can added and the possible branching strategies, also differ. It is important to note that none of those previous articles actually presented their algorithms in that way, the original pseudo-polynomial flow formulation is implicit in their dynamic programming pricing algorithms.

For each pair of vertices i and j in V, let $D(i,j)$ be the set of edges in a chosen cheapest path from i to j, having cost $C(i,j) = \sum_{e \in D(i,j)} c_e$. For each demand $r = \{u, v\} \in R$, define $o(r,u) = v$ and $o(r,v) = u$. The notation $o(r,x)$ means the endpoint of r *other* than x. We use the standard notation for graph cuts, however the graph itself is inferred from the context. If X is a subset of the vertex-set of a certain undirected graph, $\delta(X)$ is the subset of the edges of that graph with exactly one endpoint in X. If X is a subset of the vertex-set of a certain directed graph, $\delta^-(X)$ and $\delta^+(X)$ are the subsets of the arcs in that graph that enter and leave X, respectively.

2.1. Formulation 1

Define an acyclic directed graph $\mathcal{N}^1 = (\mathcal{V}^1 = \mathcal{R}^1 \cup \mathcal{O}^1, \mathcal{A}^1 = \mathcal{A}^1_1 \cup \mathcal{A}^1_2 \cup \mathcal{A}^1_3)$ with node-sets $\mathcal{R}^1 = \{(r,w,q) : r \in R; w \in r; q = d_r, \ldots, Q\}$ and $\mathcal{O}^1 = \{(0,0,q) : q = 0, \ldots, Q\}$. The arc-set $\mathcal{A}^1_1$

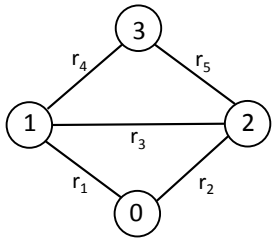

Figure 1: Illustrative instance: $d(r_4) = d(r_5) = 1$, $d(r_1) = 2$, $d(r_2) = d(r_3) = 3$, $Q = 5$.

is composed by all possible arcs that go from a node $(r_1, w, q) \in \mathcal{R}^1$ to a node $(r_2, t, q + d_{r_2}) \in \mathcal{R}^1$, represented as tuples (r_1, w, r_2, t, q). More formally, $\mathcal{A}_1^1 = \{(r_1, w, r_2, t, q) = ((r_1, w, q), (r_2, t, q + d_{r_2})) : r_1 \in R; w \in r_1; q = d_{r_1}, \ldots, Q - 1; r_2 \in R : q + d_{r_2} \leq Q; t \in r_2\}$. The cost c_a of an arc $a = (r_1, w, r_2, t, q) \in \mathcal{A}_1^1$ is defined as $C(w, t') + c_{r_2}$, where $t' = o(r_2, t)$. The arcs in arc-set $\mathcal{A}_2^1$ go from node $(0, 0, 0)$ to a node in $\{(r, w, d_r)) : r \in R; w \in r\}$ and will be represented as tuples of format $(0, 0, r, w, 0)$. The cost c_a of an arc $a = (0, 0, r, w, 0) \in \mathcal{A}_2^1$ is defined as $C(0, w') + c_r$, where $w' = o(r, w)$. Finally, the arcs in $\mathcal{A}_3^1$ go from a node $(r, w, q) \in \mathcal{R}_1$ to node $(0, 0, q)$, they are represented as $(r, w, 0, 0, q)$. The cost c_a of an arc $a = (r, w, 0, 0, q) \in A_3^1$ is defined as $C(w, 0)$. For each $r \in R$, define $\mathcal{R}_r^1 = \{(r', w, q) \in \mathcal{R}^1 : r' = r\}$. For each $a = (r_1, w, r_2, t, q) \in \mathcal{A}^1$ define a binary variable x_a^1 indicating that a vehicle collected the demand r_1 ending in vertex w with an accumulated load q, and then, took the cheapest deadhead path to collect the demand r_2 ending in vertex t with load $q + d_{r_2}$. The depot can be interpreted as a dummy null demand r_0 having both ends in vertex 0. The formulation follows:

$$\text{(F1) min} \quad \sum_{a \in \mathcal{A}^1} c_a x_a^1 \tag{1}$$

subject to

$$\sum_{a \in \delta^-(\{v\})} x_a^1 - \sum_{a \in \delta^+(\{v\})} x_a^1 = 0, \qquad \forall v = (r, w, q) \in \mathcal{R}^1, \tag{2}$$

$$\sum_{a \in \delta^-(\mathcal{R}_r^1)} x_a^1 = 1, \qquad \forall r \in R, \tag{3}$$

$$x^1 \geq 0, \tag{4}$$

$$x^1 \text{ integer.} \tag{5}$$

Equations (2) state that the same number of arcs enters and leaves a node in R_1. Equations (3) state that exactly one arc must enter in the subset of the nodes in $\mathcal{R}_1$ associated with a required edge $r \in R$, meaning that each r is served once. Consider the instance depicted in Figure 1. Assuming that $c(r_1) = c(r_3) << c(r_2) = c(r_4) = c(r_5)$, the solution with routes

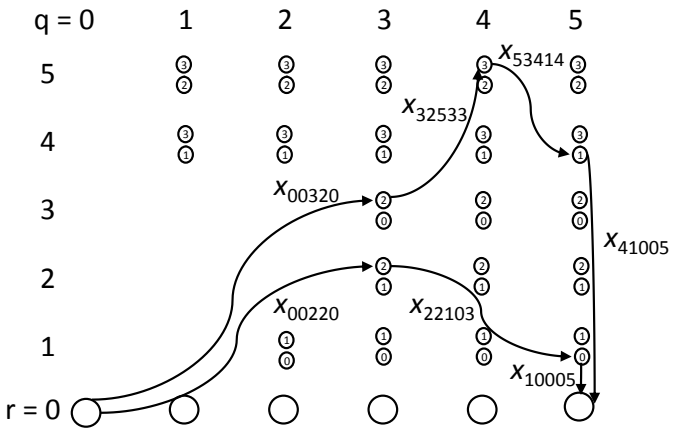

Figure 2: Solution $\{(0-1=2=3=1-0),(0=2-1=0)\}$ as a flow in $\mathcal{N}_1$.

$0-1=2=3=1-0$ and $0=2-1=0$ is optimal ("−" denotes deadheading, "=" denotes servicing). Figure 2 depicts $\mathcal{V}_1$ and the arcs in $\mathcal{A}_1$ that would have value 1 in that solution. For examples: $x^1_{00320}=1$ indicates that a vehicle departed from the depot empty, deadheaded $\{0,1\}$ (the only edge in the cheapest path between 0 and 1) and serviced $r_3=\{1,2\}$ ending at vertex 2 with load 3; $x^1_{32533}=1$ indicates that the vehicle that serviced r_3 ending at 2 with load 3 next serviced $r_5=\{2,3\}$ ending in vertex 3 with load 4. Since $D(2,2)=\emptyset$, there is no deadhead associated to that variable. Variables x^1_a where $a\in\mathcal{A}^1_3$ are not related to any service; they represent the final deadhead back to the depot. Variable $x^1_{100005}=1$ indicates that the vehicle that serviced demand $r_1=\{0,1\}$ ending at 0 with load 5, went next to the depot, also in vertex 0. Therefore, the cost of this variable is 0.

The pseudo-polynomially large number of variables and constraints in F1 makes its direct use unpractical. However, it can be rewritten in terms of paths in $\mathcal{N}_1$. Let Ω^1 be the set of all possible paths between node $(0,0,0)$ and another node in $\mathcal{O}_1$. For every $p\in\Omega_1$, define a variable λ_p. Define b^a_p as one if arc $a\in\mathcal{A}^1$ belongs to path p and zero otherwise. An equivalent formulation including both x^1 and λ variables is:

$$\min \quad \sum_{a\in\mathcal{A}^1} c_a x^1_a \tag{6}$$

subject to

$$\sum_{p\in\Omega^1} b^a_p\lambda_p - x^1_a = 0, \qquad \forall a\in\mathcal{A}^1, \tag{7}$$

$$\sum_{a\in\delta^-(\mathcal{R}^1_r)} x^1_a = 1, \qquad \forall r\in R, \tag{8}$$

$$x^1,\lambda\geq 0, \tag{9}$$

$$x^1 \text{ integer.} \tag{10}$$

Eliminating the x^1 variables, replacing their occurrences by their definition in terms of the λ variables given in (7), and relaxing the integrality, we get the following Master LP:

$$\min \quad \sum_{p\in\Omega^1} c_p\lambda_p \tag{11}$$

subject to

$$\sum_{p\in\Omega^1} b_p^r\lambda_p = 1, \qquad \forall r \in R, \tag{12}$$

$$\lambda \geq 0, \tag{13}$$

where $c_p = \sum_{a\in\mathcal{A}^1} b_p^a c_a$ is the cost of a path $p \in \Omega^1$ and $b_p^r = \sum_{a\in\delta^-(\mathcal{R}_r^1)} b_p^a$ is the number of times that r was serviced in that path. The Master LP (11–13) can also be obtained directly from (1–4) by a Dantzig-Wolfe decomposition. For that, the paths in Ω^1 should be viewed as the extreme rays of the unbounded conic polyhedron defined by (2) and (4) (that polyhedron has the null vector as its single extreme point, there is no need to associate a variable to that point). Anyway, (11–13) can be solved by a column generation where the subproblem consists of finding a minimum cost path in the directed acyclic graph $\mathcal{N}^1$, with respect to arc reduced costs calculated as:

$$\bar{c}_{(r_1,w,r_2,t,q)} = c_{(r_1,w,r_2,t,q)} - \pi_{r_2},$$

where π_{r_2} is the dual variable of the constraint in (12) corresponding to r_2, π_0 can be defined as 0. Minimum cost paths in acyclic graphs are obtainable in time proportional to the number of arcs, so the complexity of the pricing is $O(|R|^2Q)$. Any number of additional cuts defined over the x^1 variables can be introduced in the Master LP, after being translated using Equations (7). The dual variables of those cuts will only introduce additional terms in the arc reduced cost calculation, the complexity of the pricing remains unchanged. Therefore, according to the classification proposed in Poggi de Aragão and Uchoa (2003), those cuts are *robust*.

Remark that Ω^1 contains paths that are not elementary, in the sense that some required edge is served more than once. More precisely, a path $p \in \Omega^1$ is non-elementary iff $b_p^r > 1$ for some $r \in R$. Replacing Ω^1 by its subset Ω^1_{el} formed only by elementary paths would provide a stronger formulation. However, it would make the pricing subproblem strongly NP-hard (and often intractable in practice). So, it can be better to work over a set Ω^1_{elrel}, $\Omega^1_{el} \subset \Omega^1_{elrel} \subset \Omega^1$, known as an *elementarity relaxation*, that provides a good compromise between formulation strength and pricing complexity. The two most used options, both for node and arc routing problems, are:

- Allowing only routes without s-cycles, two services to the same required edge without at least s other services in-between. Eliminating routes with 1-cycles, two consecutive services to the same required edge, is trivially done in F1 by just removing some arcs from $\mathcal{A}^1_1$. Eliminating 2-cycles is still easy and only doubles the number of states in the dynamic programming. Removing s-cycles for $s \geq 3$ is more complex and increases the pricing

complexity by factors of up to $s^2 s!$ (Irnich and Villeneuve, 2006). In the CARP context, many authors refer to s-cycle elimination as $(s+1)$-loop elimination.

- Allowing only ng-paths (Baldacci et al., 2011). In the CARP context this means that each $r \in R$ should be associated to a set $NG(r) \subseteq R$, known as the neighborhood of r, usually formed by the ng-size (a chosen parameter) required edges that are closer to r. An ng-path may only re-service a required edge $r_1 \in R$ after it serves a required edge r_2 such that $r_1 \notin NG(r_2)$.

2.1.1. Properties of F1: Possibility of lifting known z cuts

In order to strength F1, any cut defined over the deadhead variables z can be converted into a cut over the x^1 variables. The conversion uses the following equalities:

$$z_e = \sum_{\substack{a = (r_1, w, r_2, t, q) \in \mathcal{A}^1 : \\ e \in D(w, o(t))}} x^1_a, \qquad \forall e \in E. \tag{14}$$

Of course, the resulting cut over the x^1 variables can be converted to the λ variables and then introduced in the DWM by using equations (7).

