Gutenberg School of Management and Economics

Discussion Paper Series

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

Claudia Bode and Stefan Irnich

November 2012

Discussion paper number 1212

Johannes Gutenberg University Mainz
Gutenberg School of Management and Economics
Jakob-Welder-Weg 9
55128 Mainz
Germany
wiwi.uni-mainz.de

Contact details

Claudia Bode

Chair of Logistics Management

Johannes Gutenberg University Mainz

Jakob-Welder-Weg 9

55128 Mainz

Germany

claudia.bode@uni-mainz.de

Stefan Irnich

Chair of Logistics Management

Gutenberg School of Management and Economics

Johannes Gutenberg University Mainz

Jakob-Welder-Weg 9

55128 Mainz

Germany

irnich@uni-mainz.de

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

Claudia Bode, Stefan Irnich

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

Abstract

Recently, Bode and Irnich ('Cut-First Branch-and-Price-Second for the Capacitated Arc-Routing Problem', *Operations Research*, 2012, doi: 10.1287/opre.1120.1079) presented a cut-first branch-and-price-second algorithm for solving the capacitated arc-routing problem (CARP). The fundamental difference to other approaches from the literature for exactly solving the CARP is that the entire algorithm works directly on the typically sparse underlying graph representing the street network. This enables the use of highly efficient dynamic programming-based pricing algorithms for solving the column-generation subproblem also known as the pricing problem. The contribution of this paper is the in-depth analysis of the CARP pricing problem and its possible relaxations, including the construction of new labeling algorithms for their solution, theoretical complexity results, and comprehensive computational tests on standard benchmark problems. We will show that a systematic variation of different relaxations provides a powerful approach to solve knowingly hard instances of the CARP to proven optimality.

Key words: CARP, column generation, branch-and-price, pricing problem, relaxations

1. Introduction

The capacitated arc-routing problem (CARP) is the fundamental multiple-vehicle arc-routing problem with applications in waste collection, postal delivery, winter services and more (Dror, 2000; Corberán and Prins, 2010). Recently, Bode and Irnich (2012) presented a new exact solution approach based on an aggregated, non-symmetric formulation that was derived via a Dantzig-Wolfe decomposition of the well-known two-index formulation (Belenguer and Benavent, 1998). For its solution, violated valid inequalities as well as missing variables are generated dynamically. The corresponding cut-and-column-generation algorithm as a whole exploits the fact that the underlying CARP graph is sparse (exploitation of sparsity is an idea that was originally coined by Letchford and Oukil (2009)). Note that any approach using a transformation of the CARP into a node-routing problem results in dense graphs (Baldacci and Maniezzo, 2006; Longo *et al.*, 2006; Bartolini *et al.*, 2012). Using the one-index formulation of the CARP, some relevant valid inequalities are computed a priori in the initial cutting phase. This provides a very fast warm-start of the column-generation process. Due to direct use of a sparse network for fast pricing, the proposed column-generation algorithm often produces strong lower bounds in relatively short computation time for many instances from the literature. Integrated into branch-and-bound, the approach becomes a cut-first branch-and-price-second algorithm. The computation of integer solutions then benefits from the non-symmetric formulation and, in particular, from an effective branching scheme.

The contribution of this paper is the in-depth analysis of the CARP pricing problem and its possible relaxations, including the construction of new labeling algorithms for their solution, theoretical complexity results, and comprehensive computational tests on standard benchmark problems. Using pricing problem

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

relaxations is a standard technique in column generation (Lübbecke and Desrosiers, 2005; Desaulniers *et al.*, 2005) because pricing problems in routing applications are typically strongly $\mathcal{NP}$-hard elementary shortest-path problems with resource constraints (ESPPRC, Irnich and Desaulniers, 2005). In fact, many successful column-generation approaches play with the trade-off that different pricing problems relaxations offer (Irnich and Villeneuve, 2006; Baldacci *et al.*, 2011b). Stronger relaxations produce tighter lower bounds, but come at the cost of being harder to solve leading to longer computation times in the pricing subproblem. The branch-and-price approach in (Bode and Irnich, 2012) made use of just one relaxation producing 2-loops free tours (Benavent *et al.*, 1992). This relaxation is particularly beneficial because it is compatible and at the same time indispensable for branching on followers. Actually, branching on followers and non-followers is the only effective technique known to guarantee the integrality in branch-and-price when pricing is performed on the original sparse network.

Bode and Irnich (2012) already showed that pricing relaxations based on k-loop elimination produce better root node lower bounds. However, for these and other possible relaxations it remained unclear how integer solutions can be computed using the aforementioned branching scheme. This paper is intended to fill the gap by showing how different pricing relaxations can be made compatible with the requirements imposed by branching. We will discuss and empirically analyze the trade-offs between hardness of pricing and strength of lower bounds for various pricing relaxations. As a result, we are able to compute new best lower bounds and optimal solutions for several knowingly hard CARP instances from the benchmark sets of Eglese and Li (1992), Brandão and Eglese (2008), and Beullens *et al.* (2003).

The remainder of this paper is structured as follows: The next section defines the CARP and briefly summarizes the cut-first branch-and-price-second approach presented in (Bode and Irnich, 2012). Section 3 presents the pricing problem, and discusses well-known and also new pricing relaxations. Several acceleration techniques for solving the shortest-path subproblems via dynamic-programming labeling algorithms such as bidirectional pricing, bounding, and scaling are summarized and adapted to the new relaxations in Section 4. In Section 5, we presents comprehensive computational results and final conclusions are drawn in Section 6.

2. Cut-First Branch-and-Price-Second for the CARP

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

Complete exact methods were recently presented in (Bartolini *et al.*, 2012; Bode and Irnich, 2012). The first method consists of computing a cascade of non-decreasing lower bounds, enumerating all routes with reduced cost smaller than the integrality gap of upper bound minus the best lower bound, and finally solving the master program with a (general purpose) mixed integer-programming solver. Note that Bartolini *et al.* (2012) make intensive use of a transformation of the CARP into a generalized vehicle-routing problem (GVRP) so that route generation is performed on a dense graph. In contrast, the sparsity of the CARP network is heavily exploited by Bode and Irnich (2012), where in the first phase a cutting-plane algorithm is applied to initialize the column-generation master program and in the second phase the branch-and-price algorithm is executed. This general approach will be explained in detail in Sections 2.2 and 2.3.

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

2.1. Notation and Definition of the CARP

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

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

Some authors define the CARP for an unlimited fleet of vehicles (Belenguer and Benavent, 2003; Longo *et al.*, 2006; Bartolini *et al.*, 2012), others fix the number of vehicles (Bode and Irnich, 2012; Belenguer and Benavent, 1998). Here, the fleet size is also fixed to the minimum number K of required vehicles (computed by solving a bin-packing problem) and we assume that each vehicle of the *homogeneous fleet* has capacity C and is stationed at the depot d.

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

2.2. Cutting-Plane Generation: First Phase

The first phase of the algorithm presented in (Bode and Irnich, 2012) consists of the generation of a relevant set of valid inequalities that are later added to the column-generation formulation. Solving the following one-index formulation with a cutting-plane procedure, the added inequalities are those that are binding at the end.

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

$$\min \quad c^\top y \tag{1}$$

$$\text{s.t.} \quad y(\delta(S)) \geq 2K(S) - |\delta_R(S)| \quad \text{for all } \emptyset \neq S \subseteq V \setminus \{d\} \tag{2}$$

$$y(\delta(S)) \geq 1 \quad \text{for all } \emptyset \neq S \subseteq V,\ |\delta_R(S)| \text{ odd} \tag{3}$$

$$y \in \mathbb{Z}_+^{|E|} \tag{4}$$

The objective (1) minimizes the costs of all deadheadings (note that service costs are constant and therefore irrelevant for routing decisions). The capacity inequalities (2) require that there are at least $2K(S)$ traversals (services and deadheadings) over the cutset $\delta(S)$. Herein, $K(S)$ is the minimum number of vehicles needed to service the edges $E_R(S) \cup \delta_R(S)$. The number $K(S)$ can be approximated by $\lceil q(E_R(S) \cup \delta_R(S))/C \rceil$ and computed exactly by solving a bin-packing problem. Furthermore, the odd-cut inequalities (3) ensure for each subset S with an odd number of required edges in the cut $\delta(S)$ that at least one deadheading is performed. Belenguer and Benavent (2003) introduced disjoint-path inequalities as another class of valid cuts for the CARP. The idea is to consider not only the demand of $E_R(S) \cup \delta_R(S)$ but also the demand on a path from the depot to the set S. The general form of all valid inequalities (including disjoint-path inequalities) can be written as $\sum_{e \in E} d_{es} y_e \geq r_s$ for $s \in \mathcal{S}$ where $\mathcal{S}$ is the set of all inequalities and d_{es} the coefficient of edge e in a particular cut indexed by s.

2.3. Branch-and-Price: Second Phase

In the second phase of the algorithm presented in (Bode and Irnich, 2012), a restricted master program is iteratively reoptimized and variables with negative reduced costs are generated at each iteration. To obtain integer solutions a branching scheme is applied.

2.3.1. Master Program

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

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

$$\min \quad \sum_{r \in \Omega} c_r \lambda_r + \sum_{e \in E} (2c_e) z_e \tag{5}$$

$$\text{s.t.} \quad \sum_{r \in \Omega} \bar{x}_{er} \lambda_r = 1 \quad \text{for all } e \in E_R \tag{6}$$

$$\sum_{r \in \Omega} d_{sr} \lambda_r + \sum_{e \in E} (2d_{es}) z_e \geq r_s \quad \text{for all } s \in \mathcal{S} \tag{7}$$

$$\mathbf{1}^\top \lambda = K \tag{8}$$

$$\lambda \geq \mathbf{0}, z \geq \mathbf{0} \tag{9}$$

The objective (5) consists of minimizing the costs of the routes plus the costs of deadheading along simple cycles. Each required edge must be covered by one route (6). Both route variables λ_r and cycle variables z_e are impacted by the additional cuts from phase one. For a specific cut $s \in \mathcal{S}$, the route $r \in \Omega$ has the coefficient $d_{sr} = \sum_{e \in E} d_{es} \bar{y}_{er}$, and the respective coefficient of the cycle variable z_e is $2d_{es}$. Thus, the general form of cuts from the one-index formulation can be transformed into the reformulated cuts (7). Since the number of vehicles is fixed, exactly K routes are used (8) and all variables are non-negative (9).

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

$$y_e = \sum_{r \in \Omega} \bar{y}_{er} \lambda_r \in \mathbb{Z}_+. \tag{10}$$

The first condition is sufficient, but not necessary, because integer solution can sometimes be reconstructed from fractional λ variables (Bode and Irnich, 2012). The latter conditions (10) are necessary, but not sufficient, see Section 2.3.3 on branching.

2.3.2. Pricing Problem

Because the restricted master program (RMP) is initialized with a proper subset of route variables λ_r, missing variables with negative reduced costs must be priced out. In fact, the task of the pricing problem is the generation of those variables. Let $\pi = (\pi_e)_{e \in E_R}$ be the vector of dual prices for covering constraints (6), $\beta = (\beta_s)$ the vector of dual prices for active valid inequalities (7), and μ the dual price to the generalized convexity constraint (8). Reduced costs for service and deadheading are defined as follows:

$$\tilde{c}_e^{serv} = c_e^{serv} - \pi_e \text{ for all } e \in E_R \qquad \text{and} \qquad \tilde{c}_e = c_e - \sum_{s \in \mathcal{S}} d_{es}\beta_s \text{ for all } e \in E. \tag{11}$$

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

$$\begin{aligned}
z_{PP}(\pi,\beta,\mu) \quad &= \quad \min \tilde{c}^{serv,\top} x + \tilde{c}^\top y - \mu && (12)\\
\text{s.t.} \quad & \quad x(\delta_R(S)) + y(\delta(S)) \geq 2x_f \quad \text{for all } S \subseteq V \setminus \{d\},\, f \in E_R(S) && (13)\\
& \quad x(\delta_R(i)) + y(\delta(i)) = 2p_i \quad \text{for all } i \in V && (14)\\
& \quad q^\top x \leq C && (15)\\
& \quad p \in \mathbb{Z}_+^{|V|}, x \in \{0,1\}^{|E_R|}, y \in \mathbb{Z}_+^{|E|} && (16)
\end{aligned}$$

The objective (12) is the minimization of the reduced costs. Constraints (13) ensure connectivity of all required edges serviced. An even node degree is guaranteed by (14) using auxiliary integer variables p_i, one for each node $i \in V$. Constraint (15) is the capacity constraint.

Obviously, whenever deadheading gives no profit, i.e., $\tilde{c}_e \geq 0$ for all $e \in E$, it is not efficient to have cycles consisting only of deadheading. However, the two-index formulation, from which Bode and Irnich (2012) derived the master program and pricing problem, allows deadheading cycles denoted as extended k-routes in (Belenguer and Benavent, 1998). These extended k-routes correspond to extreme rays of the polyhedron formed by (13)–(16). The variables z_e in the master program (5)–(9) model cycles $(e, e) = (i, j, i)$ for each edge $e = \{i, j\} \in E$. Additional variables in this master problem (the primal problem) correspond to inequalities in the associated dual problem. Therefore, the variables z_e give dual inequalities of the form $\sum_{s \in \mathcal{S}} d_{es}\beta_s \leq c_e$ for all $e \in E$. These dual inequalities result in a stabilization of the dual variables β_s (Ben Amor *et al.*, 2006). Moreover, the algorithmic advantage for pricing is the guarantee that the reduced costs $\tilde{c}_e$ of deadheadings over all edges are non-negative. The algorithms presented in Section 3 substantially rely on that property.

Note that optimal CARP tours require only the knowledge of the Eulerian subgraphs (V', E') and the partition of E' into served edges $E_s = \{e \in E : x_e = 1\}$ and deadheaded edges E_d. The pricing problem is in fact not a routing problem, since the ordering of serviced and deadheading edges is irrelevant. However, the only viable approach known to us for solving the pricing problem is to compute paths. Hence, we solve a routing problem and herewith determine an ordering of serviced and deadheading edges. We will see that this ordering is also crucial for the branching scheme presented in the next section. As pointed out earlier by Bartolini *et al.* (2012), a feasible CARP tour can then be represented by several possibilities of traversing the corresponding Eulerian subgraph.

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

2.3.3. Branching

In order to obtain integer solutions, a hierarchical branching scheme was devised. It consists of three levels of branching decisions: (1) branching on node degrees, whenever a node with a non-even degree exists, (2) branching on edges with fractional edge flow, (3) branching on follower information, whenever the information if two edges are serviced consecutive is fractional. Note that the third branching decision is applicable, since the pricing problem is solved as a routing problem, where an ordering of serviced edges is determined. This decision guarantees integer route variables and can be handled by modifying the underlying pricing network. Bode and Irnich (2012) showed that follower constraints in the branching part can be handled in the pricing problem by adding edges that represent certain paths. On the other hand, non-follower constraints are handled by associating the same task to the corresponding edges. Combinations of several follower and non-follower constraints are more intricate to implement, but follow the same idea.

3. Pricing Problem Relaxations

Letchford and Oukil (2009) analyzed two mixed integer linear programming (MIP) models for solving the elementary pricing problem (12)–(16). When solved with the general purpose MIP solver CPLEX, the resulting computation times were prohibitively long. In principle, the pricing problem (12)–(16) is solvable as an ESPPRC with tasks on service edges using known labeling techniques from the literature (see Irnich and Desaulniers, 2005). However, as paths can become rather long, ESPPRC labeling still suffers from extensive computation times.

Since the ESPPRC is strongly $\mathcal{NP}$-hard, different relaxations were considered in the literature. Letchford and Oukil (2009) proved that the non-elementary relaxation of the pricing problem can be solved in pseudo-polynomial time $\mathcal{O}\left(C(|E| + |V| \log |V|)\right)$. Their labeling algorithm comprises two building blocks invoked alternately, one is similar to standard labeling approaches for extending labels along service edges and the other is a Dijkstra-like algorithm for extensions along deadheading edges. The Dijkstra steps rely on the property that deadheading edges have non-negative reduced costs (this can be assured, see Section 2.3.2).

A stronger formulation than the non-elementary SPPRC results from the 2-loop-free (=task-1-cycle-free) pricing relaxation already known for the CARP from the work of Benavent *et al.* (1992). Note that task-2-loop-free pricing in the arc routing context allows paths containing task sequences of the form (a, b, a), whereas (a, a) is forbidden. However, in the node routing context node-2-cycle-free pricing allows subpaths (i, j, k, i) and forbids (i, j, i). Both strategies have in common requiring two paths to dominate a third one (see Section 3.4 for further details) so that one must record, for every state, a best and a second best label having a different last task. To distinguish between arc and node routing, we will always refer to *loop* freeness in the arc-routing context. Comprehensive computational results with 2-loop-free tours were already presented in (Bode and Irnich, 2012).

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

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

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

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

In essence, a shortest-path problem where paths are elementary w.r.t. $\mathcal{T}^E$ and task-2-loop-free w.r.t. $\mathcal{T}^B$ must be solved. In the following, we will skip the 'task-' prefix. Consequently, 2-loop-free tours are indispensable, since the only viable branching scheme (known to us) is based on follower and non-follower constraints resulting in edges having identical tasks.

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

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

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

A feasible path P ending at $i = i(P)$ can be extended along an edge either deadheaded or serviced. Any deadheading extension along an edge $e = \{i, j\} \in \delta(i)$ with associated reduced cost $\tilde{c}_e$ is feasible. The resulting new path P' has the following attributes:

$$\begin{aligned}
i(P') &= j \\
\tilde{c}(P') &= \tilde{c}(P) + \tilde{c}_e \\
q(P') &= q(P) \\
\mathcal{T}^E(P') &= \mathcal{T}^E(P) \\
\mathcal{T}^B(P') &= \mathcal{T}^B(P)
\end{aligned} \tag{17}$$

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

$$\begin{aligned}
i(P') &= j \\
\tilde{c}(P') &= \tilde{c}(P) + \tilde{c}_e^{serv} \\
q(P') &= q(P) + q_e \\
\mathcal{T}^E(P') &= (\mathcal{T}^E(P), \mathcal{T}^E(i, j)) \\
\mathcal{T}^B(P') &= \mathcal{T}^B(i, j)
\end{aligned} \tag{18}$$

In the pure non-elementary case considered by Letchford and Oukil (2009), the attributes $\mathcal{T}^E(P)$ and $\mathcal{T}^B(P)$ are completely ignored. Then, a path P dominates another path Q if $i(P) = i(Q)$, $\tilde{c}(P) \leq \tilde{c}(Q)$, and $q(P) \leq q(Q)$ holds. The entire labeling procedure is summarized in Algorithm 1.

Some remarks about Algorithm 1 seem appropriate here:

1. In the non-elementary case, dominance is trivial. The set $\{P \in \mathcal{P}_q : i(P) = i, q(P) = q\}$ for a given combination of i and q contains not more than a single path (sometimes no path). Whenever a new path P' is created with load q, it replaces the existing one, say Q, only if it is cheaper, i.e., $\tilde{c}(P') < \tilde{c}(Q)$. If paths are stored in arrays (index by node $i(P)$ and load $q(P)$) this dominance step needs just constant time $\mathcal{O}(1)$.
2. The use of a Fibonacci heap data structure (see Ahuja *et al.*, 1993) guarantees the worst-case complexity of $\mathcal{O}(|E| + |V| \log |V|)$ of the Dijkstra-like extensions.
3. The final filtering step is necessary, since the algorithm would otherwise output some paths that are not Pareto-optimal. Note that the dominance procedure among all paths ending at the node d requires $\mathcal{O}(C)$ time only because paths P with $i(P) = d$ are already sorted by $q(P)$ (by using the indexing).

Algorithm 1: Efficient Pricing Algorithm $\mathcal{O}\left(C \cdot (|E| + |V| \log |V|)\right)$

```
for q = 0, 1, 2, ..., C do
    // Dijkstra-like extensions
    Let P_q be the (sorted) set of paths P with q(P) = q
    // Keep P_q always sorted w.r.t. c̃(P) using a Fibonacci heap
    for P ∈ P_q do
        Extend P along deadheading edges e = {i, j} ∈ δ(i) where i = i(P) using (17)
        Add the new path P' to P_q
        Apply dominance algorithm among Q ∈ P_q with i(Q) = i(P')
    // Service extensions
    Let P_q be the (unsorted) set of paths P with q(P) = q
    for P ∈ P_q do
        Extend P along service edges e = {i, j} ∈ δ_R(i) where i = i(P) using (18)
        if new path P' is feasible then
            // path P' has load q(P') = q + q_e > q
            Add the new path P' to P_q(P')
            Apply dominance algorithm among Q ∈ P_q(P') with i(Q) = i(P')
// Filtering step
Apply dominance algorithm at destination node d among all paths P ending at d = i(P)
```

3.1. 2-Loop-free Paths

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

3.2. ng-Route Relaxation

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

The subset $\mathcal{T}^E_{NG}(P) \subseteq \mathcal{T}^E$ is defined recursively with the extension of a path P ending at node $i = i(P)$ along an edge $e = \{i, j\} \in \delta(i)$. Any deadheading extension is allowed, and the new task set for the resulting path $P' = (P, e, j)$ is

$$\mathcal{T}^E_{NG}(P') = \mathcal{T}^E_{NG}(P) \cap N_j.$$

In contrast, the extension along the service edge is considered feasible w.r.t. $(N_i)_{i \in V}$ if and only if

$$\mathcal{T}^E_{NG}(P) \cap \{\mathcal{T}^E(i, j)\} = \emptyset,$$

and, in this case, the new path P' has the task subset

$$\mathcal{T}^E_{NG}(P') = (\mathcal{T}^E_{NG}(P) \cup \{\mathcal{T}^E(i,j)\}) \cap N_j,$$

where $\{\mathcal{T}^E(i,j)\}$ denotes the *set* of tasks in the service sequence (i,j).

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

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

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

3.3. Partial Elementary

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

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

3.4. k-Loop-free Paths

It is known that solving an SPPRC with k-loop elimination is a good compromise between solving ESPPRC and SPPRC. Note that a path is k-loop-free if it does not contain a task loop of length k or smaller, e.g., for $k = 3$ no 3-loops and no 2-loops. A general labeling algorithm for k-loop-free SPPRC was presented by Irnich and Villeneuve (2006). At the time of its invention, it proved to be highly successful for computing optimal solutions to some knowingly hard VRPTW instances.

In (Bode and Irnich, 2012), computational results for solving the linear relaxation of the column-generation master program with k-loop-free pricing were presented. Due to the incompatibility of non-follower branching with simple k-loop elimination for $k \geq 3$, however, the algorithm by Bode and Irnich (2012) did not provide results for branch-and-price.

3.5. $(k, 2)$-Loop-free Paths

This section contains new theoretical results for labeling procedures that simultaneously consider two sets of tasks for which loop freeness must be guaranteed. In our CARP application, paths are desired to be k-loop-free w.r.t. tasks $\mathcal{T}^E$, where we would like $k > 2$ to be as large as possible (of course there is the trade-off between strength of the relaxation and effort for pricing), and need to be exactly 2-loop-free w.r.t. the tasks $\mathcal{T}^B$. Generalizing, we will derive results for a combined (k_1, k_2)-loop elimination for the tasks sets $\mathcal{T}^1$ and $\mathcal{T}^2$. For simplicity, we abbreviate paths feasible w.r.t. both tasks sets $\mathcal{T}^1$ and $\mathcal{T}^2$ as (k_1, k_2)-loop-free paths.

It is rather simple to define attribute updates and extension rules for (k_1, k_2)-loop elimination. The crucial part for an effective labeling algorithm is however the definition of a dominance relation. Straightforward approaches define dominance only between paths taking the last k_1-1 tasks of $\mathcal{T}^1$ and k_2-1 tasks of $\mathcal{T}^2$ into account. This is rather easy, but turns out to be ineffective due a possible number of $\mathcal{O}\left(|\mathcal{T}^1|^{k_1-1}\cdot|\mathcal{T}^2|^{k_2-1}\right)$ labels at the same node and with identical load; see also (Irnich and Villeneuve, 2006) where this point is discussed for node-k-cycle elimination. Therefore, the decisive point is the development of effective dominance rules guaranteeing a small number of labels.

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

The *dominance principle* says that labels $L_1,\dots,L_s$ $(s\geq 1)$ representing paths $P_1,\dots,P_s$ dominate a label L representing path P if

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

As a consequence, the label L does not need to be considered in a labeling algorithm because it can never produce a better feasible extension to the destination node than possible with at least one extension of the labels $L_1,\dots,L_s$. It is however crucial that the labels $L_1,\dots,L_s$ are kept.

The second condition (2.) is typically replaced by a (sufficient) condition that is easier to check, involving resource consumptions and task loops. In fact, all paths $P_1,\dots,P_s$ must have not larger loads and reduced costs than P, i.e.,

$$q(P_1),\dots,q(P_s)\leq q(P) \qquad \text{and} \qquad \tilde{c}(P_1),\dots,\tilde{c}(P_s)\leq\tilde{c}(P), \tag{19}$$

while feasibility regarding tasks loops is not checked via resources.

The fundamental idea for (k_1,k_2)-loop elimination is to efficiently encode the *set of possible extensions* of a path. For this purpose, let $\mathcal{E}(P)$ denote the set of loop-free extensions of the path P. $\mathcal{E}(P)$ solely considers task loops and not resource consumptions. The second condition above is fulfilled for $P_1,\dots,P_s$ and P if (19) and

$$\bigcup_{i=1}^{s}\mathcal{E}(P_i)\supseteq\mathcal{E}(P) \tag{20}$$

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

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

$$(\mathcal{T}^1(P),\mathcal{T}^1(Q))=(\dots,t^1_{k_1-1},\dots,t^1_2,t^1_1,s^1_1,s^1_2,\dots,s^1_{k_1-1},\dots) \text{ with } t^1_p\neq s^1_q \text{ for all } p+q\leq k_1$$

and

$$(\mathcal{T}^2(P),\mathcal{T}^2(Q))=(\dots,t^2_{k_2-1},\dots,t^2_2,t^2_1,s^2_1,s^2_2,\dots,s^2_{k_2-1},\dots) \text{ with } t^2_p\neq s^2_q \text{ for all } p+q\leq k_2.$$

The relevant entries of $\mathcal{T}^1(Q)$ and $\mathcal{T}^2(Q)$ are the first k_1-1 and k_2-1 entries, and we denote these by $\mathcal{T}^1_{k_1}(Q)$ and $\mathcal{T}^2_{k_2}(Q)$, respectively. Both sequences $\mathcal{T}^1_{k_1}(Q)$ and $\mathcal{T}^2_{k_2}(Q)$ always contain exactly k_1-1 and k_2-1 elements, respectively, where missing tasks are represented by a '$\cdot$'.