An important family of cuts over the z variables are the Odd Edge Cutsets (Belenguer and Benavent, 1998): for any $X \subseteq V$ such that $|\delta(X) \cap R|$ is odd,

$$\sum_{e \in \delta(X)} z_e \geq 1. \tag{15}$$

The inequality can be shown to be valid by the following reasoning. Just for servicing the edges in $\delta(X) \cap R$, vehicles must enter (and also leave) X at least $(|\delta(X) \cap R| + 1)/2$ times. Therefore, those vehicles must enter or leave X deadheading, at least once. Bartolini et al. (2013) realized that the direct conversion of (15) to the x^1 variables using (14),

$$\sum_{e \in \delta(X)} \sum_{\substack{a = (r_1, w, r_2, t, q) \in \mathcal{A}^1 : \\ e \in D(w, o(t))}} x^1_a = \sum_{a = (r_1, w, r_2, t, q) \in \mathcal{A}^1} |D(w, o(t)) \cap \delta(X)| x^1_a \geq 1, \tag{16}$$

is dominated by the following Lifted Odd Edge Cutset:

$$\sum_{\substack{a = (r_1, w, r_2, t, q) \in \mathcal{A}^1 : \\ |D(w, o(t)) \cap \delta(X)| \text{ is odd}}} x^1_a \geq 1. \tag{17}$$

The lifting uses the extra information available in the x^1 variables to assert that there will be a deadhead that is part of a deadhead path $D(w, o(t))$ such that $|\{w, o(t)\} \cap X| = 1$. Moreover, the coefficient of a variable $x^1_{r_1, w, r_2, t, q}$ is 1 even if $|D(w, o(t)) \cap \delta(X)|$ is an odd number greater than

1.

2.1.2. Properties of F1: Relation to CVRP Edge formulation

Define a complete undirected graph $H = (V_H, E_H)$ with vertex-set $V_H = \{0\} \cup R$. Associate a variable y_{uv} for each edge $(u, v) \in E_H$. A solution of F1 over variables x^1 can be projected onto variables y as:

$$y_{uv} = \sum_{\substack{a = (r_1, w, r_2, t, q) \in A_1 : \\ \{r_1, r_2\} = \{u, v\}}} x^1_a, \qquad \forall (u, v) \in E_H. \tag{18}$$

It can be seen that an integer y must be a solution of the Edge Formulation (a.k.a. Two-Index Formulation) (Laporte and Nobert, 1983) for a Capacitated VRP instance defined over H, with node demands given by the corresponding required edge demands and capacity Q. This means that any valid CVRP cut over that formulation can be translated to variables x^1 using (18) and used to strength F1. A very useful such family of cuts are the CVRP Rounded Capacity Cuts (RCCs):

$$\sum_{(u,v) \in \delta(X)} y_{uv} \geq 2 \lceil \frac{\sum_{e \in X} d_e}{Q} \rceil, \qquad \forall X \subseteq V_H \setminus \{0\}. \tag{19}$$

The CARP Rounded Capacity Cut, defined over the z variables, was proposed in Belenguer and Benavent (1998) as:

$$\sum_{e \in \delta(X)} z_e \geq 2 \lceil \frac{\sum_{e \in \delta(X) \cup E(X)} d_e}{Q} \rceil - |\delta(X) \cap R|, \qquad \forall X \subseteq V \setminus \{0\}. \tag{20}$$

It was proved in Bartolini et al. (2013) that CVRP RCCs (19) dominate CARP RCCs (20).

Another property of F1 related to its projection onto the CVRP Edge Formulation is that, if desired, branching can be performed only on the y variables (Longo et al., 2006). This means that (1)–(4) plus the constraint that the y variables defined by (18) are integer is a complete CARP formulation.

2.1.3. Previous works over F1

- The BCP presented in Longo et al. (2006) works over F1. That article proposes a generic reduction of the CARP to a CVRP over a complete graph with $2|R| + 1$ nodes (corresponding to the the two endpoints of each required edge plus the depot). However, the BCP in Fukasawa et al. (2006) is specialized for solving the resulting instance: (i) there is only a single constraint in the Master for each required edge, those constraints are equivalent to (12); (ii) the pricing only considers routes where the nodes corresponding to the two endpoints of a required edge are always visited in sequence, so the set of feasible routes becomes Ω^1. Actually, the pricing also performs a partial 3-cycle elimination: a requested edge can only

be serviced *in the same sense* after 3 other services, that may include servicing the same requested edge in the opposite sense. However, since the trivial full 1-cycle elimination is also performed, it is not possible to serve the same required edge in opposite senses consecutively. That BCP separates only CVRP cuts (Rounded Capacity Cuts, Strengthened Comb and Framed Capacity Cuts) and performs CVRP branching.

- The BCP in Martinelli et al. (2011) works over F1 and separates Odd Edge Cutsets and CARP RCCs. It only performs branching on the deadhead variables z, so it may get stuck in an integer solution over z that does not not correspond to a valid CARP solution. In that case, the algorithm only returns a valid lower bound.

- The method proposed in Bartolini et al. (2013) can also be viewed as working over F1. The authors propose a transformation of the CARP into an asymmetric GVRP (Generalized Vehicle Routing Problem) where each required node is represented by a cluster containing two nodes. The method computes a sequence of up to four lower bounds, the last and strongest bound LB4 corresponds to pricing *ng*-routes and to separating Lifted Odd Edge Cutsets, CVRP RCCs, and 3-Subset Row Cuts (Jepsen et al., 2008). The method does not perform branching. As proposed in Baldacci et al. (2008), it finishes by trying to enumerate all elementary routes with reduced cost smaller than the duality gap and (if the number of such routes is not too big) solving the set partitioning problem with those routes with a generic MIP solver.

- The BCP in Foulds et al. (2015) also works over F1. The article proposes a direct transformation of a CARP instance into a CVRP-like problem that *must* be solved by a BCP. However, the pricing subproblem in that algorithm should introduce two nodes for each required edge. The overall scheme can be viewed as a more streamlined version of the BCP in Longo et al. (2006).

- Martinelli et al. (2016) produced strong lower bounds with a column and cut generation algorithm over F1. They used *ng*-route relaxation and separate Odd Edge Cutsets, CARP RCCs and Subset Row Cuts with limited memory (Pecin et al., 2017b).

The complexity of the base pricing subproblem in all those works over F1 is the characteristic $O(|R|^2Q)$ *time.* The distinct schemes used for imposing partial elementarity in the routes correspond to different constant factors that do not change that complexity.

2.2. Formulation 2

Define the set $V_R \subseteq V$ containing all vertices that are endpoints of required edges in R. Vertex 0 may appear or not in V_R. Anyway, let $0'$ denote a copy of vertex 0. Vertex $0'$ will be used to represent the "depot itself", it will only appear in the start and in the end of every route. On the

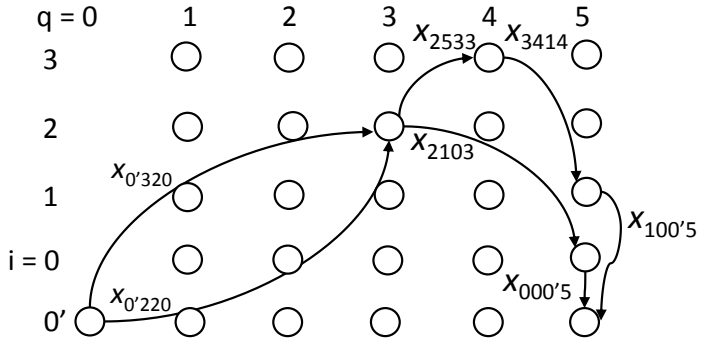

Figure 3: Solution $\{(0-1=2=3=1-0),(0=2-1=0)\}$ as a flow in $\mathcal{N}_2$.

other hand, vertex 0 represents the "street location just in front of the depot". It may appear, even more than once, in the middle of a route.

Define an acyclic multigraph $\mathcal{N}^2 = (\mathcal{V}^2 = \mathcal{R}^2 \cup \mathcal{O}^2, \mathcal{A}^2 = \mathcal{A}_1^2 \cup \mathcal{A}_2^2 \cup \mathcal{A}_3^2)$, with node-sets $\mathcal{R}^2 = \{(i,q) : i \in V_R; q = 1,\ldots,Q\}$ and $\mathcal{O}^2 = \{(0',q) : q = 0,\ldots,Q\}$. The arc-set $\mathcal{A}_1^2$ contains all arcs that can be defined as follows: from each node $(i,q) \in \mathcal{R}^2$, for every $r \in R$ and for every $j \in r$, if $(j, q+d_r)$ belongs to $\mathcal{R}^2$ then an arc denoted as (i,r,j,q) exists. Remark that $\mathcal{A}_1^2$ may contain parallel arcs. The cost of an arc $a = (i,r,j,q) \in \mathcal{A}_1^2$ is $c_a = C(i,j') + c_r$, where $j' = o(r,j)$. For every $r \in R$ and for every $j \in r$ there is an arc in $\mathcal{A}_2^2$ going from $(0',0)$ to (j,d_r), this arc will be represented as tuples of format $(0',r,j,0)$ and will cost $C(0,j') + c_r$, where $j' = o(r,j)$. Set $\mathcal{A}_2^2$ may also contain parallel arcs. Finally, an arc in $\mathcal{A}_3^2$ goes from a node $(i,q) \in \mathcal{R}^2$ to node $(0',q)$, it is represented as $(i,0,0',q)$ and costs $C(i,0)$.

For each $r \in R$, the set of arcs related to servicing r is defined as $\mathcal{A}^2(r) = \{(i,r',j,q) \in \mathcal{A}^2 : r' = r\}$. For each $a = (i,r,j,q) \in \mathcal{A}^2$ define a binary variable x_a^2 meaning that a vehicle at vertex i with an accumulated load q went (possibly with some deadheading) to collect the demand r ending in vertex t. The formulation follows:

$$\text{(F2)} \quad \min \quad \sum_{a \in \mathcal{A}^2} c_a x_a^2 \tag{21}$$

subject to

$$\sum_{a \in \delta^-(\{v\})} x_a^2 - \sum_{a \in \delta^+(\{v\})} x_a^2 = 0, \qquad \forall v = (i,q) \in \mathcal{R}^2, \tag{22}$$

$$\sum_{a \in \mathcal{A}^2(r)} x_a^2 = 1, \qquad \forall r \in R, \tag{23}$$

$$x^2 \geq 0, \tag{24}$$

$$x^2 \text{ integer}. \tag{25}$$

Consider the instance depicted in Figure 1 and the same solution considered before, with routes $0-1=2=3=1-0$ and $0=2-1=0$. Figure 3 depicts $\mathcal{V}^2$ and the arcs in $\mathcal{A}^2$ that would have value 1 in that solution. For examples: $x^2_{0'320}=1$ indicates that a vehicle departed from the depot empty, deadheaded $\{0,1\}$ and serviced $r_3=\{1,2\}$ ending at vertex 2 (with load $0+d(r_3)=3$); $x^2_{0'220}=1$ indicates that a vehicle departed from the depot empty and,without deadheading, serviced $r_2=\{0,2\}$ ending at vertex 2 (with load $0+d(r_2)=3$); $x^2_{2533}=1$ indicates that a vehicle that was at vertex 2 with load 3, without deadheading, served r_5 ending at 3 (with load $3+d(r_5)=4$). Remark that the solution $((0-1=2-1=0),(0=2=3=1-0))$ would have an identical representation in F2.

Let Ω^2 be the set of all possible paths in $\mathcal{N}^2$ between node $(0',0)$ and another node in $\mathcal{O}^2$. For every $p\in\Omega^2$, define a variable λ_p. Define b^a_p as one if arc $a\in\mathcal{A}^2$ belongs to path p and zero otherwise. Applying a Dantzig-Wolfe decomposition over F2, we get the following Master LP:

$$\min \quad \sum_{p\in\Omega^2} c_p\lambda_p \tag{26}$$

subject to

$$\sum_{p\in\Omega^2} b^r_p\lambda_p = 1, \qquad \forall r\in R, \tag{27}$$

$$\lambda \geq 0, \tag{28}$$

where $c_p=\sum_{a\in\mathcal{A}^2} b^a_p c_a$ is the cost of a path $p\in\Omega^2$ and $q^r_p=\sum_{a\in\mathcal{A}^2(r)} b^a_p$ is the number of times that r was serviced in that path. Although (26)-(28) looks very similar to (11)-(13), *the corresponding pricing subproblem can now be solved in $O(|V_R|.|R|.Q)$ time.*

- F1 and F2 are nearly equivalent if R is a matching of G, i.e., if $e\cap f=\emptyset$ for any $e,f\in R$. Otherwise, if required edges have common vertices, pricing paths in Ω^2 can be cheaper than pricing paths in Ω^1.
- On the other hand, F2 has drawbacks with respect to F1. Since its x^2 variables do not have information on the pairs of consecutive services, it is not possible to separate CVRP cuts or perform branch on the CVRP y variables, at least not without changing the pricing subproblem. It is still possible to separate cuts over the z variables and even to separate Lifted Odd Edge Cutsets.