We are able to express the above condition as

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

and

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

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

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

Example 1. *For $(4, 2)$-loop elimination in the CARP context, i.e., $k_1 = 4$, $k_2 = 2$, $\mathcal{T}^1_{k_1} = \mathcal{T}^E_4$ and $\mathcal{T}^2_{k_2} = \mathcal{T}^B_2$, let path P have $\mathcal{T}^E_4(P) = (a, b, c)$ and $\mathcal{T}^B_2(P) = (\alpha)$. It means that the last three required edges serviced were the edges a, b, and c. In addition, we are in a branch of the branch-and-price search tree where a non-follower constrained is active, e.g., say for the edges c and d, imposing that they have the new identical task α assigned in order to prevent c and d being serviced consecutively.*

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

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

Equivalently, the self-hole set $H(P)$ of P is

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

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

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

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

Now we consider a dominance situation where (19) and (20) are fulfilled for dominating paths $P_1, \dots, P_s$ and a dominated path P. By de Morgan's law, we get

$$\bigcup_{i=1}^{p} \mathcal{E}(P_i) \supseteq \mathcal{E}(P) \quad \Longleftrightarrow \quad \bigcap_{i=1}^{p} H(P_i) \subseteq H(P) \tag{21}$$

so that the condition (20) for loop-free extensions can be equivalently stated with the help of self-hole sets. The point is now that any intersection of the self-hole sets, resulting on the right hand side, can be calculated and represented as a union of set forms again.

Example 2. *Continuing Example 1, let P' be another path with $\mathcal{T}^E_4(P') = (\cdot, a, d)$ (just two edges serviced along P') and $\mathcal{T}^B_2(P') = (\beta)$. The self-hole set of P' is*

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

Then, the intersection is of self-hole sets is

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

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

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

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

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

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

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

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

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

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

giving rise to

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

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

Upper Bound on the Number of Paths with Identical Resource Vectors. The paragraph above presented results on the number of set forms in an intersection of an arbitrary number of paths. The question considered in this paragraph is about the maximum number of paths P with identical resource vectors (for the CARP, with identical load $q(P)$, the costs $\tilde{c}(P)$ may differ). Let a collection of s paths $P_1, \ldots, P_s$ with identical resource vectors ending at a node $i = i(P_1) = \cdots = i(P_s)$ be given. The corresponding labels can be sorted in a unique way using the IDs of the labels so that the following ordering of the paths is given:

$$P_1 \prec_{dom} P_2 \prec_{dom} \ldots \prec_{dom} P_s,$$

meaning that, e.g., P_s is dominated by all other paths $P_1, P_2, \ldots, P_{s-1}$. It follows for the intersections of the self-hole sets of the dominating labels (P_1 dominates P_2, P_1 and P_2 dominate P_3 etc.) that

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

holds. Irnich and Villeneuve (2006) have shown that a path P_t can be discarded if $I_t = I_{t-1}$ holds. Therefore, the *maximum length of a properly decreasing chain of intersections of self-hole sets* is a bound on the maximum number of labels to consider with identical resource vector.

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

For the special case of a combined $(k, 2)$-loop elimination, i.e., for $k_1 = k$ and $k_2 = 2$, the bound is $\alpha(k, 2) = (k+1) \cdot (k-1)!^2 \cdot k = (k-1)! \cdot (k+1)!$. In particular, we get the bounds $\alpha(3, 2) = 2 \cdot 24 = 48$ and $\alpha(4, 2) = 6 \cdot 120 = 720$ for $k = 3$ and 4, respectively.

3.6. Scaling

Scaling of instances is a technique often used in approximation algorithms (Vazirani, 2001). Depending on its concrete implementation, scaling can either provide relaxations or restrictions to a problem. Therefore, lower and upper bounds can result.

In the vehicle routing context, scaling of the demand q_i was e.g. considered by Fukasawa *et al.* (2006). They use it as a heuristic, i.e., a restriction of the pricing problem. For a given scaling factor f, both the demand and the capacity are modified via $q'_e = \lceil \frac{q_e}{f} \rceil$ and $C' = \lceil \frac{C}{f} \rceil$. Obviously, this scaling by factor f has the potential to speed up a labeling algorithm by a factor up to f because the main loop in Algorithm 1 has by the factor f less iterations.

On the other hand, scaling with $q'_e = \lfloor \frac{q_e}{f} \rfloor$ and $C' = \lfloor \frac{C}{f} \rfloor$ constitutes a pricing relaxation. The expected acceleration when solving the scaled instead of the original instance is also by the factor f.

3.7. Hierarchy of Pricing Relaxations

All presented pricing relaxations form a hierarchy of relaxations beginning with non-elementary pricing as the weakest relaxation and ending with elementary pricing combined with 2-loop elimination as the strongest. This hierarchy is shown in Figure 1. An arc connecting two relaxations indicates that the tail is a stronger formulation than the head. For example, the relaxation with $(4, 2)$-loop-free routes is stronger than with 4-loop-free routes and $(3, 2)$-loop-free routes. The relaxations on the right hand side are parameterized with one or several neighborhoods N and $(N_i)_{i \in V}$ so that these boxes represent families of relaxations. Inside each family, relaxations become stronger the larger the subsets N and N_i are (comparable only in case of subset inclusions). Moreover, the ng-route relaxation is stronger than the relaxation with partial elementarity whenever $N_i \supseteq N$ holds for all nodes $i \in V$.

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

4. Acceleration Techniques

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

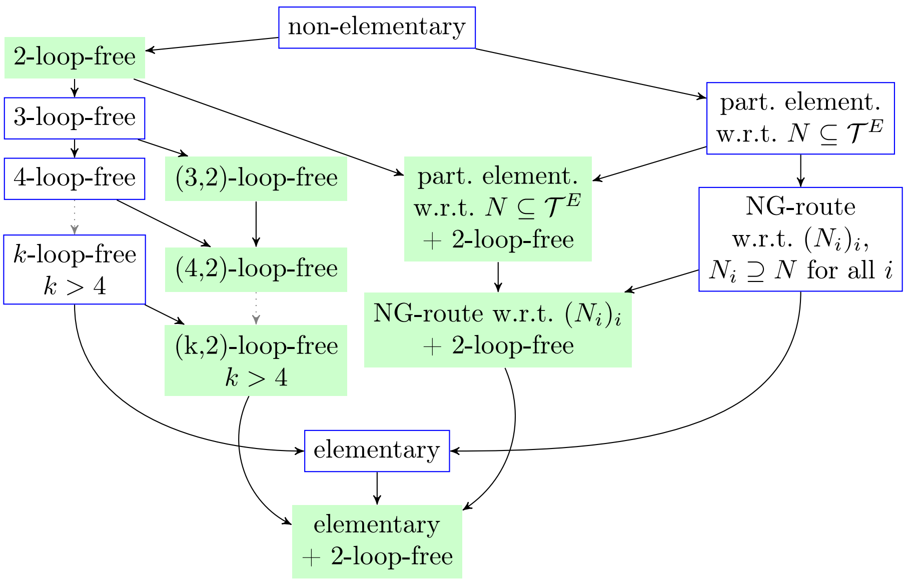


Figure 1: Hierarchy of Pricing Relaxations

4.1. Pricing Heuristics

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

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

4.2. Bi-Directional Pricing

As pointed out by Righini and Salani (2006), when solving elementary pricing problems with DP, the number of generated states rapidly increases with the stage and the problem size. They proposed a bi-directional labeling algorithm to partially countervail this effect. It outperforms standard mono-directional pricing algorithms as proven for many node-routing applications. This technique can also be applied for all pricing relaxations discussed in Section 3.

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

2-*Loop-free Paths.* A special case occurs when 2-loop-free paths are generated. If the `join` procedure is implemented in a straightforward fashion, its complexity is $\mathcal{O}\left(|V|C^2\right)$ because up to $4(C+1)^2$ pairs of paths need to be compared at each node. For the 2-loop-free relaxation, where the number of labels at a node does not grow but is constant for increasing values of the load q, preliminary tests have shown that the `join` procedure dominates the run time. Therefore, a more efficient `join` is needed.

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

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

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

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

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

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

4.3. Bounding

Bounding is intended to reduce the number of states to expand in a DP approach. It has become a key technique for solving the TSP with time windows (Mingozzi *et al.*, 1997; Baldacci *et al.*, 2011c) and variants of the VRP (Baldacci *et al.*, 2009).

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

There is a trade-off between the quality of that lower bound and the time needed for its computation. In general, any relaxation of an (E)SPPRC and backward paths generated as solutions to the all-to-destination problem provide feasible lower bounds (for details see, e.g., Baldacci *et al.*, 2011c). Note first that in the CARP the network is fully symmetric so that forward and backward labeling is identical. Any relaxation solved with mono-directional labeling on the original network so provides lower bounds. The hierarchy of relaxations depicted in Figure 1 offers numerous possibilities for pricing problem relaxations and proper relaxations of these that in combination allow bounding.

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

5. Computational Results

This section reports computational results of the various pricing relaxations tested when solving the respective linear relaxation and integer formulations of the CARP. The first benchmark set `egl` was introduced by Eglese and Li (1992) and can be downloaded from `http://www.uv.es/~belengue/carp/`. This set consists of 24 instances based on the road network of Cumbria. Group `e` consists of instances with 77 nodes and 98 edges, whereas group `s` is larger and has instances with 140 nodes and 190 edges. Each group is further split into four subsets $m \in \{1, \ldots, 4\}$, where the number of required edges increases with m. On the lowest level, each subgroup differs in the vehicle capacity, where three different sizes are assumed, indicated by $n \in \{\texttt{a},\texttt{b},\texttt{c}\}$. Within each subgroup, the instances `a` have highest capacity tending to result in less but longer routes, and instances `c` have lowest capacity resulting in more but shorter routes. Overall, instance names are coded as follows: `egl`-lm-n with $l \in \{\texttt{e},\texttt{s}\}$, $m \in \{1, \ldots, 4\}$, and $n \in \{\texttt{a},\texttt{b},\texttt{c}\}$.

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

5.1. Computational Setup

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

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

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

There exist several methods of determining the concrete sets N_i. Desaulniers *et al.* (2008) proposed an algorithm for partially elementary, i.e., $N_i = N$ for all $i \in V$, in which iteratively the linear relaxation of the RMP is solved. As long as the neighborhood size $|N|$ is smaller than a predefined maximal size n_{max} and there exists a task cycle in the solution, this task is added to the neighborhood N. Tasks with a large flow on cycles are chosen with priority. On the other hand, Baldacci *et al.* (2011b) use individual neighborhoods N_i for every node $i \in V$. The sets N_i are pre-computed by adding a customer j to N_i if it is among the n_{ng} nearest nodes to node i. We combine these two ideas because we dynamically generate individual neighborhoods N_i (a similar idea was presented by R. Roberti in the presentation (Baldacci *et al.*, 2011a)). The procedure is summarized in Algorithm 2.

Algorithm 2: Generation of Neighborhoods $(N_i)_{i \in V}$

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

Note that when adding new tasks to a neighborhood N_i, the resulting relaxation becomes more restrictive so that a formerly feasible route r can become infeasible. Those routes that become infeasible have to be removed from the RMP at the beginning of every main loop of Algorithm 2. Thus, the RMP first gets smaller, while it increases again with every newly generated route.

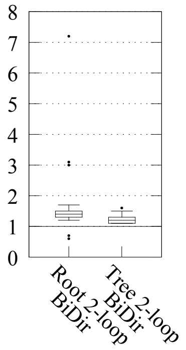
Finally, bidirectional labeling can be applied in every pricing algorithm. In the following, we indicate bidirectional labeling with the term 'BiDir'.

5.2. Impact of Acceleration Techniques

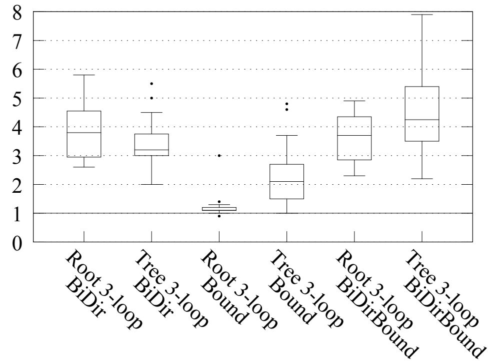
We start with analyzing the impact of the acceleration techniques presented in Section 4. In order to measure the improvement of bounding and bidirectional pricing for different pricing relaxations, both the root node and the full branch-and-bound tree were solved with no, one, or both techniques active. Computations were performed for all 24 `egl` instances and the different relaxations. The improvement is then calculated as the ratio of the time for pricing without acceleration and the time with one or both techniques active, respectively, for each instance. For abbreviation, we refer to the these numbers as *acceleration factors*. For not biasing the acceleration factors, we turned off all heuristic pricing procedures. Figures 2 and 3 show the resulting box-and-whisker diagrams (McGill *et al.*, 1978).