There are no previous cut and column generation algorithms that can be viewed as working over F2. The formulation is introduced here because, as a natural intermediate between F1 and F3, it clarifies some explanations.

2.3. Formulation 3

Define a directed acyclic graph $\mathcal{N}^3=(\mathcal{V}^3=\mathcal{R}^3\cup\mathcal{O}^3,\mathcal{A}^3=\mathcal{A}^3_1\cup\mathcal{A}^3_2\cup\mathcal{A}^3_3\cup\mathcal{A}^3_4)$, with node-sets $\mathcal{R}_3=\{(i,q),(i,q'):i\in V_R\cup\{0\};q=0,\ldots,Q\}$ and $\mathcal{O}^3=\{(0',q):q=1,\ldots,Q\}\cup\{(0',0')\}$. The

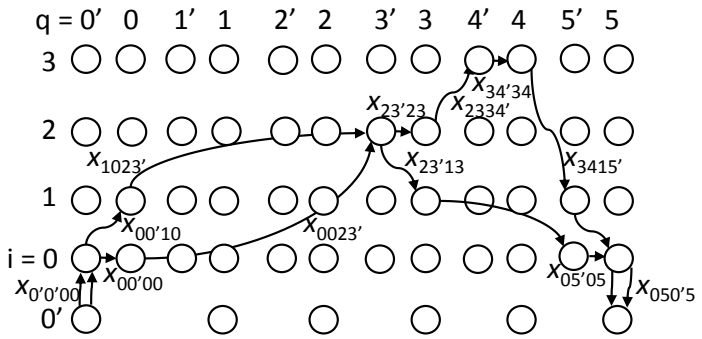

Figure 4: Solution $\{(0-1=2=3=1-0),(0=2-1=0)\}$ as a flow in $\mathcal{N}_3$.

arc-set $\mathcal{A}_1^3$ contains the service arcs, defined as $\{(i,q,j,(q+d_r)') = ((i,q),(j,(q+d_r)')),(j,q,i,(q+d_r)') = ((j,q),(i,(q+d_r)')) : \{i,j\} \in R; q = 0,\ldots,Q-d_r\}$, and the deadhead path arcs $\mathcal{A}_2^3$, defined as $\{(i,q',j,q) = ((i,q'),(j,q)) : (i,j) \in V_R \times V_R; q = 0,\ldots,Q\}$. Let $\mathcal{A}_1^3(r) \subseteq \mathcal{A}_1^3$ be the arcs associated to servicing a demand $r \in R$. The arc-sets $\mathcal{A}_3^3 = \{(0',0',0,0') = ((0',0'),(0,0')),(0',0',0,0) = ((0',0'),(0,0))\}$ and $\mathcal{A}_4^3 = \{(0,q,0',q) = ((0,q),(0',q)) : q = 1,\ldots,Q\}$ are formed by the arcs leaving and returning to the depot, respectively. The cost of an arc $a = (i,q_1,j,q_2) \in \mathcal{A}_1^3$ is $c_{\{i,j\}}$, the cost of an arc $a = (i,q',j,q) \in \mathcal{A}_2^3$ is $C(i,j)$. The remaining arcs cost are zero. For each $a \in \mathcal{A}^3$, define a binary variable x_a^3. The formulation follows:

$$\text{(F3) min} \quad \sum_{a\in\mathcal{A}^3} c_a x_a^3 \tag{29}$$

subject to

$$\sum_{a\in\delta^-(\{v\})} x_a^3 - \sum_{a\in\delta^+(\{v\})} x_a^3 = 0, \qquad \forall v = (i,q) \in \mathcal{R}^3, \tag{30}$$

$$\sum_{a\in\mathcal{A}_1^3(r)} x_a^3 = 1, \qquad \forall r \in R, \tag{31}$$

$$x^3 \geq 0, \tag{32}$$

$$x^3 \text{ integer.} \tag{33}$$

Consider again the solution with routes $0-1=2=3=1-0$ and $0=2-1=0$ for the instance in Figure 1. Figure 4 depicts $\mathcal{V}^3$ and the arcs in $\mathcal{A}^3$ that would have value 1 in that solution. It can be seen that paths in $\mathcal{N}^3$ alternate service arcs and deadhead arcs. For examples: $x_{00'10}^3 = 1$ indicates that a vehicle that was empty at vertex 0 deadheaded $\{0,1\}$; $x_{10'23}^3 = 1$ indicates that that vehicle serviced $r_3 = \{1,2\}$ ending at vertex 2 (with load $0 + d(r_3) = 3$), $x_{23'23}^3 = 1$ indicates "a null deadhead", etc.

Let Ω^3 be the set of all paths in $\mathcal{N}^3$ between node $(0',0')$ and another node in $\mathcal{O}^3$. For every $p \in \Omega^3$, define a variable λ_p. Define b_p^a as one if arc $a \in \mathcal{A}^3$ belongs to path p and zero otherwise.

A Dantzig-Wolfe decomposition over F3 yields the following Master LP:

$$\min \quad \sum_{p\in\Omega^3} c_p\lambda_p \tag{34}$$

subject to

$$\sum_{p\in\Omega^3} b_p^r\lambda_p = 1, \qquad \forall r \in R, \tag{35}$$

$$\lambda \geq 0, \tag{36}$$

where $c_p = \sum_{a\in\mathcal{A}^3} b_p^a c_a$ is the cost of a path $p \in \Omega^3$ and $q_p^r = \sum_{a\in\mathcal{A}_1^3(r)} b_p^a$ is the number of times that r was serviced in that path. *The pricing subproblem can be solved in $O(|V_R|^2.Q)$ time, characteristic of that formulation.* That complexity comes from the cardinality of the deadhead path set $\mathcal{A}_2^3$, since service set $\mathcal{A}_1^3$ has only $O(|R|.Q)$ arcs and $|R|$ is never larger than $|V_R|^2$.

- The cutting plane and column generation algorithm in Gómez-Cabrero et al. (2005) can be viewed as working over F3. Odd Edge Cutsets and CARP RCCs are separated. Remark that it is not trivial to perform even 1-cycle elimination in the pricing for F3, since two consecutive services are associated to non-adjacent arcs in $\mathcal{N}^3$. This is done in that work (they called it 2-loop elimination) using a dynamic programming algorithm that only doubles the number of states, as described in Benavent et al. (1992a).

2.4. Formulation 4

Define a directed graph $\mathcal{N}^4 = (\mathcal{V}^4 = \mathcal{R}^4 \cup \mathcal{O}^4, \mathcal{A}^4 = \mathcal{A}_1^4 \cup \mathcal{A}_2^4 \cup \mathcal{A}_3^4 \cup \mathcal{A}_4^4)$, with node-sets $\mathcal{R}^4 = \{(i,q) : i \in V; q = 0,\ldots,Q\}$ and $\mathcal{O}^4 = \{(0',q) : q = 0,\ldots,Q\}$. Let $\mathcal{R}^4(q) \subseteq \mathcal{R}^4$ be the subset of vertices associated with accumulated demand q. The arc-set $\mathcal{A}_1^4$ contains the service arcs, defined as $\mathcal{A}_1^4 = \{(i,q,j,q+d_r) = ((i,q),(j,q+d_r)),(j,q,i,q+d_r) = ((j,q),(i,q+d_r)) : \{i,j\} \in R; q = 0,\ldots,Q-d_r\}$, and the deadhead arcs $\mathcal{A}_2^4$, defined as $\{(i,q,j,q) = ((i,q),(j,q)),(j,q,i,q) = ((j,q),(i,q)) : \{i,j\} \in E; q = 0,\ldots,Q\}$. Let $\mathcal{A}_1^4(r) \subseteq \mathcal{A}_1^4$ be the arcs associated to servicing a demand $r \in R$ and let $\mathcal{A}_2^4(q) \subseteq \mathcal{A}_2^4$ be the deadhead arcs associated with accumulated demand q. The arc-sets $\mathcal{A}_3^4 = \{(0',0,0,0) = ((0',0),(0,0))\}$ and $\mathcal{A}_4^4 = \{(0,q,0',q) = ((0,q),(0',q)) : q = 1,\ldots,Q\}$ are formed by the arcs leaving and returning to the depot, respectively. The cost of an arc $a = (i,q_1,j,q_2) \in \mathcal{A}^4$ is $c_{\{i,j\}}$ (define $c_{\{0,0'\}} = 0$). For each $a \in \mathcal{A}^4$, define a binary variable x_a^4. The formulation follows:

$$\text{(F4)} \ \min \quad \sum_{a\in\mathcal{A}^4} c_a x_a^4 \tag{37}$$

subject to

$$\sum_{a\in\delta^-(\{v\})} x_a^4 - \sum_{a\in\delta^+(\{v\})} x_a^4 = 0, \qquad \forall v = (i,q) \in \mathcal{R}^4, \tag{38}$$

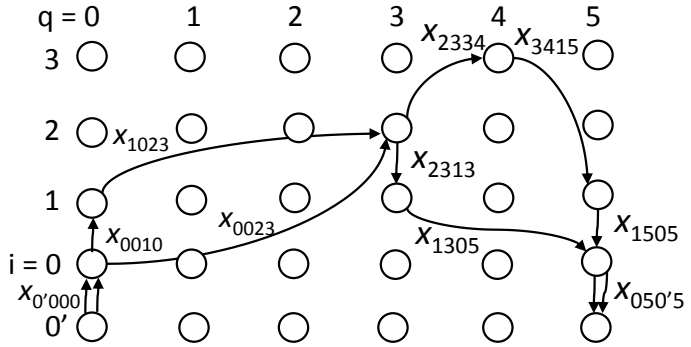

Figure 5: Solution $\{(0-1=2=3=1-0),(0=2-1=0)\}$ as a flow in $\mathcal{N}_4$.

$$\sum_{a\in\mathcal{A}_1^4(r)} x_a^4 = 1, \qquad \forall r \in R, \tag{39}$$

$$x^4 \geq 0, \tag{40}$$

$$x^4 \text{ integer}. \tag{41}$$

Consider again the instance in Figure 1 and the same solution used in previous examples. Figure 5 depicts $\mathcal{V}^4$ and the arcs in $\mathcal{A}^4$ that would have positive value in that solution. Remark that $x^4_{0'000}$ and $x^4_{050'5}$ have value 2.

The existence of cycles in $\mathcal{N}_4$ makes the Dantzig-Wolfe decomposition of F4 more complex. The unbounded conic polyhedral region defined by (38) and (40) has two sets of extreme rays: (i) Set Ω^4, composed by all paths in $\mathcal{N}_4$ from $(0',0)$ to another vertex in $\mathcal{O}^4$; (ii) Set Θ^4, composed by all simple cycles in $\mathcal{N}_4$, those cycles are formed only by arcs in the same $\mathcal{A}_2^4(q)$ set. For every $p \in (\Omega^4 \cup \Theta^4)$, define a variable λ_p. Define b_p^a as one if arc a belongs to set $p \subseteq \mathcal{A}^4$ and zero otherwise. The resulting Master LP is:

$$\min \qquad \sum_{p\in\Omega^4} c_p\lambda_p + \sum_{p\in\Theta^4} c_p\lambda_p \tag{42}$$

subject to

$$\sum_{p\in\Omega^4} b_p^r\lambda_p = 1, \qquad \forall r \in R, \tag{43}$$

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

where $c_p = \sum_{a\in\mathcal{A}^4} b_p^a c_a$ and $b_p^r = \sum_{a\in\mathcal{A}_1^4(r)} b_p^a$. The variables corresponding to the cycles in Θ^4 do not appear in (43) and can be ignored in the solution of that Master LP. For that reason, the reduced cost of all arcs in $\mathcal{A}_2^4$ is equal to their original costs, and therefore, non-negative. This is crucial for the efficient pricing of paths in Ω^4 proposed by Letchford and Oukil (2009), described below:

- Let $f(i,q)$ be the minimum reduced cost of a path in $\mathcal{N}^4$ from $(0',0)$ to (i,q) (∞ if no

such path exists) and let $g(i,q)$ be the minimum reduced cost of a path from $(0',0)$ to (i,q) that arrives in (i,q) by a service arc in $\mathcal{A}_1^4$ or directly from $0'$ (∞ if no such path exists). Naturally, $f(i,q) \leq g(i,q)$.

- At each stage q of the procedure, from $q=0$ to Q, there are two phases: (1) given that the values of $g(i,q)$ are already known, Dijkstra's algorithm over the arcs in $\mathcal{A}_2^4(q)$ is used for obtaining all $f(i,q)$ values in $O(|V|\log|V|+|E|)$ time. This is done by adding a dummy vertex v connected to vertices in $\mathcal{R}^4(q)$ by an arc with cost $g(i,q)$ and calculating the shortest paths from v to every vertex in $\mathcal{R}^4(q)$. Since all those paths passes by exactly one arc from v, it is possible to add the same positive constant to all $g(i,q)$ values, in order to avoid negative costs. (2) the calculated values of $f(i,q)$ are expanded over the arcs $(i,q,j,q+d_{ij})$ from $\mathcal{A}_1^4$ for updating the values of $g(j,q+d_{ij})$, this takes $O(|R|)$ time.

The overall complexity of the pricing is $O((|V|\log|V|+|E|).Q)$, characteristic of F4.

That relaxation can be strengthened with additional cuts. For example, cuts defined over the z variables, like Odd Edge Cutsets and CARP RCCs, can be converted into x^4 variables using the following equations:

$$z_e = \sum_{q=0}^{Q}(x^4_{iqjq} + x^4_{jqiq}), \qquad \forall e=\{i,j\}\in E. \tag{45}$$

Any cut over the x^4 variables can be introduced in the Master LP by applying the following transformation:

$$x^4_a = \sum_{p\in(\Omega^4\cup\Theta^4)} b^a_p \lambda_p, \qquad \forall a \in \mathcal{A}^4. \tag{46}$$

However, a complication arises. The new dual variables may cause arcs in $\mathcal{A}_2^4$ to have negative reduced costs, breaking the Dijkstra phases of the pricing. This can be handled using the following observation. Since the cuts defined over the z variables are symmetric, if some arc (i,q,j,q) has negative reduced cost, arc (j,q,i,q) also has the same negative reduced cost. Therefore, those two opposite arcs define a cycle p in Θ^4 with negative reduced cost. Adding the corresponding λ_p variable to the restricted Master LP would make those reduced costs zero in the next iteration.

Actually, in order to completely avoid negative reduced costs in $\mathcal{A}_2^4$ during the algorithm, Bode and Irnich (2012) add individual variables z_e, $e\in E$, to the initial restricted Master LP. This is equivalent to adding to the Master LP new variables λ_p, $p\in Z$, where each element of Z is the union of the $Q+1$ cycles of size 2 in Θ^4 corresponding to an edge $e\in E$. That is, $Z=\{Z_e=\{(i,q,j,q),(j,q,i,q):q=0,\ldots,Q\} : \{i,j\}\in E\}$. According to our definition of c_p, the variable λ_p where $p=Z_e$ has a large cost $2(Q+1).c_e$. But, according to (45) and (46), it also has large coefficients $2(Q+1).\alpha_e$ in a z cut with coefficient α_e on variable z_e. If desired, those costs and coefficients may be scaled by factor $1/(2(Q+1))$ without changing the LP value. Anyway,

the trick by Bode and Irnich is correct because neither the original arc costs nor the cuts over the z variables depend on q, so all arcs in the same Z_e have the same reduced cost. Therefore, the presence of the λ_{Z_e} variables in the restricted Master LP automatically avoids negative reduced costs in $\mathcal{A}_2^4$. Those authors present those additional variables as being Dual-Optimal Inequalities (Ben Amor et al., 2006). In fact, they can never have a non-zero value in any optimal integer solution, since deadhead cycles never appear in those solutions. *However, we remark that those variables can have positive values in fractional solutions; thus making the cuts over the z variables weaker and potentially decreasing the lower bounds obtained.*

It is not easy to propose a full branch-and-price algorithm over the Dantzig-Wolfe decomposition of F4, at least if one desires to keep its attractive pricing complexity as much as possible:

- Branching over constraints defined over the z variables (that includes branching over individual z variables) can be handled with the tricks above described for avoiding negative reduced costs in the Dijkstra phases of the pricing. However, as mentioned in Section 1, this is not a complete branching scheme. There are solutions where all z variables are integral with cost strictly smaller than the cost of the optimal CARP solution. Since the variables x^4 in F4 do not carry information on consecutive services, it is not possible to complete the scheme with branching on the CVRP variables (18), like it can be done in F1.

- Bode and Irnich (2012) proposed the so-called follower/non-follower branching scheme. Two chosen required edges $e, e' \in R$ are set to be followers in one branch ($f_{ee'} = 1$), meaning that the service of e must follow (perhaps with deadheading in between) the service of e' or vice-versa. On the other branch they are set to be non-followers ($f_{ee'} = 0$), meaning that e and e' can not be served consecutively in a route. The concept is indeed akin to branching on CVRP edges. However, follower/non-follower constraints should be implementing by modifying the pricing algorithm. Constraints of type $f_{ee'} = 0$ are handled by the same mechanism used for avoiding 1-cycles/2-loops. On the other hand, enforcing constraints of type $f_{ee'} = 1$ is more complex and involves replacing e and e' by 4 super-edges corresponding to the possible ways of serving those edges in sequence. When several follower/non-follower constraints are set, the resulting pricing algorithm becomes intricate. In fact, the complexity of that pricing may grow exponentially with the number of ($f_{ee'} = 1$) constraints. In practice, as the branching on follower/non-followers only starts to be performed deeper in the tree, after the branching over z variables is exhausted, that bad behavior is not likely to be observed.

- Bode and Irnich later proposed improvements on their branch-and-cut-and-price. In particular, they show how the pricing should be modified for handling follower/non-follower branching scheme together with s-loop elimination for $s \geq 3$ (Bode and Irnich, 2014). In Bode and Irnich (2015) they analyze, both in theory and in practice, several possible elementarity relaxations, including the combination of 2-loop elimination with ng-path.

Taking advantage of the insights brought by stating F4 explicitly, we propose a new branching rule that fully preserves the pricing algorithm. Branching on any constraint over the x^4 variables (including constraints over a single variable) associated to arcs in $\mathcal{A}_1^4 \cup \mathcal{A}_3^4 \cup \mathcal{A}_4^4$ does not affect the pricing at all, the new dual variables just introduce additional terms in the precomputation of arc reduced costs. On the other hand, a branching over a single variable associated to an arc in $\mathcal{A}_2^4$ may break the Dijkstra's phase of the pricing, by introducing an asymmetry in the reduced cost of opposite arcs. However, branching over a pairs of opposite arcs (i.e., enforcing that $x^4_{iqjq} + x^4_{jqiq}$ should be integer for some $\{i, j\} \in E$ and some q, $0 \leq q \leq Q$) would be OK. This happens because if opposite arcs have the same negative reduced cost, those reduced costs can be zeroed by pricing the variable in Θ^4 corresponding to that cycle. Anyway, as will be shown next, it is not necessary to branch over arcs in $\mathcal{A}_2^4$:

Theorem 1. *A complete branch-and-price algorithm over F4, even if cuts over the z variables are added, can be built by only branching on arcs in* $\mathcal{A}_1^4 \cup \mathcal{A}_3^4 \cup \mathcal{A}_4^4$.

Proof: Suppose that a BCP node obtains a solution x^4, with value $c(x^4)$, such that x^4_a is integral for every $a \in (\mathcal{A}_1^4 \cup \mathcal{A}_3^4 \cup \mathcal{A}_4^4)$ but fractional for some variables x_a, $a \in \mathcal{A}_4^2$. We show that it is possible to convert x^4 into a feasible integral solution x'^4 with value $c(x'^4)$ not larger than $c(x^4)$. For every $a \in (\mathcal{A}_1^4 \cup \mathcal{A}_3^4 \cup \mathcal{A}_4^4)$, set $x'^4_a = x^4_a$. For each $q = 0$ to Q, solve a minimum cost flow problem over the network $(\mathcal{R}^4(q), \mathcal{A}_2^4(q))$, where arcs have their original cost and infinity capacity, and the demand of a node $(i, q) \in \mathcal{R}^4(q)$ is given by the integer values $\sum_{a \in (\delta^-((i,q)) \cap (\mathcal{A}_1^4 \cup \mathcal{A}_3^4))} x^4_a - \sum_{a \in (\delta^+((i,q)) \cap (\mathcal{A}_1^4 \cup \mathcal{A}_4^4))} x^4_a$. By the Min-Cost Flow Integrality Theorem, there must be an integral optimal solution. That solution is used for setting $x'^4(q)$, the part of the solution vector associated to the arcs in $\mathcal{A}_2^4(q)$. Moreover, the possibly fractional solution $x^4(q)$ to that flow problem, given by the restriction of x^4 to the arcs in $\mathcal{A}_2^4(q)$, can not cost less than $c(x'^4(q))$. □

The above branching scheme still has drawbacks. As the original graph G is undirected, one may reverse the sequence of edges in a route without changing its cost. A route and its reversal correspond to two usually distinct paths p and $p' = rev(p)$ in Ω^4. This means that a first branch of format $x^4_a = 0$, where $a \in (\mathcal{A}_1^4 \cup \mathcal{A}_3^4 \cup \mathcal{A}_4^4)$ will probably not move the lower bounds. This will happen if for each fractional route p that uses a, $rev(p)$ does not use a. Additional branches of format $x^4_a = 0$ will eventually move the bounds. Anyway, the number of nodes in the branch-and-bound tree is likely to be larger than the number of nodes obtained by a branching scheme that is not affected by that kind of route symmetry, like the follower/non-follower scheme.

3. Branch-Cut-and-Price Algorithm

3.1. Choosing a Formulation

This subsection provides the arguments that justified our decision of designing the BCP proposed in this article over F1.

3.1.1. Complexity of the pricing subproblems

We compare the previously presented formulations in terms of their characteristic pricing complexity. This is important, as the pricing time is often the bottleneck in the performance of BCP algorithms for vehicle routing problems (Poggi and Uchoa (2014)). For general CARP instances, its clear that the complexity of the pricing in F3 can be better than in F2, which in turn can be better than the complexity of F1. In fact, $O(|V_R|^2.Q)$ can be much better than $O(|R|^2.Q)$ if G is a dense graph. However, as discussed in Section 1, known practical CARP applications yield instances that are defined over graphs that are planar or near-planar and with few (if any) parallel edges. The following result is the justification for favoring F1 over F2 or F3:

Theorem 2. *Assuming that G is a simple planar graph, the complexities of the pricing subproblems in F2 and F3 are not asymptotically better than the complexity of the pricing in F1.*

Proof: If $G = (V, E)$ is simple and planar, subgraph $G = (V_R, R)$ is also simple and planar for any $R \subseteq E$. It is well-known that the maximum average degree of a simple planar graph is strictly less than 6, so $|R| < 3|V_R|$. Therefore, the complexity of the pricing in F1 is reduced to $O(|V_R|^2.Q)$. The pricing complexity in F2 is reduced to $O(|V_R|^2.Q)$ and does not change in F3. □

Assuming that G is simple and planar, the complexity of the pricing in F4 is reduced to $O((|V| \log |V|).Q)$. This is not always better than $O(|V_R|^2.Q)$. Assume the most favorable case for F4, instances where $V = V_R$. That second assumption is not always reasonable in practice, but holds for the instances from the literature. Indeed, a large proportion of the vertices are endpoints of some required edge in those instances. Anyway, under those two assumptions, the resulting $O((|V_R| \log |V_R|).Q)$ complexity of F4 is asymptotically better than the $O(|V_R|^2.Q)$ of F1.

3.1.2. Advantages of F1 over F4

In spite of the advantage of F4 over F1 in terms of pricing complexity, our design decision was based on the following list of advantages of F1:

1. Elimination of 1-cycles (a.k.a. 2-loops) in F1 is trivial and for free. The same elimination in F4 is not trivial and doubles the number of states in the dynamic programming used in the pricing. In general, as discussed in length in Bode and Irnich (2015), imposing elementary relaxations (s-cycle elimination or ng-path) in F4 lead to intricate algorithms and may increase pricing times by larger constant factors.

2. It possible to separate stronger Lifted Odd Edge Cutsets in F1.

3. In F1, it is possible to separate any cut valid for the CVRP Edge Formulation. In particular, it is possible to separate CVRP RCCs, that are stronger than CARP RCCs.