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

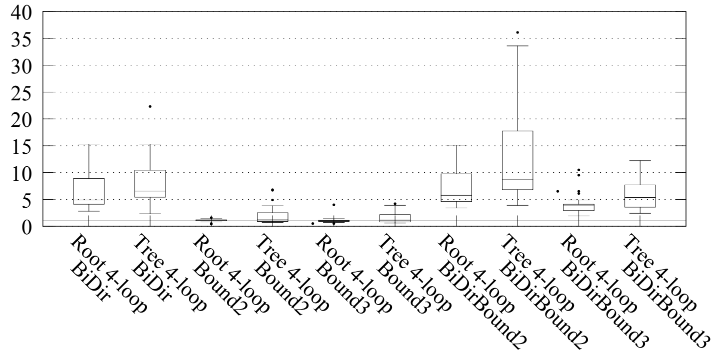
This median increases to 3.8 and 5.1 for 3-loop and 4-loop, respectively (see Figures 2b and 2c). For these relaxations, bidirectional pricing has always an impact greater than one, nevertheless the data scatters more. For example, for the instance `e4-a` solving the root node with bidirectional pricing is about 15 times faster than with the basic 4-loop algorithm, and for the instance `s4-c` just 2.8 times faster. For indicating



(a) 2-loop



(b) 3-loop



(c) 4-loop

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

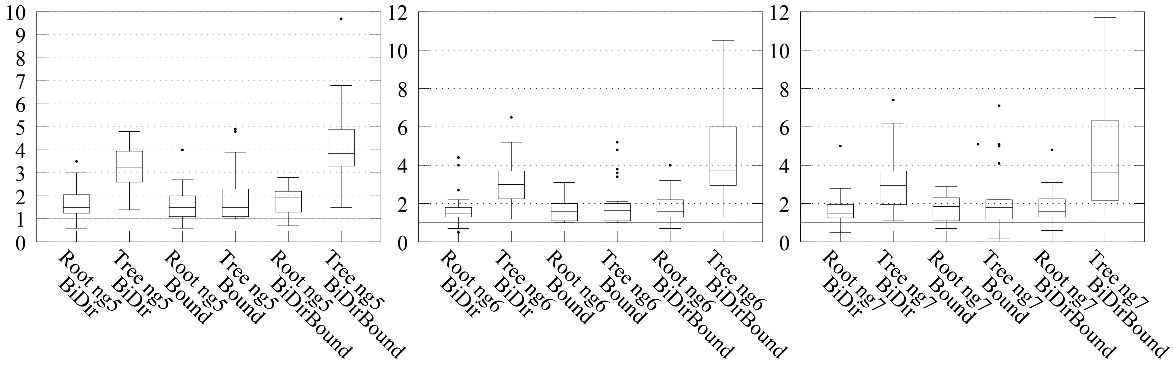
the spread of the data, the end of the whiskers show data that lying within the 1.5 interquartile range. Any other data is outliers and they are represented by small dots.

Comparing the results over the full branch-and-bound tree solely using bidirectional pricing, there is an improvement compared to the root node only for 4-loop pricing. However, combined with bounding the positive impact of using acceleration techniques is strengthened. Sometimes a speed up factor of 36 can be reached (instance `s2-c` in $4b2$-loop pricing).

The impact of using bounding alone is very small, in particular for solving the linear relaxation ('Root'). The median within 3-loop pricing is only slightly above 1.0 and the lower whisker is ending at 1.0. There, bounding has always a small but non-negative impact compared to 4-loop pricing. The median for bounding with 2-loop and 3-loop bounding is 1.0 and 0.9, respectively. Hence, bounding alone often results in longer computation times. Considering the whole branch-and-bound tree ('Tree'), the acceleration factors are slightly higher.

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

The impact of bidirectional pricing and bounding is, at the root node, very similar for all tested ng-route relaxations (see Figures 3a–3c). The median of all acceleration techniques is between approximately 1.5 and 2.0, and the dispersion of the data is not as high as for the k-loop relaxations. However, except for solely bounding within $ng6$, there are instances where solving the root node takes longer than without any acceleration techniques. Similar to k-loop, considering the full branch-and-bound tree, the impact of



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

Figure 3: Impact of Bidirectional Pricing and/or Bounding for the ng-route Relaxations: Box-and-Whisker Diagrams of the Acceleration Factors

bounding and/or bidirectional pricing is at least as good as at the root node, but often better. The only exception is bounding within the $ng7$-route relaxation: The median is approximately the same comparing the root node and the full tree, but there are instances (e.g., `e1-b` and `e2-b`) where solving the pricing problem is up to five times slower than the basic $ng7$-route algorithm. In general, combining all presented acceleration techniques for solving the branch-and-price part gives the best results. Therefore, all following computational results are presented for combined bidirectional pricing with bounding.

5.3. Linear Relaxation Results

The focus of the following analysis is on lower bounds obtained with the linear relaxations (at the root node). A comprehensive study for the `egl` instances and relaxations with k-loop elimination was already presented in (Bode and Irnich, 2012). However, no acceleration techniques and no ng-route relaxations were considered. Therefore, we will now present lower bounds and computation times for k-loop elimination and ng-route relaxations with the presented acceleration techniques activated. Table 1 presents aggregated results for the `egl` instances and Table 2 for the `bmcv` instances.

Table 1: Aggregated Linear Relaxation Results for `egl` Instances

	2-loop	3-loop	4b2-loop	4b3-loop	$ng5$	$ng6$	$ng7$
Minimum gap (%)	0.07	0.05	0.05	0.05	0.00	0.00	0.00
Average gap (%)	0.84	0.74	0.68	0.68	0.61	0.59	0.58
Maximum gap (%)	1.60	1.30	1.29	1.29	1.24	1.23	1.23
Minimum time (s)	9	22	21	26	63	67	65
Average time (s)	90	233	511	615	1,646	2,220	2,601
Maximum time (s)	294	837	4,151	3,660	10,507	10,016	14,306

Table 2: Aggregated Linear Relaxation Results for `bmcv` Instances

	2-loop	3-loop	4b2-loop	4b3-loop	$ng5$	$ng6$	$ng7$
Minimum gap (%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Average gap (%)	0.55	0.54	0.52	0.52	0.48	0.48	0.48
Maximum gap (%)	2.79	2.69	2.69	2.69	2.39	2.37	2.37
Minimum time (s)	1	3	2	2	2	2	2
Average time (s)	20	317	426	274	1,668	2,055	2,277
Maximum time (s)	194	22,760	14,914	12,902	14,373	14,288	14,253

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

Due to the integration of 2-loop-free pricing in *ng*-route relaxations (see Section 3), the lower bounds obtained with any *ng*-route relaxation are always at least as good as the lower bounds with the 2-loop-free relaxation. Therefore, a stronger relaxation results in better lower bounds, i.e., smaller gaps in the best, average, and worst case. Some substantial improvements were observed, e.g., 69 units for the instances `egl-e2-c` and `egl-s1-c`. For all relaxations, the minimum gap for the `bmcv` instances is zero meaning that at the root node the gap is closed. As expected, solving the linear relaxation becomes more time consuming for both increasing values of k and n_{ng}. However, bounds alone do not provide a comprehensive assessment because, on the average, solving the root node with k-loop relaxation is significantly faster than with an *ng*-route relaxation. Detailed results with lower bound values and computation times for all instances can be found in the Appendix in Tables 6–8.

5.4. Integer Solution Results

Next we summarize integer results for the `egl` and `bmcv` instances. Given the time limit of four hours (14,400s) for solving each instance, we report the number of instances solved to optimality ('Num. opt. sol.'), the number of instances where the respective relaxation produced the best lower bound among all tested relaxations ('Num best *lb*'), and the remaining gap at the end of the branch-and-price tree (using the best known upper bound *ub*). Note that the node-selection rule was best first. Aggregated results are presented in Tables 3 and 4, while detailed results for individual instances can be found in the Appendix in Tables 9–11.

Table 3: Aggregated Integer Results for `egl` Instances

	2-loop	3-loop	4b2-loop	4b3-loop	*ng*4	*ng*5	*ng*6	*ng*7
Num. opt. sol. (all/a/b/c)	5/4/1/0	6/4/2/0	6/4/2/0	6/4/2/0	4/3/1/0	3/2/1/0	4/2/2/0	3/1/2/0
Num. best *lb* (all/a/b/c)	7/6/1/0	6/4/2/0	7/4/2/1	6/4/2/0	6/3/2/1	6/3/2/1	13/2/5/6	13/2/4/7
Average gap (%)	0.69	0.62	0.57	0.58	0.48	0.43	0.44	0.43
Maximum gap (%)	1.12	1.09	1.09	1.10	1.04	1.06	1.06	1.07

Table 4: Aggregated Integer Results for `bmcv` Instances

	2-loop	3-loop	4b2-loop	4b3-loop
Num. opt. sol. (all/C/D/E/F)	75/17/21/15/22	75/17/20/16/22	76/17/19/19/21	76/17/19/19/21
Num. best *lb* (all/C/D/E/F)	85/21/23/16/25	72/17/19/14/22	72/17/18/17/20	67/17/16/15/19
Average gap (%)	0.31	0.34	0.42	0.41
Maximum gap (%)	1.48	2.03	2.23	2.26

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

For the `egl` instances, the k-loop relaxations are able to find more integer solutions, while for the `bmcv` the *ng*-route relaxation and the k-loop relaxations produce approximately the same number of optima. Whenever the time limit is reached, *ng*6 and *ng*7 produce the best lower bounds for the `egl` instances, and both the average and maximum gap is generally better for *ng*-route relaxations. In contrast, for `bmcv` instances, the 2-loop relaxation gives the best solutions both on average and with respect to the maximum gap. However, there is the tendency that the 2-loop relaxation can solve problems of groups with higher vehicle capacity (i.e. egl-lm-a and bmcv D and F) better (6, 23, and 25 best lower bounds), while the best ng-route relaxation, i.e., $ng7$, performs worse on these instances (only 2, 12, and 15 best lower bounds). On the other hand, for instances with lower capacity, i.e., egl-lm-c, bmcv C and E, the 2-loop-free relaxations results in 0, 21, and 16 best lower bounds, while $ng7$ gives 7, 20, and 21 best results. The detailed analysis of those groups of instances and the question why one relaxation performs good on one group and poorly on another is subject of the next section.

5.5. Performance Analysis by Instance Groups

There is no single relaxation that solves all types of instances best, i.e., provides the tightest lower bounds in least time. Instead, different groups of instances are best solved with different relaxations. We will identify the groups of instances and the most effective relaxations for each group and we will try to explain the reasons why a relaxation is more or less effective. For that purpose, we will provide different analyses about the time that the components of a branch-and-price require.

Lower Bounds over Time. A first insightful analysis is the evolution of the lower bound values over time for different instances and pricing relaxations. For the egl instances, the principal behavior is mainly affected by the parameter $n \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ leading to groups egl-lm-a, egl-lm-b and egl-lm-c. A typical example is shown for the three instances egl-e4-n in Figure 4. These instances have been chosen because none of the relaxations is able to prove optimality within the time limit. Hence, bounds can be compared over the full four hours.

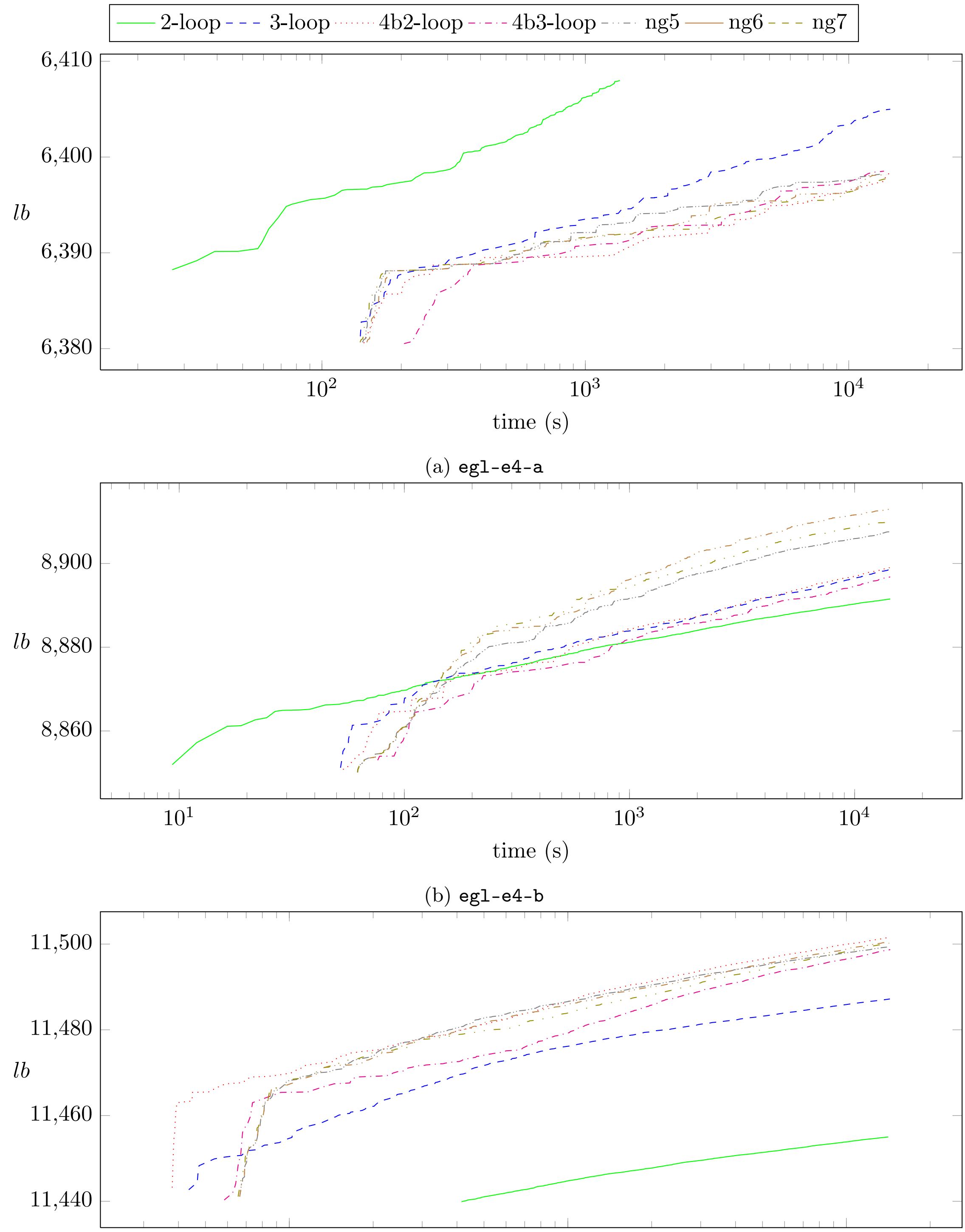
As shown in Figure 4a for egl-e4-a, the same lower bound values are reached approximately ten times faster with the 2-loop-free relaxation than with any other relaxation. Similarly, for almost all egl-lm-a instances, the 2-loop-free relaxation outperforms all other relaxations. An exception is instance egl-s1-a, were both the 3-loop-free and $4b2$-loop-free relaxation prove optimality in less time. Further exceptions are the instances egl-s2-a and egl-s4-a, where the ng-route relaxations outrun after approximately half of the available time all k-loop-free relaxations. They end up with a bound three units better than the 2-loop-free relaxation. The Appendix provides detailed figures for all egl instances and relaxations and lower bound values over time.

Typical for all egl-lm-b instances is the existence of an intersection point from where on the ng-route relaxations become more effective than the k-loop-free-relaxations. For example, in Figure 4b, the 2-loop-free relaxation performs better than the $ng6$ relaxation before that point and less effective afterwards. This intersection point is at approximately 100 seconds for egl-em-b instances and at about 1,000 seconds for egl-sm-b instances.

If just a few seconds of computation time are available, the egl-lm-c instances are solved best with an ng-route relaxation compared to 3-loop and 4-loop relaxations (2-loop-free is not competitive at all). Moreover, the ng-route relaxation delivers sometimes significantly better lower bounds at the end, see Figure 4c. Compared to the egl-lm-a group, the performance order of the relaxations is reversed for group c.

Branching Decisions and Relative Times. To explain this behavior an even more detailed analysis of the algorithms is done for the 2-loop-free and $ng6$ relaxations. The number of branch-and-bound nodes solved and the type of branching decision taken impacts which and how often a particular algorithmic component is invoked. Therefore, we recorded the number and the type of of branching decisions. Moreover, we kept track of the relative times spent (1) on updating the RMP ('update'), i.e., addition and removal of constraints and columns as well as the modification of the network, (2) for re-optimizing the RMP ('re-opt') using the primal simplex method (on average a little faster than the dual simplex algorithm), (3) for pricing ('pricing'), and (4) for other components ('other'). For egl-e4-n, these numbers are depicted in Figure 5.

One can notice in Figure 5a that for both the 2-loop-free relaxation and the $ng6$ relaxation, the number of solved branch-and-bound nodes increases from a to c. This results from the fact that due to the choice of demands and capacities, the routes are on average longer in a, become shorter for b, and are shortest for c instances. Longer routes require longer computation times per pricing iteration leading to longer computing times per node.

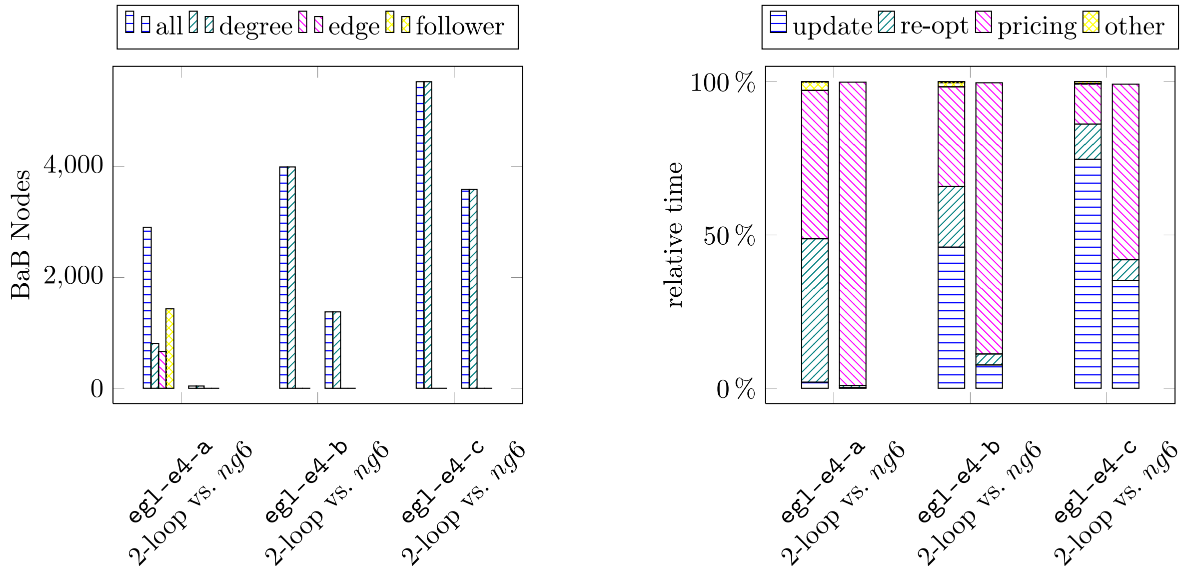


(a) `egl-e4-a`

(b) `egl-e4-b`

(c) `egl-e4-c`

Figure 4: Lower bounds over Time



(a) Num. B&B Nodes

(b) Relative Times

Figure 5: Number of Branch-and-Bound Nodes/Decisions and Relative Times spent in Components

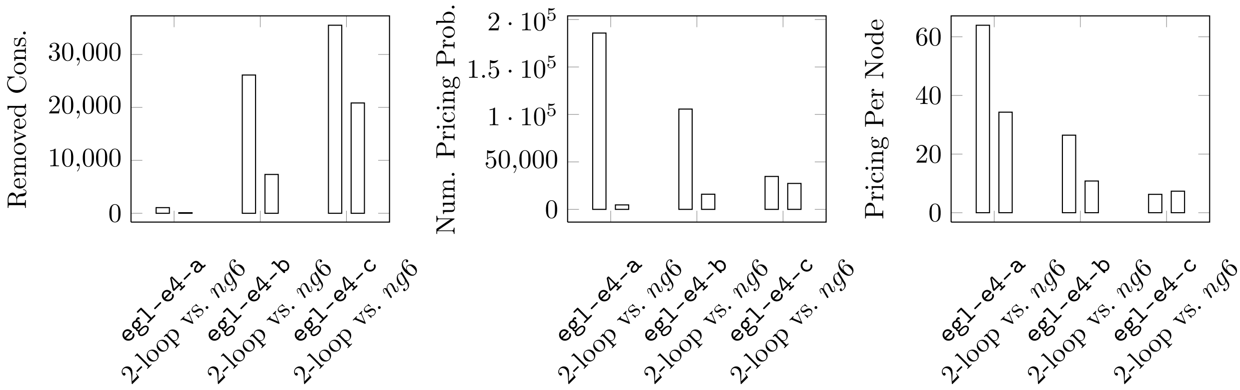
Because 2-loop is a relaxation of $ng6$, it is always less time consuming. However, while many more nodes are solved with the 2-loop-free relaxation for the `egl-`lm`-a` group, the overall number of solved nodes becomes more and more comparable for the `egl-`lm`-b` and `egl-`lm`-c` groups. This explains well why the 2-loop-free relaxation is much more effective for instances with rather long routes and only very few routes (as for `egl-`lm`-a`).

For the instance `egl-e4-a` and the 2-loop relaxation, approximately half of the nodes result from branching on followers and non-followers, and the others from branching on node degrees and edge flows, respectively. Branching on follower information entails a network modification with computing, removing, and adding edges that represent shortest paths (see Bode and Irnich, 2012, Sect 4.3.2). The structural modification is done once and at the very beginning of each branch-and-bound node. Furthermore, least-cost deadheading paths must be computed in every pricing iteration. We expected that these modifications contribute with a significant computation time. However, during our experiments we found that when solving a branch-and-bound node, the most time-consuming steps are 'update' and 're-opt' the RMP, and 'pricing'. For none of the instances where branching on followers and non-followers was performed, the time for modifying the network reaches a relevant computation time. Hence, it is not considered explicitly, but subsumed in 'update' in the further analysis. For `egl-e4-b` and `egl-e4-c`, both algorithms branch almost exclusively on node degrees.

The relative percentage of time spent in these different components is shown in Figure 5b. For both algorithms, the time spent with pricing decreases when comparing the three groups `egl-`lm`-a`, `egl-`lm`-b`, and `egl-`lm`-c`. On the one hand, for the $ng6$ relaxation, almost the entire time is spent on pricing for the instance `egl-e4-a`, while the relative time decreases to approximately 60% for the instance `egl-e4-c`. On the other hand, the 2-loop-free relaxation starts at about 60% pricing time for `egl-e4-a`; it decreases to almost 10% for the instance `egl-e4-c`.

At the same time, updating the RMP consumes relatively more time. At its extreme, updating takes about 80% of the time for the instance `egl-e4-c` when solved with the 2-loop-free relaxation. This time also increases for $ng6$, but ends up at about only 30% for the instance `egl-e4-c`.

Effort for Updates. We further analyze the effort for updating the RMP and the pricing problem. The results are shown in Figure 6.



(a) Removed Constraints

(b) Num. Pricing Problems

(c) Pricings Per Node

Figure 6: Number of Removed Constraints and Number of Pricing Problems overall/per Node

Updating the RMP consists of finding and adding the active branch-and-bound constraints regarding to node degrees and edge flows. The number of added and removed constraints for a RMP over the whole branch-and-price tree is approximately identical. Therefore, in Figure 6a, only the number of the 'removed' constraints of the RMP is plotted. Moreover, one must take care of having only compatible columns in the current RMP with regard to (non-)follower constraints and ng-route constraints. Incompatible columns are discarded by setting the lower and upper bound of this column to zero, without any complex modification of the RMP.

The Figures show that there is a strong correlation between the number of branch-and-bound nodes and the number of removed constraints, independent of the considered relaxation. The more branch-and-bound nodes are evaluated, the more constraints are removed and added. This additional effort explains partly the increasing time consumption within the RMP update.

On the other hand, while the relative time for updating the RMP increases, the relative time for pricing decreases. As shown in Figure 6b, the number of solved pricing problems for the 2-loop relaxation decreases from about 250,000 for `egl-e4-a` to merely 50,000 for `egl-e4-c`. Related to the branch-and-bound nodes, the number of solved pricing problems per node also decreases from group `a` to `c`. Even so, the absolute number of solved pricing problems slightly increases for the ng6-relaxation, the relative number of pricing problems per node decreases (see Figure 6c). In general, combining these relative numbers with the computation time needed to solve a single pricing problem, we end up with having many and computationally intensive pricings for `egl-`lm`-a` instances and less and computationally easier pricings for `egl-`lm`-c` instances. This explains the decreasing time consumption of the pricing part from groups `a` to `c`. Additionally, the ng6 relaxation is a stronger relaxation because it includes 2-loop-free relaxation, which makes the pricing problems more difficult to solve. Therefore, the time consumption for pricing is always higher compared to 2-loop elimination.

Furthermore, the ng6 relaxation as a stronger relaxation results in tighter lower bounds while having approximately the same number of pricing problems for the instance `egl-e4-c`. To conclude this section, the effect of fast pricing for the 2-loop relaxation is nullified if there is only a small number of columns to be priced out at each iteration. Then, the effect of a stronger relaxation is more important.

5.6. Strong Branching and Integer Solution Results

Strong branching is another technique often yielding better lower bounds when large branch-and-bound trees have to be explored. Instead of choosing a single variable/decision for branching, the idea is to evaluate several candidates for branching before taking the actual branching decision. Because evaluating candidates takes time, trees with less branch-and-bound nodes result. Nevertheless, the nodes provide relatively better lower bounds, which can be beneficial at the end. For a general discussion of strong branching techniques we refer to (Achterberg *et al.*, 2005).

We tested the k-loop relaxations for $k \in \{2, 3, 4\}$ and the $ng6$ and $ng7$ relaxations with five and ten candidates on the `egl` instances. We restrict strong branching to branch-and-bound nodes at levels not exceeding ten, i.e., with not more than ten nodes between the the root node and the node under consideration. Table 13 in the Appendix presents detailed results for computations with strong branching for all `egl`-lm-n instances, while Table 5 presents aggregated information.

Table 5: Aggregated Integer Results with Strong Branching for `egl` Instances

	2-loop		3-loop		4b2-loop		4b3-loop	
	sb5	sb10	sb5	sb10	sb5	sb10	sb5	sb10
Num. opt. sol. (all/a/b/c)	5/4/1/0	5/4/1/0	5/4/1/0	5/4/1/0	5/4/1/0	4/3/1/0	4/3/1/0	4/3/1/0
Num. best *lb* (all/a/b/c)	6/4/2/0	9/8/1/0	6/4/2/0	5/4/1/0	5/4/1/0	6/3/1/2	4/3/1/0	4/3/1/0
Average gap (%)	0,66	0,66	0,57	0,56	0,52	0,50	0,53	0,53
Maximum gap (%)	1,10	1,09	1,08	1,08	1,10	1,09	1,11	1,12

	$ng6$		$ng7$	
	sb5	sb10	sb5	sb10
Num. opt. sol. (all/a/b/c)	4/2/2/0	4/2/2/0	4/3/1/0	4/2/2/0
Num. best *lb* (all/a/b/c)	10/2/6/2	8/2/4/2	8/3/1/4	7/2/3/2
Average gap (%)	0,43	0,44	0,45	0,45
Maximum gap (%)	1,07	1,09	1,08	1,08

Comparing the number of optimal solutions, the k-loop and ng-relaxations are able to find about the same number of integer solutions. However, similar to the results in Section 5.4, k-loop solves more instances of groups with higher capacity (i.e. `egl`-lm-`a`) to optimality. On the other hand, looking at the number of best lower bounds among all relaxation with strong branching, $ng6$ and $ng7$ with five or ten candidates perform always better, resulting also in smaller average and maximum gaps. Overall, several lower bounds are improved compared to the integer results without strong branching (`egl-e3-b`, `egl-e4-c`, `egl-s3-a`, and `egl-s4-a`).