4. In F1, branching over the CVRP edges is complete and does not change the pricing. In F4, the related concept of follower/non-follower branching can be used. However, this requires complex adaptations in the pricing algorithms and results in an exponential worst-case complexity. The alternative branching proposed in this article is not completely satisfactory either, as it is prone to symmetry problems.

5. The extra variables (or Dual-Optimal cuts) required for not breaking the Dijkstra phase in the fast pricing for F4 may make cuts over the deadhead z variables weaker.

6. The fixing by reduced cost can be stronger in F1 than in F4. As a single variable $x^1_{r_1,w,r_2,t,q}$ corresponds to quite long sequence of decisions (when compared to a single variable in x^4), it is easier to prove that it can be fixed to zero. This means that a larger proportion of x^1 variables would be fixed *even with the same integrality gap*. Moreover, for the 5 reasons presented before, the gaps with F1 are likely to be smaller, further reducing the differences in the pricing time between the two formulations.

3.2. Limited Memory Rank-1 Cuts

The proposed BCP separates Lifted Odd Edge Cutsets and CVRP Rounded Capacity Cuts. However, it may also separate Limited Memory Rank-1 Cuts. Jepsen et al. (2008) introduced the Subset Row Cuts (SRCs), a family of cuts defined over the Set Partitioning Formulation that is obtained in many VRPs after a Dantzig-Wolfe decomposition. Those cuts can be generalized to Rank-1 Cuts (R1Cs). In the context of F1, they are defined as a Chvátal-Gomory rounding of some rows in (12). Given $S \subseteq R$ and a positive multiplier s_r for each $r \in S$, the following R1C is valid:

$$\sum_{p \in \Omega^1} \left\lfloor \sum_{r \in S} s_r b_p^r \right\rfloor \lambda_p \leq \left\lfloor \sum_{r \in S} s_r \right\rfloor . \tag{47}$$

The SRCs are the particular R1Cs obtainable with multipliers of format $1/k$, for some integer k. Pecin et al. (2017c) investigated the Set Partitioning Polyhedron and determined the optimal sets of R1C multipliers, for $|S|$ up to 5. R1Cs are usually strong and have the potential for increasing lower bounds substantially. However, since they are defined directly over the λ variables (and not over the x^1 variables), those cuts are *non-robust* (Poggi de Aragão and Uchoa, 2003). This means that each added cut changes the pricing subproblem, making it harder.

In order to mitigate the negative impact of those cuts in the pricing subproblem, Pecin et al. (2017c) proposed the concept of cuts with limited-memory. A R1C over a subset $S \subseteq R$ and with multiplier vector s, is associated to a memory node set $M(S,s)$, $S \subseteq M(S,s) \subseteq R$. The idea is that, when a route $p \in \Omega^1$ services a required edge not in $M(S,s)$, it may "forget" previous services to required edges in S, yielding a coefficient for λ_p in the cut that may be smaller than the one in (47). The memory set is defined, after the separation of a violated R1C, as a minimal set that preserves the coefficients of the variables λ_p with positive value in the current linear

relaxation solution. Of course, variables priced later may not have the best possible coefficients in previous cuts, but this can be corrected in the next separation rounds. Eventually, limited-memory R1Cs achieves the same bounds that would be obtained by the original R1Cs. Yet, those limited-memory R1Cs, while still non-robust, are better handled by the labeling algorithm used for solving the pricing subproblem. In order to further reduce the impact of R1Cs, Pecin et al. (2017a) introduced the generalized concept of memory on arc/edge sets. In the CARP case, a R1C is associated with a memory edge set $EM(S,s) \subseteq E_H$; a route that makes a sequence of services not in $EM(S,s)$ may "forget" previous services in S. Consider a R1C over a required edge subset $S \subseteq R$ and with an $|S|$-dimensional vector of multipliers s. The corresponding Limited Memory Rank-1 Cut, with associated memory set $EM(S,s)$ is defined as:

$$\sum_{p \in \Omega^1} \alpha(S,s,EM(S,s),p)\lambda_p \leq \left\lfloor \sum_{r \in S} s_r \right\rfloor, \tag{48}$$

where coefficient $\alpha(S,s,EM(S,s),p)$ of variable λ_p is computed as described in Algorithm 1, explained as follows. The algorithm follows the sequence of nodes and edges in $H(p)$, the path in $H = (V_H, E_H)$ corresponding to a route $p \in \Omega^1$. Whenever $H(p)$ passes by a node in S, its multiplier is added to the *state* variable. When $state \geq 1$, *state* is decremented and α is incremented. If $EM(S,s) = E_H$, the procedure would always return $\lfloor \sum_{r \in S} s_r q_p^r \rfloor$ and the limited-memory cut would be equivalent to the original cut. On the other hand, if $EM(S,s) \subset E_H$, it may happen that $H(p)$ passes by an edge not in $EM(S,s)$ when $state > 0$, causing *state* to be reset to zero and "forgetting" some previous visits to nodes in S. In this case, the returned coefficient may be less than the original coefficient.

Algorithm 1 Computing coefficient $\alpha(S,s,EM(S,s),p)$

```
1: α ← 0, state ← 0
2: for every edge (r1, r2) in path H(p) (in sequence) do
3:     if (r1, r2) ∉ EM(S, s) then
4:         state ← 0
5:     if r2 ∈ S then
6:         state ← state + s_{r2}
7:         if state ≥ 1 then
8:             α ← α + 1, state ← state − 1
9: return α
```

3.3. Pricing Algorithm and Fixing by Reduced Costs

The pricing algorithm for F1 consists in finding shortest paths in $\mathcal{N}^1$. However, the algorithm is complicated by the need of allowing only *ng*-paths and also by the possible presence of the non-robust limited memory R1Cs, described in the previous section. Cuts defined over the x^1, like Lifted Odd Edge Cutsets, CVRP Rounded Capacity Cuts or the branching constraints used

in this work, are invisible to the pricing. They only affect the calculation of the reduced costs $\bar{c}_{r_1,w,r_2,t,q}$ for each arc in $\mathcal{A}^1$.

The forward dynamic programming labeling algorithm represents an ng-feasible partial path $P = ((0,0,0),\ldots,(r,w,q))$ in $\mathcal{N}^1$ as a *label* $L(P) = (\bar{c}(P), r(P) = r, v(P) = w, q(P) = q, \Pi(P), state(P), pred(P))$ storing its reduced cost, end required edge, end vertex, load, set of required edges forbidden as immediate extensions due to ng-sets, vector of states corresponding to the nR active lm-R1Cs in the current Master LP solution, and a pointer to its predecessor label. For each active lm-R1C l, $1 \leq l \leq nR$, $S[l]$, $s[l]$ and $EM[l]$ denotes its base set, multiplier set and edge memory set, respectively.

Each $(r,w,q) \in \mathcal{V}^1$ defines a *bucket* $F(r,w,q)$. A label $L(P)$ is stored in bucket $F(r(P), v(P), q(P))$. A label $L(P_1)$ dominates a label $L(P_2)$ if every feasible completion of P_2 yields a route with reduced cost not smaller than the feasible route obtained by applying the same completion into P_1. The following conditions, together, are sufficient to ensure such a domination:

$$(i)\ r(P_1) = r(P_2), v(P_1) = v(P_2), \quad (ii)\ q(P_1) = q(P_2), \quad (iii)\ \Pi(P_1) \subseteq \Pi(P_2), \text{ and}$$

$$(iv)\ \bar{c}(P_1) \leq \bar{c}(P_2) + \sum_{1 \leq l \leq nR : state(P_1)[l] > state(P_2)[l]} \sigma_l,$$

where $\sigma_l < 0$ is the dual variable associated to lm-R1C l. Remark that the fact that some x^1 variables may have been fixed to zero by reduced cost in previous iterations prevents strengthening (ii) to $q(P_1) \leq q(P_2)$. This happens because, if $q(P_1) \neq q(P_2)$, due to the fixing, a feasible completion for P_2 may be infeasible for P_1. The second term in the right-hand side of (iv) is an upper bound on what a completion of P_2 can gain over the same completion of P_1, by avoiding the penalizations of servicing edges in $S[l]$ for lm-R1Cs l in which $state(P_1)[l] > state(P_2)[l]$. Only non-dominated labels are kept in the buckets. To accelerate the checking for dominated labels, it is convenient to keep labels of the same bucket ordered by reduced cost.

Define $NG(0)$ as $\emptyset$. For every other $r \in V_H$, $NG(r) \subseteq R$ includes the ng-size required edges closest to r, ties broken arbitrarily. The distance between $r_1 = \{u,v\}$ and $r_2 = \{w,t\}$ is given by $\min\{C(u,w), C(u,t), C(v,w), C(v,t)\}$. Algorithm 2 presents the pseudocode of the Forward Labeling procedure. In the end of the algorithm, each non-empty bucket $F(0,0,q)$, $1 \leq q \leq Q$, will contain only one label, representing the minimum reduced cost route with load q.

The actual algorithm used in pricing uses bidirectional labeling, as proposed by Righini and Salani (2006). The forward labeling algorithm is executed until q is equal to a value q_f, a similar backward labeling algorithm is also executed from Q to q_f, the minimum reduced cost routes are then obtained by a concatenation phase. The original bidirectional labeling sets q_f as the middle value $Q/2$. However, as described in Pecin et al. (2017b), the performance can be improved by executing forward and backward steps "in parallel", in order to balance the number of forward and backward labels. In fact, the resulting q_f will be the value that minimizes the absolute difference

Algorithm 2 Procedure Forward Labeling

$F(r,w,q) \leftarrow \emptyset, \forall (r,w,q) \in \mathcal{V}^1$
$F(0,0,0) \leftarrow \{(0,0,0,0,\emptyset,\mathbf{0},nil)\}$
for $q = 0$ to Q **do**
  for all (r_1,w) such that $(r_1,w,q) \in \mathcal{R}^1$ **do**
    for all (r_2,t) such that $(r_1,w,r_2,t,q) \in \mathcal{A}^1$ and $x^1_{r_1,w,r_2,t,q}$ is not fixed to 0 **do**
      for all $L_1 = (\bar{c}_1, r_1, w, q, \Pi_1, state_1, \_) \in F(r_1,w,q)$ **do**
        if $r_2 \notin \Pi_1$ **then**
          $\bar{c}_2 \leftarrow \bar{c}_1 + \bar{c}_{r_1,w,r_2,t,q}$
          $state_2 \leftarrow state_1$
          for $l = 1$ to nR **do**
            if $\{r_1,r_2\} \notin EM(l)$ **then** $state_2[l] \leftarrow 0$
            else if $r_2 \in S(l)$ **then**
              $state_2[l] \leftarrow state_2[l] + s_{r_2}[l]$
              if $state_2[l] \geq 1$ **then** $\bar{c}_2 \leftarrow \bar{c}_2 - \sigma_s$, $state_2[l] \leftarrow state_2[l] - 1$
          $L_2 \leftarrow (\bar{c}_2, r_2, t, q + d_{r_2}, (\Pi_1 \cap NG(r_2)) \cup \{r_2\}, state_2, \text{pointer to } L_1)$
          $insertLabel \leftarrow$ **true**
          for all $\mathcal{L} \in F(r_2, t, q + d_{r_2})$ **do**
            if L_2 dominates $\mathcal{L}$ **then** delete $\mathcal{L}$
            else if $\mathcal{L}$ dominates L_2 **then** $insertLabel \leftarrow$ **false**, **break**
          if $insertLabel$ **then**
            $F(r_2, t, q + d_{r_2}) \leftarrow F(r_2, t, q + d_{r_2}) \cup \{L_2\}$

between the final number of forward and backward labels. That strategy was extensively tested in Tilk et al. (2017), with positive results.

The fixing by reduced costs follows the same scheme used in Pecin et al. (2017b). A full run of both forward and backward labeling algorithms is performed. This allows calculating the minimum reduced cost of a route that uses each particular arc $a \in \mathcal{A}^1$, if that cost is larger than the gap then x^1_a can be fixed to zero. This will speedup the next pricing iterations.