5.7. New Best Solutions for `egl` and `bmcv` Instances

Compared to the best known results from the literature several lower bounds for both data set were improved. Tables 9, 12 and 13 summarize the results for the standard and large-scale `egl` instances, while Tables 10 and 11 present results for the `bmcv` instances. The dataset of the large-scale `egl` instances was proposed by (Brandão and Eglese, 2008) and contains instances with up to 255 nodes, 375 edges and 347 or 375 required edges. Values printed in bold indicate new best solutions. New best lower bounds were calculated for all large-scale `egl` instances and five standard `egl` instances (`egl-e3-b`, `egl-e4-c`, `egl-s3-a`, `egl-s4-a`, `egl-s4-b`). The instance `egl-e2-b` is solved to optimality for the first time. During preliminary experiments we found a new upper bound for the instance `egl-e4-c`. The corresponding solutions are shown in Section D of the Appendix.

For previously 33 unsolved `bmcv` instances, we obtained either better lower bounds or optimal solutions in 32 cases. In detail, better lower bounds were computed for six open `C` instances (`C01`, `C09`, `C11`, `C12`, `C15` and `C23`) and four new optimal solutions were found (`C04`, `C19`, `C21` and `C24`). However, compared to Bartolini *et al.* (2012), our lower bound for `C18` is seven units worse. For the six remaining `D` instances, we computed three better lower bounds (`D21`, `D23` and `D24`) and three optimal solutions (`D08`, `D14` and `D19`). On the downside, we were not able to solve `D07` which was solved to optimality by Bartolini *et al.* (2012). Furthermore, the root node for the instances `D15` and `D18` could not be solved within four hours with some relaxations. Five better lower bounds (`E01`, `E09`, `E15`, `E18` and `E23`) and six optimal solutions (`E11`, `E16`, `E10`, `E20`, `E21` and `E24`) were found for `E` instances. Note that for the instance `E12` we ended up one unit worse than Bartolini *et al.* (2012). For `F` instances, we obtained three better lower bounds (`F18`, `F19` and `F23`) and three optimal solutions (`F04`, `F08` and `F12`). Furthermore, Bartolini *et al.* (2012) already mentioned that the objective value for `bmcv` instances is always a multiple of five because all edge costs are multiples of five. Therefore, they proved optimality for the the instance `E21`. Using the same argument, we can match the lower bounds of five additional instances with the upper bound (`D23`, `E12`, `E18`, `E23` and `F23`). In the end, twelve standard `egl` instances and 14 `bmcv` instances remain open.

6. Conclusion

In this work, different relaxations known from the node-routing context were adapted to solve the CARP with a branch-and-price approach. The adaptation to column generation-based approaches that price out new CARP tours over the original graph is by no means trivial, but is however attractive because it offers the application of highly effective pricing procedures that exploit the sparsity of the CARP network. Exploiting sparsity results in, compared to standard node-routing problems, a more intricate branching scheme, which in turn complicates the pricing. In essence, the effective approach of Bode and Irnich (2012) requires that the shortest-path pricing problem resulting from a relaxation must be able to handle two sets of tasks: One set $\mathcal{T}^E$ models elementary routes and the other set $\mathcal{T}^B$ incorporates non-follower constraints implied by the branching scheme. While for $\mathcal{T}^E$ any relaxation of elementary routes is applicable, routes must be exactly 2-loop-free regarding to tasks in $\mathcal{T}^B$.

First, we have adapted the ng-route relaxations (Baldacci *et al.*, 2011b) and the k-loop-free relaxations (Irnich and Villeneuve, 2006) leading to combined ng-route 2-loop-free relaxations and combined $(k, 2)$-loop-free relaxations. For the latter, a new labeling algorithm was developed. Its key component are strong dominance rules that we derived, based on new worst-case complexity results guaranteeing that, for a fixed parameter k, the number of labels to consider never exceeds $(k-1)!(k+1)!$ times the size of the underlying state space. Concluding, a pricing problem resulting from the $(k, 2)$-loop-free relaxation with k fixed can be solved in $\mathcal{O}(C \cdot (|E| + |V| \log |V|))$ time, where C is the vehicle capacity and $\mathcal{O}(|E| + |V| \log |V|)$ the best known bound for solving shortest-path problems using Dijkstra's algorithm.

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

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

Overall, the newly considered relaxations with loop elimination for $k = 3$ and $k = 4$ as well as the use of the ng-route relaxations outperformed the already remarkable results with elementary routes presented by Bartolini *et al.* (2012) and with the pure 2-loop-free relaxation presented by Bode and Irnich (2012). The different branch-and-price algorithms delivered 22 new best lower bounds of the `egl` and `bmcv` benchmark sets, and improved all lower bounds for the twelve large-scale `egl` instances by Martinelli *et al.* (2011b). Finally, 20 previously open instances, one for the standard `egl` and 19 for `bmcv` benchmark set, are solved to optimality for the first time.

References

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

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

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

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

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

Baldacci, R., Mingozzi, A., and Roberti, R. (2011a). Dynamic ng-path relaxation. Presentation at the ROUTE 2011 conference. `http://www.uv.es/route2011/Roberti.pdf`.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011b). New Route Relaxation and Pricing Strategies for the Vehicle Routing Problem. *Operations Research*, **59**(5), 1269–1283.

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

Bartolini, E., Cordeau, J.-F., and Laporte, G. (2012). Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming*, pages 1–44.

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

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

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

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

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

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

Bode, C. and Irnich, S. (2012). Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research*, **(accepted for publication)**. Available at `http://logistik.bwl.uni-mainz.de/158.php`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Martinelli, R., Poggi de Argão, M., and Subramanian, A. (2011b). Improved bounds for large scale capacitated arc routing problem. Preprint submitted to computers & operations research, Departamento de Informática, Rio de Janeiro, RJ 22453-900, Brazil.

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

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

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

Muyldermans, L. (2012). Personal Communication.

Polacek, M., Doerner, K., Hartl, R., and Maniezzo, V. (2008). A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, **14**, 405–423. 10.1007/s10732-007-9050-2.

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

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

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

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

Vazirani, V. (2001). *Approximation Algorithms*. Springer, New York.

Wøhlk, S. (2008). A decade of capacitated arc routing. In R. Sharda, S. Voß, B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 29–48. Springer. 10.1007/978-0-387-77778-8_2.

Appendix

A. Proofs

This section contains proofs of the worst-case complexity results for combined (k_1, k_2)-loop elimination as introduced in Section 3.4 of the paper. Note that the proofs follow similar ideas as discussed in the first article on k-cycle elimination (focused on node-routing applications) and we refer the reader to this (Irnich and Villeneuve, 2006) for a more detailed motivation.

Theorem. *Let the first set of tasks required to be k_1-loop-free and the second set of tasks to be k_2-loop-free. Then the maximum number of different set forms needed to represent any intersection $H(P_1) \cap H(P_2) \cap \cdots \cap H(P_l)$ of self-hole sets of any set of l paths is $(k_1-1)!^2 \cdot (k_2-1)!^2 \cdot \binom{(k_1-1)+(k_2-1)}{k_1-1}$. This bound is tight.*

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

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

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

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

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

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

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

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

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

Theorem. *A collection of s dominating paths $P_1\prec_{dom}P_2\prec_{dom}\dots\prec_{dom}P_s$ with identical resource vectors ending at the same node is given. Let the intersections of the corresponding self-hole sets $H(P_1),H(P_2),\dots,H(P_s)$ form a* properly *decreasing chain, i.e. $H(P_1)\supsetneq H(P_1)\cap H(P_2)\supsetneq\cdots\supsetneq\cap_{i=1}^s H(P_i)$. Then, the length q of the properly decreasing chain is bounded by $\alpha(k_1,k_2)=(k_1+k_2-1)\cdot(k_1-1)!^2\cdot(k_2-1)!^2\cdot\binom{(k_1-1)+(k_2-1)}{k_1-1}$.*

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

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

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

B. Tables

Linear Relaxation Results. The Tables 6–8 present the linear relaxation results for the `egl` and `bmcv` instances. The meaning of the table entries are as follows:

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

Integer Solution Results. The Tables 9–11 present the integer results for the `egl` and `bmcv` instances. The meaning of the table entries are as follows:

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

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

The Table 12 presents the integer results for the large-scale `egl` instances. The meaning of the table entries are as follows:

instance	name of the instance
ub_{best}	the best known upper bound At the time of writing the best upper bounds ub were computed by Martinelli *et al.* (2011b).
lb^{tree}	lower bound provided by the branch-and-price algorithm within the time limit of 10 hours (rounded up to the next integer)

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

Table 6: Linear Relaxation Results for `egl` Instances

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

Table 7: Linear Relaxation Results for `bmcv` Instances, Subsets `C` and `E`

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

Table 8: Linear Relaxation Results for `bmcv` Instances, Subsets `D` and `F`

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

Table 9: Integer Results for `egl` Instances

instance	ub_{best} or $\underline{opt}$	2-loop lb^{tree}	3b2-loop lb^{tree}	4b2-loop lb^{tree}	4b3-loop lb^{tree}	$ng4$ lb^{tree}	$ng5$ lb^{tree}	$ng6$ lb^{tree}	$ng7$ lb^{tree}	lb^{best}_{own}
`e1-a`	$\underline{3548}$	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
`e1-b`	$\underline{4498}$	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT
`e1-c`	$\underline{5595}$	5545	5551	5555	5554	5560	5571	5570	5572	5572
`e2-a`	$\underline{5018}$	OPT	OPT	OPT	OPT	OPT	5018	5018	5012	OPT
`e2-b`	$\underline{6317}$	6301	6301	6306	6305	6308	6311	**OPT**	**OPT**	**OPT**
`e2-c`	$\underline{8335}$	8242	8269	8303	8302	8300	8304	8315	8317	8317
`e3-a`	$\underline{5898}$	OPT	OPT	OPT	OPT	OPT	OPT	OPT	5898	OPT
`e3-b`	7775	7730	7735	7732	7733	7734	**7741**	**7741**	7740	**7741**
`e3-c`	10292	10191	10220	10226	10225	10226	10228	10228	10229	10229
`e4-a`	6444	6408	6405	6399	6399	6398	6399	6399	6398	6408
`e4-b`	8961	8892	8899	8900	8897	8905	8908	8913	8910	8913
`e4-c`	**11529**	11456	11488	**11501**	11499	11499	11500	**11501**	**11501**	**11501**
`s1-a`	$\underline{5018}$	OPT	OPT	OPT	OPT	5018	5018	5018	5015	OPT
`s1-b`	$\underline{6388}$	6386	OPT	OPT	OPT	6384	6384	6385	6383	OPT
`s1-c`	$\underline{8518}$	8440	8476	8500	8499	8501	8504	8509	8507	8509
`s2-a`	9884	9805	9806	9804	9803	9807	9806	9806	9808	9808
`s2-b`	13100	12970	12978	12982	12980	12991	12991	12994	12994	12994
`s2-c`	$\underline{16425}$	16351	16377	16380	16379	16393	16392	16393	16393	16393
`s3-a`	10220	10160	10154	10150	10149	10153	10153	10154	10152	10160
`s3-b`	13682	13630	13629	13627	13625	13637	13640	13644	13640	13644
`s3-c`	$\underline{17188}$	17096	17122	17125	17123	17138	17143	17142	17141	17143
`s4-a`	12268	12149	12147	12142	12141	12150	**12151**	**12151**	12150	**12151**
`s4-b`	16283	16104	16106	16105	16104	**16113**	16111	16111	16108	**16113**
`s4-c`	20481	20374	20397	20406	20405	20418	20420	20422	20423	20423
Num. lb^{best}_{own}		7	6	7	6	6	6	12	11	

Table 10: Integer Results for `bmcv` Instances, Subsets `C` and `E`

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

Table 11: Integer Results for `bmcv` Instances, Subsets `D` and `F`

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

Table 12: Integer Results for Large-Scale `egl` Instances

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

Finally, the Table 13 presents the integer results for strong branching using the standard `egl` instances. The meaning of the table entries are as follows:

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

Table 13: Integer Solutions with Strong Branching for `egl` Instances

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

C. Best Known Lower and Upper Bounds

The Tables 14–16 list the best known lower and upper bounds for the standard and large-scale `egl` instances and the `bmcv` instances. The meaning of the table entries are as follows:

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

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

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

Table 14: Best Known Bounds for the `egl` Instances

instance	lb_{best}	computed by	ub_{best}	computed by	opt	proved by
`egl-e1-a`			3548	Lacomme *et al.* (2001)	3548	Longo *et al.* (2006)
`egl-e1-b`			4498	Lacomme *et al.* (2001)	4498	Baldacci and Maniezzo (2006)
`egl-e1-c`			5595	Lacomme *et al.* (2001)	5595	Bartolini *et al.* (2012)
`egl-e2-a`			5018	Lacomme *et al.* (2001)	5018	Baldacci and Maniezzo (2006)
`egl-e2-b`			6317	Brandão and Eglese (2008)	6317	own
`egl-e2-c`			8335	Brandão and Eglese (2008)	8335	Bartolini *et al.* (2012)
`egl-e3-a`			5898	Lacomme *et al.* (2001)	5898	Longo *et al.* (2006)
`egl-e3-b`	7744	own	7775	Polacek *et al.* (2008)		
`egl-e3-c`	10244	Bartolini *et al.* (2012)	10292	Polacek *et al.* (2008)		
`egl-e4-a`	6408	Bode and Irnich (2012)	6444	Santos *et al.* (2010)		
`egl-e4-b`	8935	Bartolini *et al.* (2012)	8961	Bartolini *et al.* (2012)		
`egl-e4-c`	11512	own	11529	own		
`egl-s1-a`			5018	Lacomme *et al.* (2001)	5018	Baldacci and Maniezzo (2006)
`egl-s1-b`			6388	Brandão and Eglese (2008)	6388	Bartolini *et al.* (2012)
`egl-s1-c`			8518	Lacomme *et al.* (2001)	8518	Bartolini *et al.* (2012)
`egl-s2-a`	9825	Bartolini *et al.* (2012)	9884	Santos *et al.* (2010)		
`egl-s2-b`	13017	Bartolini *et al.* (2012)	13100	Brandão and Eglese (2008)		
`egl-s2-c`			16425	Brandão and Eglese (2008)	16425	Bartolini *et al.* (2012)
`egl-s3-a`	10165	own	10220	Santos *et al.* (2010)		
`egl-s3-b`	13648	Bartolini *et al.* (2012)	13682	Polacek *et al.* (2008)		
`egl-s3-c`			17188	Bartolini *et al.* (2012)	17188	Bartolini *et al.* (2012)
`egl-s4-a`	12153	own	12268	Santos *et al.* (2010)		
`egl-s4-b`	16113	own	16283	Fu *et al.* (2010)		
`egl-s4-c`	20430	Bartolini *et al.* (2012)	20481	Bartolini *et al.* (2012)		
`egl-g1-a`	976907	own	1049708	Martinelli *et al.* (2011a)		
`egl-g1-b`	1093884	own	1140692	Martinelli *et al.* (2011a)		
`egl-g1-c`	1212151	own	1282270	Martinelli *et al.* (2011a)		
`egl-g1-d`	1341918	own	1420126	Martinelli *et al.* (2011a)		
`egl-g1-e`	1482176	own	1583133	Martinelli *et al.* (2011a)		
`egl-g2-a`	1067262	own	1129229	Martinelli *et al.* (2011a)		
`egl-g2-b`	1185221	own	1255907	Martinelli *et al.* (2011a)		
`egl-g2-c`	1311339	own	1417145	Martinelli *et al.* (2011a)		
`egl-g2-d`	1446680	own	1516103	Martinelli *et al.* (2011a)		
`egl-g2-e`	1581459	own	1701681	Martinelli *et al.* (2011a)		

Table 15: Best Known Bounds for the bmcv Instances, Subsets C and E

instance	lb_{best}	computed by	ub_{best}	computed by	opt	proved by
C01	4145	own	4150	Beullens *et al.* (2003)		
C02			3135	Beullens *et al.* (2003)	3135	Beullens *et al.* (2003)
C03			2575	Beullens *et al.* (2003)	2575	Bartolini *et al.* (2012)
C04			3510	Beullens *et al.* (2003)	3510	own
C05			5365	Brandão and Eglese (2008)	5365	Bartolini *et al.* (2012)
C06			2535	Beullens *et al.* (2003)	2535	Bartolini *et al.* (2012)
C07			4075	Beullens *et al.* (2003)	4075	Bartolini *et al.* (2012)
C08			4090	Beullens *et al.* (2003)	4090	Bartolini *et al.* (2012)
C09	5245	own	5260	Brandão and Eglese (2008)		
C10			4700	Brandão and Eglese (2008)	4700	Bartolini *et al.* (2012)
C11	4615	own	4630	Mei *et al.* (2009)		
C12	4235	own	4240	Beullens *et al.* (2003)		
C13			2955	Beullens *et al.* (2003)	2955	Bartolini *et al.* (2012)
C14			4030	Beullens *et al.* (2003)	4030	Bartolini *et al.* (2012)
C15	4920	own	4940	Beullens *et al.* (2003)		
C16			1475	Beullens *et al.* (2003)	1475	Bartolini *et al.* (2012)
C17			3555	Beullens *et al.* (2003)	3555	Bartolini *et al.* (2012)
C18	5580	Bartolini *et al.* (2012)	5620	Santos *et al.* (2010)		
C19			3115	Beullens *et al.* (2003)	3115	own
C20			2120	Beullens *et al.* (2003)	2120	Beullens *et al.* (2003)
C21			3970	Beullens *et al.* (2003)	3970	own
C22			2245	Beullens *et al.* (2003)	2245	Beullens *et al.* (2003)
C23	4075	own	4085	Beullens *et al.* (2003)		
C24			3400	Beullens *et al.* (2003)	3400	own
C25			2310	Beullens *et al.* (2003)	2310	Beullens *et al.* (2003)
E01	4900	own	4910	Brandão and Eglese (2008)		
E02			3990	Beullens *et al.* (2003)	3990	Bartolini *et al.* (2012)
E03			2015	Beullens *et al.* (2003)	2015	Beullens *et al.* (2003)
E04			4155	Beullens *et al.* (2003)	4155	Bartolini *et al.* (2012)
E05			4585	Brandão and Eglese (2008)	4585	Bartolini *et al.* (2012)
E06			2055	Beullens *et al.* (2003)	2055	Beullens *et al.* (2003)
E07			4155	Beullens *et al.* (2003)	4155	Bartolini *et al.* (2012)
E08			4710	Beullens *et al.* (2003)	4710	Bartolini *et al.* (2012)
E09	5805	own	5820	Tang *et al.* (2009)		
E10			3605	Beullens *et al.* (2003)	3605	Beullens *et al.* (2003)
E11					4650	own
E12			4180	Bartolini *et al.* (2012)	4180	Bartolini *et al.* (2012)
E13			3345	Beullens *et al.* (2003)	3345	Bartolini *et al.* (2012)
E14			4115	Beullens *et al.* (2003)	4115	Bartolini *et al.* (2012)
E15	4200	own	4205	Santos *et al.* (2010)		
E16			3775	Beullens *et al.* (2003)	3775	own
E17			2740	Beullens *et al.* (2003)	2740	Beullens *et al.* (2003)
E18			3835	Beullens *et al.* (2003)	3835	own
E19			3235	Beullens *et al.* (2003)	3235	own
E20			2825	Beullens *et al.* (2003)	2825	own
E21			3730	Beullens *et al.* (2003)	3730	Bartolini *et al.* (2012)
E22			2470	Beullens *et al.* (2003)	2470	Bartolini *et al.* (2012)
E23			3710	Beullens *et al.* (2003)	3710	own
E24			4020	Beullens *et al.* (2003)	4020	own
E25			1615	Beullens *et al.* (2003)	1615	Beullens *et al.* (2003)

Table 16: Best Known Bounds for the `bmcv` Instances, Subsets `D` and `F`

instance	lb_{best}	computed by	ub_{best}	computed by	*opt*	proved by
D01			3215	Beullens *et al.* (2003)	3215	Beullens *et al.* (2003)
D02			2520	Beullens *et al.* (2003)	2520	Beullens *et al.* (2003)
D03			2065	Beullens *et al.* (2003)	2065	Beullens *et al.* (2003)
D04			2785	Beullens *et al.* (2003)	2785	Beullens *et al.* (2003)
D05			3935	Beullens *et al.* (2003)	3935	Beullens *et al.* (2003)
D06			2125	Beullens *et al.* (2003)	2125	Beullens *et al.* (2003)
D07			3115	Beullens *et al.* (2003)	3115	Bartolini *et al.* (2012)
D08			3045	Beullens *et al.* (2003)	3045	own
D09			4120	Beullens *et al.* (2003)	4120	Beullens *et al.* (2003)
D10			3340	Beullens *et al.* (2003)	3340	Bartolini *et al.* (2012)
D11			3745	Tang *et al.* (2009)	3745	Beullens *et al.* (2003)
D12			3310	Beullens *et al.* (2003)	3310	Beullens *et al.* (2003)
D13			2535	Beullens *et al.* (2003)	2535	Beullens *et al.* (2003)
D14			3280	Beullens *et al.* (2003)	3280	own
D15			3990	Beullens *et al.* (2003)	3990	Beullens *et al.* (2003)
D16			1060	Beullens *et al.* (2003)	1060	Beullens *et al.* (2003)
D17			2620	Beullens *et al.* (2003)	2620	Beullens *et al.* (2003)
D18			4165	Beullens *et al.* (2003)	4165	Beullens *et al.* (2003)
D19			2400	Beullens *et al.* (2003)	2400	own
D20			1870	Beullens *et al.* (2003)	1870	Beullens *et al.* (2003)
D21	3005	own	3050	Beullens *et al.* (2003)		
D22			1865	Beullens *et al.* (2003)	1865	Beullens *et al.* (2003)
D23			3130	Beullens *et al.* (2003)	3130	own
D24	2705	own	2710	Beullens *et al.* (2003)		
D25			1815	Beullens *et al.* (2003)	1815	Beullens *et al.* (2003)
F01			4040	Beullens *et al.* (2003)	4040	Beullens *et al.* (2003)
F02			3300	Beullens *et al.* (2003)	3300	Beullens *et al.* (2003)
F03			1665	Beullens *et al.* (2003)	1665	Beullens *et al.* (2003)
F04			3485	Beullens *et al.* (2003)	3485	own
F05			3605	Beullens *et al.* (2003)	3605	Beullens *et al.* (2003)
F06			1875	Beullens *et al.* (2003)	1875	Beullens *et al.* (2003)
F07			3335	Beullens *et al.* (2003)	3335	Beullens *et al.* (2003)
F08			3705	Beullens *et al.* (2003)	3705	own
F09			4730	Beullens *et al.* (2003)	4730	Beullens *et al.* (2003)
F10			2925	Beullens *et al.* (2003)	2925	Beullens *et al.* (2003)
F11			3835	Beullens *et al.* (2003)	3835	Beullens *et al.* (2003)
F12			3395	Beullens *et al.* (2003)	3395	own
F13			2855	Beullens *et al.* (2003)	2855	Beullens *et al.* (2003)
F14			3330	Beullens *et al.* (2003)	3330	Beullens *et al.* (2003)
F15			3560	Beullens *et al.* (2003)	3560	Beullens *et al.* (2003)
F16			2725	Beullens *et al.* (2003)	2725	Beullens *et al.* (2003)
F17			2055	Beullens *et al.* (2003)	2055	Beullens *et al.* (2003)
F18	3065	Bartolini *et al.* (2012)	3075	Beullens *et al.* (2003)		
F19	2515	own	2525	Beullens *et al.* (2003)		
F20			2445	Beullens *et al.* (2003)	2445	Beullens *et al.* (2003)
F21			2930	Beullens *et al.* (2003)	2930	Beullens *et al.* (2003)
F22			2075	Beullens *et al.* (2003)	2075	Beullens *et al.* (2003)
F23			3005	Beullens *et al.* (2003)	3005	own
F24			3210	Beullens *et al.* (2003)	3210	Beullens *et al.* (2003)
F25			1390	Beullens *et al.* (2003)	1390	Beullens *et al.* (2003)

D. Integer Solutions

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

New Upper Bounds and Best Known Solutions.

`egl-e4-c` $ub = 11529$

veh 1	1=2=3–2=4–5=6–5=4–2–1 load 127
veh 2	1–2–4–5=7–8=9=10–11=59–69–4–2–1 load 130
veh 3	1–2–4–5–7=8–9–10–11=48–47=46–44–59–69–4–2–1 load 130
veh 4	1–2–4–5–7–8–9–10=11–12–76–20–18–72–73=74–73–72=18–20–76–12–11–10–9–8–7–5–4–2–1 load 129
veh 5	1–2–4–5–7–8–9–10–11=12–76–20=19=50=52=54–52–50=49–47=48–47–46=45–44–59–69–4–2–1 load 130
veh 6	1–2–4–5–7–8–9–10–11–12=16=13=14=15=17–15=77=76–12–11–10–9–8–7–5–4–2–1 load 130
veh 7	1–2–4–5–7–8–9–10–11–12–16–13=77–15=18=72=73=71=70–71=72–18=20=76–12–11–10–9–8–7–5–4–2–1 load 130
veh 8	1–2–4–69–59–44–46–47=49=51=53–51–21=19=18=20–76=12–11–10–9–8–7–5–4–2–1 load 130
veh 9	1–2–4–69–59–44–46–47–49–51–21=22–75=23–31=32=33–37=38–37–39–40–41–42–57–58–69–4–2–1 load 130
veh 10	1–2–4–69–59–44–46–47–49–51–21–22–75–23=31=30–31–23=26=27–26=25–24=22–21–51–53=52–50–49–47 –46–44–59–69–4–2–1 load 130
veh 11	1–2–4–69–59–44–46–47–49–51–53=24=25=29=28=26–25=75=22–21=51–49–47–46–44–59–69–4–2–1 load 130
veh 12	1–2–4–69–58–57–42–41–35–32=34–32=35–41–42–57–58–69–4–2–1 load 129
veh 13	1–2–4–69–58–57–42–41–40–39–37–33=36–33=37=39=40–41–42–57–58–69–4–2–1 load 128
veh 14	1–2–4–69–58=57–42–41=40–39=35=41=42–57–58–69–4–2–1 load 127
veh 15	1–2–4–69–59=58–57=42=43=44–59–69–4–2–1 load 128
veh 16	1–2–4–69–59–44=45–46=44=59–69=4–2–1 load 128
veh 17	1–2–4–69–58=60=67=56=55–56=42–57–58–69–4–2–1 load 130
veh 18	1–2–4–69–58–60=61–60–67=62=63=65–63=64–63–62–60–58–69–4–2–1 load 130
veh 19	1–2–4–69=58–60=62=66=68–66–62–60–58–59=69–4–2–1 load 127

New Optimal Solutions.