3.4. Route Enumeration and Strong Branching

The algorithm by Bartolini et al. (2013) does not perform branching. Instead, like proposed in Baldacci et al. (2008), in the end of the root node it tries to enumerate all elementary routes in Ω^1 with reduced cost smaller than the duality gap, with respect to some known upper bound. If the set T of enumerated routes is not too large, the problem is finished by a MIP solver, that receives a set partitioning problem containing only those routes. However, if T is too large for the MIP solver,the algorithm halts without solving the instance. Contardo and Martinelli (2014) proposed an improvement by still performing enumeration even if T has up to several million routes, those routes are stored in a pool. From that point, the pricing starts to be done by inspection in the pool. The non-elementary routes are removed and more rounds of non-robust cut separation are performed, thus reducing gaps, and, therefore, also reducing the number of routes in the pool. At some point, a set partitioning problem of reasonable size may be given to the MIP solver.

As in Pessoa et al. (2009), our BCP uses a hybrid strategy, using both enumeration to a pool and strong branching. In the end of a node, enumeration may be tried. However, it it fails (i.e.,

$|T|$ would be too large), branching is performed. We used three different kinds of branching, all of them can be expressed as constraints over the x^1variables, so they do not affect the pricing:

1. Branching on the degree of a vertex i in G. As $|\delta(\{i\}) \cap R| + \sum_{e \in \delta(\{i\})} z_e$ must be even, $\sum_{e \in \delta(\{i\})} z_e$ must be even if $|\delta(\{i\}) \cap R|$ is even, and odd otherwise.
2. Branching on a single variable z_e.
3. Branching on the CVRP variables y, as defined in (18).

There is no predefined priority, we let the strong branching mechanism choose among branching candidates from those three kinds. It was observed that a candidate from the first kind of branching is often chosen in the top levels of the search tree, the second kind being the second most frequent. Deeper in the tree, when few candidates from the other kinds remain, most chosen branchings are from the third kind. Those branching alternatives are similar to those used in Bode and Irnich (2012, 2014, 2015), with the difference the third kind of branching in those previous works are based on the concept of followers/non-followers.

The hierarchical strong branching procedure, inspired by those found in Røpke (2012) and Pecin et al. (2017b), has the following phases:

- The Phase Zero performs the first selection of $\min\{100, TS(v)\}$ branching candidates, where $TS(v)$ is an estimate of the size of the subtree rooted in v based on the node gap and the average bound improvements obtained in previous branchings, $TS(v) = \infty$ for the root node.
- The Phase One performs a quick evaluation of each candidate by solving the current restricted Master LP twice, adding the constraint corresponding to each child node. Column and cut generation are not performed. The candidates are scored by the product rule (Achterberg, 2007) and the $\min\{3, \lceil TS(v)/10 \rceil\}$ best candidates go to Phase Two.
- Phase Two performs more precise evaluations of each candidate, doing heuristic column generation (but no cut generation) on both child nodes.

The whole procedure is guided by the principle that the strong branching effort in a node should depend on the expected subtree size. The rationale is the following. If $TS(v)$ is large, even a small improvement in that branching will compensate the cost of a more precise evaluation of several candidates. On the other hand, if $TS(v)$ is small, branching should be fast. The estimation of $TS(v)$ is done by the model proposed in (Kullmann, 2009; Le Bodic and Nemhauser, 2017).

4. Computational Experiments

The resulting BPC algorithm was coded in C++ and solves the linear and integer programs using IBM CPLEX Optimizer 12.6. All experiments were conducted in a single thread of an Intel

Xeon ES-2637 v2 3.5GHz with 128GB RAM, running Linux Oracle Server 6.7. The tests were performed in the following sets of benchmark instances:

- The 24 Eglese instances (Eglese and Li, 1992) are based on real-world data from winter gritting problems in Lancashire (UK). The graphs have from 98 to 190 edges, the number of required edges ranges from 51 to 190. This is the only set of instances that consistently appears in all articles from the recent CARP literature.
- The 34 Val instances (aka BCCM instances, after the authors of Benavent et al. (1992b)) are constructed over sparse random graphs ranging from 39 to 97 edges, all of them are required.
- The 100 BMCV instances (Beullens et al., 2003) are derived from road networks in Flanders (Belgium). The largest number of required edges is 121. The BMCV instances are divided into 4 classes of 25 instances: C, D, E and F. Instances C and E have $Q = 300$, instances D and F are the same except that the vehicle capacity is doubled to $Q = 600$. As all edge costs are multiples of 5, all solution values are multiples of 5.

In all those instances there is a number K fixing the number of routes in a solution. Our tables do not present results on the 6 KSHS and on the 23 GDB instances, since those older benchmarks became too easy for modern algorithms. Just for information, our BCP solves them in the average time of 0.12 seconds (KSHS) and 0.21 seconds (GDB). For the opposite reason, we also do not present results on the recent set Egl-large (Brandão and Eglese, 2008). Those instances, ranging from 347 to 375 required edges are beyond the reach of current exact algorithms. The BCP proposed in this paper was actually run with two different configurations:

1. The first configuration, used for classes Eglese, C, and E, includes the separation of Limited Memory R1Cs. The enumeration is aborted if the limit of 3 million non-dominated forward labels (partial elementary paths) is reached, since this forecasts that far too many elementary routes will be generated.
2. The second configuration, used on classes D, F and Val, does not separate non-robust cuts. Since routes in those classes are longer, Limited Memory R1Cs are less effective and may increase too much the pricing time. In the second configuration, the limit for enumeration is 300,000 non-dominated labels.

The other parameters are the same for both configurations. We used ng-size=8. Enumeration to the pool is only tried in a node if the gap is smaller than 0.15%. A node is only solved as a MIP when the number of routes in the pool is less than 25,000. So, if in the end of a node solution there are still more than that number of routes in the pool, branching is performed.

Detailed results for the proposed BCP and an instance-by-instance comparison with previous algorithms are given in the appendix. Table 1 is a summary comparison containing the following

information. Columns `RGap` are the average percent gap in the root node. Those gaps are calculated from the root lower bounds, with respect to the same best known upper bounds, including the two improved upper bounds presented in this article. Columns `RT` are the average root times (in seconds), over the indicated processors. In order to provide an indication of the relative speed of each processor, we report in parenthesis the scores from the PassMark site (https://cpubenchmark.net/singleThread.html). Columns `Opt` indicate the number of instances solved to optimality by the complete method, either using branching, enumeration or both. The count always includes cases where the algorithm finds a lower bound that proves that an existing upper bound is optimal. The previous algorithms are the following:

- **LPU06** refers to the BCP over F1 in Longo et al. (2006).
- **BI12** refers to the Cut-First Branch-and-Price Second algorithm in Bode and Irnich (2012), composed of two phases. Phase 1 is a cut generation algorithm over the aggregated deadhead variables. Phase 2 is a branch-and-price over F4 that includes the cuts generated in Phase 1. The authors present root node lower bounds, but only Phase 1 times. We use those times for calculating `RT`. We believe that the approximation is reasonable since the column generation in the root node is likely to converge quickly, because no additional cuts are being added. The authors used a time limit of 4 hours for the Phase 2. This method solved all instances in Val class for the first time.
- **BLC13** refers to the cut and column generation algorithm over F1 in Bartolini et al. (2013). This method obtained particularly good results on the Eglese instances, solving 10 instances of that class.
- **BI14** refers to the improved Cut-First Branch-and-Price Second algorithm presented in Bode and Irnich (2014), that can price routes without k-loops for $k \geq 3$ in a way that is compatible with the follower/non-follower branching scheme. The authors give separate results for different values of k. `Opt` indicates the number of instances solved within the time limit of 4 hours for some $k \in \{2, 3, 4\}$. However, Column `RGap` correspond only to results for $k = 4$. Root times are not available in the article.
- **FLM15** refers to the BCP over F1 in Foulds et al. (2015).
- **BI15** refers to the Cut-First Branch-and-Price Second algorithms in Bode and Irnich (2015), where eight possible combinations of loop elimination and ng-paths in the pricing are discussed and extensively tested. Moreover, different strong branching alternatives are also tested. We recommend that the interested reader consult that rich paper for more details. Anyway, Column `Opt` indicates the number of instances solved by any of the variants. In most of the runs, the time limit was set to 4 hours. However, for a number of harder BMCV instances the authors also run a chosen variant for up to 12 hours. Columns `RGap` and `RT`

	LPU06			BI12			BCL13		
Class/NP	RGap	RT	Opt	RGap	RT	Opt	RGap	RT	Opt
Eglese/24	0.57	1334	2	0.77	582	5	0.30	3268	10
Val/34	0.23	1939	26	0.24	11	34	0.16	755	29
C/25							0.41	2293	14
D/25							0.30	1629	19
E/25							0.26	2860	14
F/25							0.10	1468	19
Total/158			28			39			105
Processor	Pentium IV 2.4 GHz (?)			i7-2600 3.4GHz (1921)			Xeon E5310 1.6GHz (639)		

	BI14		FLM15			BI15		
Class/NP	RGap (k=4)	Opt	RGap	RT	Opt	RGap (ng7)	RT (ng7)	Opt
Eglese/24	0.61	6	0.96	61	4	0.51	2601	7
Val/34								
C/25	0.88	17				0.83	215	20
D/25	0.44	22				0.41	3978	23
E/25	0.57	19				0.51	304	23
F/25	0.13	23				0.13	4610	23
Total/158		87			4			96
Processor	i7-2600 3.4GHz (1921)		Core i3 3.0GHz (1764)			i7-2600 3.4GHz (1921)		

	MMP16		This BCP		
Class/NP	RGap	RT	RGap	RT	Opt
Eglese/24	0.29	2228	0.21	2216	23
Val/34			0.18	45	34
C/25			0.29	462	25
D/25			0.30	74	25
E/25			0.17	489	25
F/25			0.08	55	24
Total/158					156
Processor	i7-3960X 3.3GHz (1988)		Xeon E5-2637 3.5 GHz (1817)		

Table 1: Summary of CARP algorithms based on cut and column generation.

were taken from variant $ng7$, pricing routes without 2-loops and ng-paths with dynamic neighborhoods of size 7. That variant produces the smallest root gaps, but it is also the most expensive. The variant 2-loop produce the largest root bounds, but root times are much smaller. The other variants are in-between, with different trade-offs.

- **MMP16** refers to the to the cut and column generation algorithm over F1 in Martinelli et al. (2016). Since that algorithm does not includes a branching/enumeration scheme, only Columns `RGap` and `RT` are present.

It can be seen in Table 1 that the algorithm in BLC13 and the family of algorithms in BI12, BI14 and BI15 were the non-dominated exact methods for CARP. The F1-based algorithm in BLC13 uses heavier bounding mechanisms and indeed obtains smaller root gaps in all classes, having the best overall performance in the Eglese instances. On the other hand, the F4-based algorithms by Bode and Irnich use lighter bounding mechanisms and a more flexible branching scheme, obtaining the best results on the Val and BMCV instances. The newly proposed F1-based BCP obtains root bounds even better than those from BLC13, but the incorporation of other elements like the concept of limited-memory cuts and the hybridization of strong branching with enumeration makes it more robust and able to outperform existing algorithms on both Eglese and BMCV instances.

5. Conclusion

The analysis of VRP column and cut generation algorithms starting from the original pseudo-polynomial flow formulation implicit in their dynamic programming pricing algorithms can provide important insights. In this work, it was instrumental for classifying all existing column and cut generation algorithms for CARP, pointing similarities between previously unrelated approaches and also making their differences more clear. In particular, the analysis was used for justifying key algorithmic engineering decisions on creating a new method that tried to combine the "best of CARP" with some important developments on recent CVRP and VRPTW algorithms. The obtained results indicate very significant progress with respect to previous algorithms, 22 out of the 24 open instances (excluding the Egl-large set) in the literature could be solved.

Acknowledgments

We thank Teobaldo Bulhões for proofreading this report.

References

Achterberg, T., 2007. Constraint integer programming. Ph.D. thesis, Technische Universitat Berlin.

Baldacci, R., Christofides, N., Mingozzi, A., 2008. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. Mathematical Programming 115 (2), 351–385.

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

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

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

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

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

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

Benavent, E., Campos, V., Corberán, A., Mota, E., 1992a. The capacitated arc routing problem: lower bounds. Networks 22 (7), 669–690.

Benavent, E., Campos, V., Corberan, A., Mota, E., 1992b. The Capacitated Arc Routing Problem: Lower Bounds. Networks 22, 669–690.

Beullens, P., Muyldermans, L., Cattrysse, D., 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., Irnich, S., 2012. Cut-first branch-and-price-second for the capacitated arc-routing problem. Operations research 60 (5), 1167–1182.

Bode, C., Irnich, S., 2014. The shortest-path problem with resource constraints with (k, 2)-loop elimination and its application to the capacitated arc-routing problem. European Journal of Operational Research 238 (2), 415–426.

Bode, C., Irnich, S., 2015. In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. Transportation Science 49 (2), 369–383.

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

Chen, Y., Hao, J.-K., Glover, F., 2016. A hybrid metaheuristic approach for the capacitated arc routing problem. European Journal of Operational Research 253 (1), 25–39.

Contardo, C., Martinelli, R., 2014. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. Discrete Optimization 12, 129–146.

Dror, M., 2012. Arc routing: theory, solutions and applications. Springer.

Eglese, R., Li, L., 1992. Efficient routeing for winter gritting. Journal of the Operational Research Society, 1031–1034.

Foulds, L., Longo, H., Martins, J., 2015. A compact transformation of arc routing problems into node routing problems. Annals of Operations Research 226 (1), 177–200.

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

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

Gómez-Cabrero, D., Belenguer, J., Benavent, E., 2005. Cutting plane and column generation for the capacitated arc routing problem. Proceeding of ORP3 Meeting, Valencia.

Irnich, S., 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., Pisinger, D., 2008. Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research 56 (2), 497–511.

Kullmann, O., 2009. Fundaments of branching heuristics. Handbook of Satisfiability, A. Biere, M. Heule, H. van Maaren and T. Walsh (Eds), IOS Press, 205–244.

Laporte, G., Nobert, Y., 1983. A branch and bound algorithm for the capacitated vehicle routing problem. OR Spectrum 5 (2), 77–85.

Le Bodic, P., Nemhauser, G., 2017. An abstract model for branching and its application to mixed integer programming. Mathematical Programming, 1–37.

Letchford, A., 1996. Polyhedral results for some constrained arc-routing problems. Ph.D. thesis, University of Lancaster.

Letchford, A., 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., Uchoa, E., 2006. Solving capacitated arc routing problems using a transformation to the CVRP. Computers & Operations Research 33 (6), 1823–1837.

Martinelli, R., Malta, M., Poggi, M., 2016. Improved lower bounds using lm-SRCs for the capacitated arc routing problems. Presentation in International Workshop on Column Generation.

Martinelli, R., Pecin, D., Poggi, M., Longo, H., 2011. A branch-cut-and-price algorithm for the capacitated arc routing problem. In: International Symposium on Experimental Algorithms. Springer, pp. 315–326.

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

Pearn, W. L., Assad, A., Golden, B. L., 1987. Transforming Arc Routing into Node Routing Problems. Computers & Operations Research 14 (4), 285–288.

Pecin, D., Contardo, C., Desaulniers, G., Uchoa, E., 2017a. New enhancements for the exact solution of the vehicle routing problem with time windows. INFORMS Journal on Computing 29 (3), 489–502.

Pecin, D., Pessoa, A., Poggi, M., Uchoa, E., 2017b. Improved branch-cut-and-price for capacitated vehicle routing. Mathematical Programming Computation 9, 61–100.

Pecin, D., Pessoa, A., Poggi, M., Uchoa, E., Santos, H., 2017c. Limited memory rank-1 cuts for vehicle routing problems. Operations Research Letters 45 (3), 206–209.

Pessoa, A., Uchoa, E., Poggi de Aragão, M., 2009. A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. Networks 54 (4), 167–177.

Poggi, M., Uchoa, E., 2014. Vehicle Routing: Problems, Methods, and Applications. Vol. 18 of MOS-SIAM Series on Optimization. SIAM, Philadelphia, Ch. New exact algorithms for the capacitated vehicle routing problem, pp. 59–86.

Poggi de Aragão, M., Uchoa, E., 2003. Integer program reformulation for robust branch-and-cut-and-price. In: Wolsey, L. (Ed.), Annals of Mathematical Programming in Rio. Búzios, Brazil, pp. 56–61.

Righini, G., 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.

Røpke, S., 2012. Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems. Presentation in Column Generation 2012.

Tilk, C., Rothenbächer, A.-K., Gschwind, T., Irnich, S., 2017. Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. European Journal of Operational Research 261 (2), 530–539.

Vidal, T., 2017. Node, edge, arc routing and turn penalties: Multiple problems, one neighborhood extension. Operations Research.

Appendix A. Detailed results

Tables A.2-A.7 present detailed results for the new BCP. Columns `Ins` gives the instance name. Columns `IUB` present the initial upper bound used in the BCP, taken from the recent heuristic literature (Chen et al., 2016; Vidal, 2017) (except on instance egl-s4-B, as will be explained later). Values in bold indicate that the bound was already proved to be optimal in some previous work. Columns `OPT` shows the optimal solution value proved by the BCP. An underlined value indicates a solution that improves upon the best published solution. The following columns are root node information. The first of these columns indicates the lower bound (`RLB`) obtained by only separating robust cuts: Lifted Odd Edge Cutsets and CVRP Rounded Capacity Cuts. For the classes where the BCP was run with Configuration 1, columns `NRLB` and `nR` are the improved non-robust lower bounds and the number of active R1Cs, respectively. Next columns are the percentage of variables x^1 fixed to zero by reduced cost (`Fix`) and the accumulated time (in seconds) up to this point (`T1`). Then, the number of routes enumerated to the pool (`|R|i`), "not

tried" means that enumeration at the root was not tried because the gap was larger than 0.15%, "limit" indicates that the enumeration was aborted because the label limit was exceeded. When enumeration succeeds, `NLB` gives the improved root node lower bound, obtained by the removal of non-elementary routes and by additional rounds of separation, `|R|f` is the final number of routes in the pool, `T2` is the time spent with route enumeration plus the time to go from lower bound `NRLB` to lower bound `NLB` (or only the time spent with enumeration, if it is aborted) and, finally, the time spent solving the resulting MIP (`TMIP`) complete the information of the root. The BCP only calls the MIP if at most 25,000 routes are in the pool. Otherwise, branching is performed. Finally, Column `Nds` gives the number of nodes in the branch-and-bound tree and `TT` gives the total time in seconds. The rightmost columns indicate whether the instance was already solved in each previous work. If so, the total time spent is given. In cases of BI12, BI14 and BI15, the data in those columns are more complex.

- The runs in those works do not use external upper bounds. The algorithm itself tries to find a feasible solution and prove its optimality in the given time limit. However, when the algorithm finishes by time limit, the authors indicate whether the lower bound obtained is enough for proving the optimality of some previously published solution.

- BI14 contains results on 3 variants. Unless stated otherwise, the reported total times are from 4-loop variant. BI15 contains results on up to 15 variants corresponding to different elementarity relaxations and combinations of strong branching. We report $\leq 4h$ and $\leq 12h$ to state that some variant solved the instance in less than 4 and 12 hours, respectively.

In order to avoid those complications, we refrain from comparing average total times from BI12, BI14 and BI15 with the average total times of the proposed BCP. On the other hand, we believe that comparing the number of solved instances is fair, because we are counting instances where any variant of the algorithms in BI14 and BI15 could prove that an existing upper bound is optimal in the given time limit.

Our BCP could not solve instance egl-s4-B with its best published upper bound of 16,201 (Chen et al., 2016). We performed 350 runs (with different seeds) of the heuristic code HGS-CARP (https://github.com/vidalthi/HGS-CARP), recently presented in Vidal (2017). The overall cpu time was 4.2 days. In only one of those runs, HGS-CARP found an improving solution with value 16,187. Then, the BCP could prove its optimality in 3.9 days. We also made similar long runs of HGS-CARP on egl-s4-A, but they could not improve the best published upper bound of 12,216. The BCP could prove a lower bound of 12,196 by exploring 275 nodes, taking 1.85 days. The other instance not solved by BCP was F18. The root lower bound is 3063, on the first branch the left node is conquered and the right node has a lower bound of 3065. The corresponding solution is integral over the z variables but fractional over the x^1 variables, so only CVRP branching is possible. Due to some kind of symmetry, even with strong branching, BCP explores thousands of nodes without improving the bound of 3065.

Ins	IUB	OPT	RLB	NRLB	nR	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	LPU06	BI12	BCL13	BI14	FLM15	BI15
egl-e1-A	**3548**	**3548**	3548	3548	0	100	33						1	33	144	1621	16	589	775	$\leq$ 4h
egl-e1-B	**4498**	**4498**	4474	4492	129	98	389	6265	4494	2291	1.7	0.6	1	392		1836	2620	3827	2513	$\leq$ 4h
egl-e1-C	**5595**	**5595**	5553	5565	74	88	1386	57833	5580	5697	101	6.9	1	1494			733		9715	
egl-e2-A	**5018**	**5018**	5018	5018	0	100	81						1	81		1991		6345		$\leq$ 4h
egl-e2-B	**6317**	**6317**	6294	6302	84	94	425	limit			24		3	629						$\leq$ 4h
egl-e2-C	**8335**	**8335**	8288	8307	108	92	489	323297	8323	19323	631	16	1	1137			4204			
egl-e3-A	**5898**	**5898**	5898	5898	0	100	233						1	233	924	844	73	761		$\leq$ 4h
egl-e3-B	7775	**7775**	7704	7744	191	90	10649	not tried					17	35702						
egl-e3-C	10292	**10292**	10196	10233	126	81	465	not tried					65	5590						
egl-e4-A	6444	**6444**	6395	6410	131	85	5248	not tried					163	94657						
egl-e4-B	8961	**8961**	8904	8930	163	90	1958	not tried					17	8530						
egl-e4-C	11529	**11529**	11473	11498	107	95	263	not tried					9	609						
egl-s1-A	**5018**	**5018**	5016	5018	1	94	209						1	209		8670	1232	4312		$\leq$ 4h
egl-s1-B	**6388**	**6388**	6382	6385	28	100	82	482	6386	156	0.5	0.1	1	83			344	12250		$\leq$ 4h
egl-s1-C	**8518**	**8518**	8506	8514	26	99	44	289	8516	190	0.3	0.1	1	44			196		20879	
egl-s2-A	9874	**9868**	9826	9842	231	91	9955	not tried					145	94774						
egl-s2-B	13057	**13057**	12993	13021	215	93	2988	not tried					53	76094						
egl-s2-C	**16425**	**16425**	16376	16395	73	96	285	1469631	16400	682896	194		25	872			15083			
egl-s3-A	10201	**10201**	10148	10167	199	92	6310	not tried					145	89406						
egl-s3-B	13682	**13682**	13627	13651	207	90	2989	not tried					67	51609						
egl-s3-C	**17188**	**17188**	17121	17152	126	95	551	1787522	17169	111541	291		3	913			13202			
egl-s4-A	12216		12144	12168	248	86	3990	not tried												
egl-s4-B	16187	**16187**	16090	16142	322	86	5695	not tried					315	337962						
egl-s4-C	20461	**20461**	20403	20443	220	98	936	142585	20452	9330	61	18	1	1016						
Solved														**23**	**2**	**5**	**10**	**6**	**4**	**7**
Average time over the 10 instances also solved by BCL13														**541**			**3770**			
Geometric mean time over the 10 instances also solved by BCL13														**281**			**830**			

Table A.2: Detailed results on Eglese instances.

Ins	IUB	OPT	RLB	NRLB	nR	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BCL13	BI14	BI15
C01	**4150**	**4150**	4105	4133	93	95	281	not tried					7	934			$\leq$ 12h

C02	**3135**	**3135**	3135	3135	0	100	5.3						1	5.3	5	60	$\leq$ 4h
C03	**2575**	**2575**	2564	2569	30	96	40	101301	2575	0	133		1	174	548	252	$\leq$ 4h
C04	**3510**	**3510**	3478	3494	27	94	95	not tried					5	188		4403	$\leq$ 4h
C05	**5365**	**5365**	5338	5365	13	98	10						1	10	2462	272	$\leq$ 4h
C06	**2535**	**2535**	2523	2525	12	95	28	not tried					3	63	1278	196	$\leq$ 4h
C07	**4075**	**4075**	4024	4033	60	85	239	not tried					13	387	827	724	$\leq$ 4h
C08	**4090**	**4090**	4071	4090	46	100	12						1	12	95	634	$\leq$ 4h
C09	5260	**5260**	5234	5239	29	93	134	not tried					17	10343			
C10	4700	**4700**	4635	4660	81	95	30	not tried					21	100	128	1493	$\leq$ 4h
C11	**4630**	**4630**	4582	4595	137	90	2570	not tried					39	22889			
C12	**4240**	**4240**	4185	4209	85	90	137	not tried					3	193			$\leq$ 12h
C13	**2955**	**2955**	2919	2939	61	93	59	not tried					3	102	4200	589	$\leq$ 4h
C14	**4030**	**4030**	4016	4020	20	98	8.2	21735	4020	14598	0.8	1.9	1	11	381		$\leq$ 4h
C15	4940	**4940**	4913	4924	51	96	161	not tried					11	788			
C16	**1475**	**1475**	1474	1475	0	100	1.5						1	1.5	24	196	$\leq$ 4h
C17	**3555**	**3555**	3555	3555	0	100	1.6						1	1.6	3	23	$\leq$ 4h
C18	5605	**5605**	5578	5584	51	94	963	not tried					27	9391			
C19	**3115**	**3115**	3104	3115	30	100	31						1	31		6706	$\leq$ 4h
C20	**2120**	**2120**	2120	2120	0	100	5.8						1	5.8	12	392	$\leq$ 4h
C21	**3970**	**3970**	3960	3963	11	94	53	limit			5.6		3	75		3284	$\leq$ 4h
C22	**2245**	**2245**	2245	2245	0	100	4.3						1	4.3	5	256	$\leq$ 4h
C23	4085	**4085**	4038	4072	119	97	6214	not tried					7	7783			
C24	**3400**	**3400**	3384	3392	39	99	227	393610	3393	188054	96		3	388		2325	$\leq$ 4h
C25	**2310**	**2310**	2310	2310	0	100	1.5						1	1.5	2	20	$\leq$ 4h
Solved														**25**	**14**	**17**	**20**

Table A.3: Detailed results on C instances.

Ins	IUB	OPT	RLB	NRLB	nR	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BCL13	BI14	BI15
E01	4910	**4910**	4867	4886	225	94	574						21	2053			
E02	**3990**	**3990**	3972	3981	100	99	31	2064	3981	1378	0.3	0.7	1	32	252	862	$\leq$ 4h
E03	**2015**	**2015**	2015	2015	0	100	6.3						1	6.3	9	23	$\leq$ 4h
E04	**4155**	**4155**	4146	4155	19	99	30						1	30	2062	2789	$\leq$ 4h
E05	**4585**	**4585**	4579	4585	15	98	7.4						1	7.4	61	68	$\leq$ 4h

E06	**2055**	**2055**	2055	2055	0	100	4.0						1	4.0	4	54	$\leq$ 4h
E07	**4155**	**4155**	4112	4126	66	94	135						9	197	319	6684	$\leq$ 4h
E08	**4710**	**4710**	4691	4702	8	93	13	2802672	4702	119577	5759		3	5781	200	198	$\leq$ 4h
E09	5810	**5810**	5780	5787	58	95	231						13	1712			
E10	**3605**	**3605**	3605	3605	0	100	2.5						1	2.5	4	48	$\leq$ 4h
E11	**4650**	**4650**	4637	4644	98	98	414	15587	4644	9790	4.0	2.9	1	421		$\leq$ 4h	$\leq$ 4h
E12	**4180**	**4180**	4139	4161	32	95	33						5	52	6585		$\leq$ 4h
E13	**3345**	**3345**	3316	3334	89	97	69						3	101	1362	437	$\leq$ 4h
E14	**4115**	**4115**	4115	4115	0	100	5.9						1	5.9	67	2089	$\leq$ 4h
E15	**4205**	**4205**	4190	4194	94	96	1028						7	3224			$\leq$ 12h
E16	**3775**	**3775**	3763	3775	51	98	89						1	89		380	$\leq$ 4h
E17	**2740**	**2740**	2740	2740	0	100	2.3						1	2.3	3	10	$\leq$ 4h
E18	**3835**	**3835**	3825	3828	90	96	2174						7	2648			$\leq$ 4h
E19	**3235**	**3235**	3226	3235	35	100	47						1	47		13935	$\leq$ 4h
E20	**2825**	**2825**	2800	2807	48	86	292						3	378		3112	$\leq$ 4h
E21	**3730**	**3730**	3729	3730	0	100	20						1	20		$\leq$ 4h	$\leq$ 4h
E22	**2470**	**2470**	2467	2470	0	100	5.8						1	5.8	149	74	$\leq$ 4h
E23	**3710**	**3710**	3690	3710	178	100	1161						1	1161			$\leq$ 4h
E24	**4020**	**4020**	4005	4011	7	97	81						7	186		13135	$\leq$ 4h
E25	**1615**	**1615**	1615	1615	0	100	0.7						1	0.7	1	2	$\leq$ 4h
Solved														**25**	**14**	**19**	**23**

Table A.4: Detailed results on E instances.

Ins	IUB	OPT	RLB	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BCL13	BI14	BI15
D01	**3215**	**3215**	3215	100	90						1	90	84	4117	$\leq$ 4h
D02	**2520**	**2520**	2520	100	12						1	12	19	286	$\leq$ 4h
D03	**2065**	**2065**	2065	100	12						1	12	49	1472	$\leq$ 4h
D04	**2785**	**2785**	2785	100	24						1	24	70	9022	$\leq$ 4h
D05	**3935**	**3935**	3935	100	20						1	20	22	166	$\leq$ 4h
D06	**2125**	**2125**	2125	100	6.8						1	6.8	16	1615	$\leq$ 4h
D07	3115	**3115**	3054	83	20	not tried					45	247	10446		
D08	**3045**	**3045**	3004	84	45	not tried					25	328		(2-loop)3730	$\leq$ 4h
D09	**4120**	**4120**	4120	100	52						1	52	103	1654	$\leq$ 4h

D10	**3340**	**3340**	3333	98	15	524575	3334	1065	18	0.1	1	33	107	493	$\leq$ 4h
D11	**3745**	**3745**	3745	100	71						1	71	123	11009	$\leq$ 4h
D12	**3310**	**3310**	3310	100	105						1	105	78	198	$\leq$ 4h
D13	**2535**	**2535**	2535	100	15						1	15	18	605	$\leq$ 4h
D14	**3280**	**3280**	3272	90	23	limit			0.3		3	32		1804	$\leq$ 4h
D15	**3990**	**3990**	3990	100	247						1	247	336	(2-loop)602	$\leq$ 4h
D16	**1060**	**1060**	1060	100	161						1	161	9	677	$\leq$ 4h
D17	**2620**	**2620**	2620	100	23						1	23	9	42	$\leq$ 4h
D18	**4165**	**4165**	4165	100	139						1	139	455	(2-loop)2951	$\leq$ 4h
D19	**2400**	**2400**	2395	94	18	limit			0.2		3	35		13090	$\leq$ 4h
D20	**1870**	**1870**	1870	100	13						1	13	27	2004	$\leq$ 4h
D21	3050	**3050**	2988	62	61	not tried					237	30560			
D22	**1865**	**1865**	1865	100	7.8						1	7.8	20	3200	$\leq$ 4h
D23	**3130**	**3130**	3120	94	249	not tried					17	1521		(2-loop) $\leq$ 4h	$\leq$ 4h
D24	**2710**	**2710**	2676	84	385	not tried					15	2428			$\leq$ 12h
D25	**1815**	**1815**	1815	100	8.2						1	8.2	11	155	$\leq$ 4h
Solved												**25**	**19**	**22**	**23**

Table A.5: Detailed results on D instances.

Ins	IUB	OPT	RLB	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BCL13	BI14	BI15
F01	**4040**	**4040**	4040	100	82						1	82	88	2170	$\leq$ 4h
F02	**3300**	**3300**	3300	100	12						1	12	19	1957	$\leq$ 4h
F03	**1665**	**1665**	1665	100	20						1	20	23	507	$\leq$ 4h
F04	**3485**	**3485**	3476	91	173	limit			0.2		13	745		9654	$\leq$ 4h
F05	**3605**	**3605**	3605	100	47						1	47	28	1023	$\leq$ 4h
F06	**1875**	**1875**	1875	100	7.2						1	7.2	12	350	$\leq$ 4h
F07	**3335**	**3335**	3335	100	8.3						1	8.3	11	226	$\leq$ 4h
F08	**3705**	**3705**	3693	92	42	limit			0.3		3	54		1927	$\leq$ 4h
F09	**4730**	**4730**	4730	100	88						1	88	137	(2-loop) 526	$\leq$ 4h
F10	**2925**	**2925**	2925	100	9.5						1	9.5	8	373	$\leq$ 4h
F11	**3835**	**3835**	3835	100	122						1	122	134	4889	$\leq$ 4h
F12	**3395**	**3395**	3390	97	75	limit			0.3		3	85		(2-loop) 2341	$\leq$ 4h
F13	**2855**	**2855**	2855	100	12						1	12	14	306	$\leq$ 4h

F14	**3330**	**3330**	3330	100	36			1	36	26		822	$\leq$ 4h
F15	**3560**	**3560**	3560	100	158			1	158	279		(2-loop) 277	$\leq$ 4h
F16	**2725**	**2725**	2725	100	30			1	30	27		543	$\leq$ 4h
F17	**2055**	**2055**	2055	100	5.4			1	5.4	6		90	$\leq$ 4h
F18	3075		3063	93	74	not tried							
F19	2525	**2525**	2504	86	87	not tried		9	6571				
F20	**2445**	**2445**	2445	100	15			1	15	42		2210	$\leq$ 4h
F21	**2930**	**2930**	2930	100	26			1	26	70		3582	$\leq$ 4h
F22	**2075**	**2075**	2075	100	5.6			1	5.6	18		141	$\leq$ 4h
F23	**3005**	**3005**	3005	100	135			1	135			$\leq$ 4h	$\leq$ 4h
F24	**3210**	**3210**	3210	100	117			1	117	146		10106	$\leq$ 4h
F25	**1390**	**1390**	1390	100	2.7			1	2.7	2		89	$\leq$ 4h
Solved									**24**	**19**		**23**	**23**

Table A.6: Detailed results on F instances.

Ins	IUB	OPT	RLB	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	LPU06	BI12	BCL13
val1A	**247**	**247**	247	100	8.7						1	8.7	98	17	5.9
val1B	**247**	**247**	247	100	5.9						1	5.9	55	4	4.8
val1C	**319**	**319**	319	100	0.8						1	0.8	8917	56	134
val2A	**298**	**298**	298	100	5.1						1	5.1	79	10	5.5
val2B	**331**	**330**	329	90	8.1	not tried					5	18	671	9	290
val2C	**528**	**528**	528	100	0.3						1	0.3	1	1	0.2
val3A	**105**	**105**	105	100	4.7						1	4.7	128	6	5.5
val3B	**111**	**111**	111	100	3.0						1	3.0	134	3	3.6
val3C	**162**	**162**	161	96	0.3	827	162	0	0.01		1	0.3	329	2	3.0
val4A	**522**	**522**	522	100	59						1	59	2475	413	80
val4B	**534**	**534**	534	100	47						1	47	1178	73	50
val4C	**550**	**550**	550	100	40						1	40	825	32	28
val4D	**650**	**650**	648	94	20	not tried					5	133		716	
val5A	**566**	**566**	566	100	31						1	31	629	76	34
val5B	**589**	**589**	588	90	40	limit			0.3		5	65		25	
val5C	**617**	**617**	613	83	25	not tried					55	306		406	
val5D	**718**	**718**	716	97	7.5	not tried					9	27		28	702

val6A	**330**	**330**	330	100	11		1	11	159	20	11
val6B	**340**	**340**	337	72	16	not tried	13	66		209	
val6C	**424**	**424**	419	85	1.5	not tried	125	54		4027	56
val7A	**382**	**382**	382	100	56		1	56	319	$\leq$ 4h	25
val7B	**386**	**386**	386	100	28		1	28	164	35	13
val7C	**437**	**437**	433	62	19	not tried	15	71	132	$\leq$ 4h	1760
val8A	**522**	**522**	522	100	24		1	24	359	38	21
val8B	**531**	**531**	531	100	18		1	18	169	18	16
val8C	**657**	**657**	654	91	6.7		19	85		25	2496
val9A	**450**	**450**	450	100	187	not tried	1	187	17723	685	213
val9B	**453**	**453**	453	100	72		1	72	4520	154	115
val9C	**459**	**459**	459	100	85		1	85	1461	75	63
val9D	**515**	**515**	512	91	21	not tried	45	289		$\leq$ 4h	
val10A	**637**	**637**	637	100	237		1	237	13337	542	215
val10B	**645**	**645**	645	100	204		1	204	13719	7555	157
val10C	**655**	**655**	655	100	181		1	181	5079	172	97
val10D	**734**	**734**	734	100	68		1	68	474	196	61
Solved								**34**	**26**	**34**	**29**

Table A.7: Detailed results on Val instances.