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

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

`C04` $opt = 3510$

veh 1	44=45=46=47=2=1=48=44 load 280
veh 2	44–45–46=26=25=3–5–6=24–17=18=19=16–15=18=19=20–21=28–27–3–4=26–46–45–44 load 285
veh 3	44–45–46–26–25–3=27=22–21–20–29–31=35=32–35=34=33–11=38=39=41–42–43–45–44 load 285
veh 4	44–45–46–47=4–3–27=28=29=31=30=29=20=21=22–23=5–3–25=43–45–44 load 300
veh 5	44–45–46–26–25–3–5–6=8=7=14=13–14=17=24=23=22–27–3–25–26–46–45–44 load 300
veh 6	44–45=43=42=36=37–36=35=37=39=40=41=42–43=45–44 load 300 44=49=48–44 load 130
veh 7	44–49=51=50–51–54=55=56=57=60=59=58=56=54=52=53–52=51=54–55=49–44 load 290

`C19` $opt = 3115$

veh 1	31–27–14–46–47=48=50=43=9=10–9=11=16=17–16=15=10–15=18=19–20–6–5–23–26–27–31 load 300
veh 2	31=30=32–33=13=12–13=51–52=37=55–37=35=36–35=33–32–30–31 load 300
veh 3	31–30–40=39=38=44=49=45=14=46–14–27–31 load 300
veh 4	31=27=14–46=22=20=48=19=20=22=21–5–23=26–27–31 load 300
veh 5	31–27=26=24–26=28=29=25–3–2–34=33=32=29–28=31 load 220
veh 6	31–30=40–39–42=38=54=53–54–38=56=58=59=60=57–60=41=42–41=59–58=42=39–40–30–31 load 300

C21 $opt = 3970$

veh 1 34–31–33–38–40–43=44=49=2=1=7=10=9=8=47–45=43–40–38–33–31–34 load 300
veh 2 34–35–37–19=21=53=42–41=51=50=3=52=4–52=51–50=49–44=45–43=40=38–33–31–34 load 300
veh 3 34–35–37–19=20=22=60=59=58=57=11=12–11=13–15–16=17=18–19–37–35–34 load 290
veh 4 34–31–33–38–40=41=42=21=20=17=15=14=13=15=16=22=21–19–37–35–34 load 280
veh 5 34–35=37=19=18=23=37=36=35–34 load 265
veh 6 34–31=23=24–23=25=26=28–26=27=25–27=29–27=30=32–30=31–34 load 300
veh 7 34=31=32=33=38=39–38=36–35=34 load 210
veh 8 34–31=33–38–40–43–44=49=47=48–47=45=46–45–43–40–38–33–31–34 load 300

C24 $opt = 3400$

veh 1 49–47–48–42–52=51–35–30=3–4=53=55–54=60=56–1=2=6=21–6=5=52=42–48–47–49 load 300
veh 2 49–47–48–13–12=5=3=4=2=54=55=28–62=61=59=58=57=60=61=55–53=29=3–30–35–36–31=43=41–39–47–49 load 300
veh 3 49–7=10=11=12=13=48=42=41=40=44=43=42–48=47=49 load 285
veh 4 49–47=39=46=45=66=68=8–9=15=50=14–50=9=8=7=49 load 295
veh 5 49–7=16=17=18=20=19=11=13=10–7–49 load 300
veh 6 49–47–39=40–44–32–33–38=37=34–37=26=27=25=24=76–70–75–74–63=38=33–32–44–40–41=39–47–49 load 295
veh 7 49–47=46=44=32=69–65=38=64–65=69=67=45–46–47–49 load 265

D08 $opt = 3045$

veh 1 45–46=35=36=33=32=1–32=31=30–31=12–11=34=35=37–43=44–45 load 575
veh 2 45–46–48–49–51=50=60=61=2=40=41–42=38=5=4=3=39=41–44=45 load 600
veh 3 45–46–48=47=16–18=54=29=19=18–16=17=15=10=9=34=33–36=37=43=42=41=44–45 load 600
veh 4 45=46=48=49=47=55=53–55=56–57=64=65=66=29–54=56=57=62=63=58=52=59–52=53=51=49–48–46–45 load 585

D14 $z = y3280$

veh 1 34–35–37=47–48=36=39=40–39=38=33–32=22–21–1=2–1=14–15=16–17–35–34 load 600
veh 2 34=35–17=16=20=19=18=51=52=4=1=21=22=23=3=2=5=4=53–4=19–18=17–35–34 load 600
veh 3 34–35=37=36=33=32=31–11–25=24–26=28=30–28=29–12–27=26=24=23–22–21=15=14=20–16–17=35–34 load 570
veh 4 34=33–38=42=45=46–45=44–10–43=42=40=41=49=50=41=39–38–33–34 load 580

D19 $z = y2400$

veh 1 31–28=26=24–26=23–5–21=22–20=48=19–48=47–48=50=43=9=10–9=11=16=17–16=15=10–15=18=19=20=22=46=14 –27=31 load 590
veh 2 31–28–29=32=30–32=33=13=12–13=51–52=37–55–57–60=41–42=58–59=60=57–55=37=35=36–35=33=34–2–3–25=29=28 =31 load 535
veh 3 31=30=40=39=42–39=38=54=53–54–38=56=58=59=41=42=38=44=49=45=14=27=26–27–31 load 595

E11 $opt = 4650$

veh 1 45–71–72–73–74–47–48=2=1=76=77–50–49=48=47=74–73–72–71–45 load 300
veh 2 45–71–72–73–67=44=43–3=14=2=50=49=52=65=68=69=74–73–72–71–45 load 300
veh 3 45–42=41=7=6=5=4=13–4=3=43=42–45 load 295
veh 4 45–71–21=22=9=8=11=12=5–12=6–42=45 load 300
veh 5 45–42=6–7=8–9=10=80=78=79–78=10–78=19=75–19=22–21=71–45 load 300
veh 6 45–42–41=22=38=27=28–15–20=25–29=24–23–39–64=69=70–71–45 load 300
veh 7 45–71–21=26=29=25=24=23=30–23=40=63=68=47–46–66=67–73–72–71–45 load 300
veh 8 45–71–72–73–74–47–48–49–50=77=51–53=56–61=60=63–40–23=39=64=70–71–45 load 300
veh 9 45–71=72=73=67=44=46=66=74=73–72=70=71=45 load 125
veh 10 45–71–70–69–68–65–52=62–61–60=58=59–58=57–58=55=54–55=56=61=62=65–68–69–70–71–45 load 300

E16 $opt = 3775$

veh 1 54–55–56=53=35–34=50–54 load 260
veh 2 54–55=53=52=36–52=53=55–54 load 145
veh 3 54=55=56=57=9=8=7=6=52=36–52=6=7=8=9=57–56=55=54 load 255
veh 4 54–55–53–52–36–60–59–1–2=3=5=6–7–8=57=56–55–54 load 300
veh 5 54–55–56–57–8–7=58=22=21–4=22–26=20=14=11=10=9–57–56–55–54 load 295
veh 6 54–50–45–43=33–43=30–32–29=44=45=50=54 load 300
veh 7 54–50–45=43–45–44=49=48=11=10–9–57–56–55–54 load 300
veh 8 54–55–56–57–9–10=51–10–11=12=47–13=17=16–17=14=19=15=18–15=16=47–46=31–46=47=13=12=48–49 =42=50–54 load 295

E19 *opt* = 3235

veh 1	27–23–22=20–4–19–5–6–16=45–46=47=51=40=9=14=15–14=9=40=51=47=46–45=16–6–5–19–4–20=22–23–27 load 300
veh 2	27–23–13–42=49=43=48=39–48=16–18=49=13–23–27 load 300
veh 3	27–23–22=19–5–17=18=16–48=45=46–47=14=15–14=47–46=45=48–16=18=17–5–19=22–23–27 load 300
veh 4	27–26–36–35–38–44=41=50=42=13=23=27 load 300
veh 5	27=26=28–29=11=10–11=12=54–12=34=55–34=33=31–29=30–2–3–21=25=24=27 load 300
veh 6	27–26–36–35–38=61=60–61=62=63=59–56=33–31=32–31=29=28=25–24=22=23–27 load 300
veh 7	27–26–36–35=44=53=57–53=58=60=44=38–37=62–63=37=38=35=36=26–27 load 300

E20 *opt* = 2825

veh 1	42=41=40=39–40=47=48–47=46=44=42 load 145
veh 2	42–43=38–36=34=35–34=32=33–32=13–32=31=30–43–42 load 300
veh 3	42–43–30=27=29=28=27–29=11=4=10–4=5=7–5=51–45–44–42 load 290
veh 4	42–44–46=45=11=12=26–12=30=43–42 load 260
veh 5	42–43–30–31–25=24=14=25=26=24–25=31=36=37–36=38=41–42 load 295
veh 6	42–44–46=50=49–23–56=9=1=3=2=54=52=51–45–44–42 load 290
veh 7	42=43=44=45=51=50=53–55–1=2=8–6=3–1=55=53=52–51–45–44–42 load 255

E21 *opt* = 3730

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

E24 *opt* = 3510

veh 1	69–96–8=10–12=44–2=1–2=44=45=6–9=43=42–70–69 load 300
veh 2	69–70–42–40–47=46–37–50=4=5=3=2=56=58=57–58=53–54–52=51=4=6=47=40–42–70–69 load 295
veh 3	69–70–74–39=40–47–6–45=3=53=54=55–59=60–48=49=38–41–39=74–70–69 load 300
veh 4	69–70–74–39–41–38=49=37=50=51=55=59=57=54=52=5–4–6=9=10=43=8=96–69 load 300
veh 5	69=96=11=7=94=93=7=92=97–92=93–94=95–96–69 load 215
veh 6	69–70–74–73–72–64–75=76=62–90–61=76=63–76=26=25=28–25=75=64–72–73=74–70–69 load 300
veh 7	69–70=42=40=41=38=29–30=64=72=41=39=73=72=71=70–69 load 290 69=70=74=71=65–66=68=67–68 =95=96–69 load 235

F04 *opt* = 3485

veh 1	51–56=55=4–55=56–51 load 180
veh 2	51–52–53–54=32=8–10–11–7=28–7–6=17=16–17=20–27=26=25–26=29=33–30=31–53–52–51 load 600
veh 3	51–52–50–49=43–44=42–38=36=35=34=25=24=23=22=21–22=19–18=21=20=27=28=29–33=32=31=53–52–51 load 600
veh 4	51–52=50=30=33=34–35=24=23=36=37=38=42=41=40–41–15–39=42=43=44=46=48=47=46=45–46–48 =49=50–52–51 load 600
veh 5	51=52=53=54=5=55=4–55=5=54=53=52=51 load 350
veh 6	51=56=57=63=62–63=64=70–66=69=68–69=67–69=68–69=66–70=64=63=62–63=57=56=51 load 515
veh 7	51=57=59=60=61–60=62=59=58–59–62=64=65=69=67–69=65=64=62–59=58–59=62=60=61–60=59=57=51 load 445

F08 *opt* = 3705

veh 1	50–49=48–42=40=39=15–16=36=35–36=37=34=2–34=3–34–37=38=39=18–39–38=41–42–48–49–50 load 565
veh 2	50=51=54–55–57=56=33=32=63–32=4=45=46=49=50 load 520
veh 3	50–49–46=47–46=44=5–12–6=43=47=48=42=41=40=51–50 load 400
veh 4	50–51–54–55=53=52=20=19=52=21–52=53=54–51–50 load 495
veh 5	50–51–54–55=57=59=60=67=68=74–73=21=68–67=64–65=66=61=58–62=33–62=58=59–60=53–54–51–50 load 580

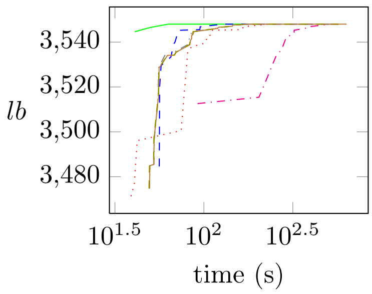
F12 *opt* = 3395

veh 1	21=20=18=17=16=19=21 load 85
veh 2	21=22=35=36=34=29=28–29=24–46=47=44–47=41=23=21 load 600
veh 3	21–19=18–20=15=14=43–14=7=15=6=64–72–70–67=39=40=6=17–18–19–21 load 600
veh 4	21–19–16–17=38=39–40=67=65=66=68–69–2=8–30=32=34=33=12=50–12=10–9=31=32–37=36–37=39–38=22 =16=18–19–21 load 600
veh 5	21–23=24=46–45=25=48=26=28=27=13=49–53–50=11=3–11=73–11=10=9=8=30–32=37=35–22–21 load 600

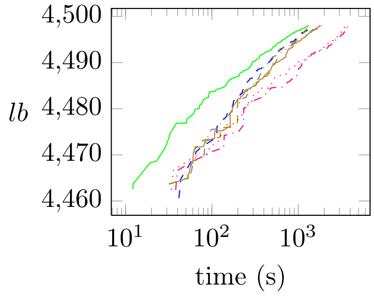
E. Figures

This section presents different analyses about the time that the components of a branch-and-price require. Every page depicts three figure groups presenting data for one of the eight instance groups `egl-e1-`n to `egl-e4-`n and `egl-s1-`n to `egl-s4-`n with $n \in \{\texttt{a}, \texttt{b}, \texttt{c}\}$. First, those instances solved to optimality are mentioned. The first figure group shows the evolution of the lower bound values over time for different instances and pricing relaxations and correspond to Figure 4 in the main paper. Thereafter, the number of branch-and-bound nodes solved and the type of branching decision taken impacts which and how often a particular algorithmic component is invoked (related to Figure 5). The last figure group is concerned with the effort for solving the pricing problem (corresponding to Figure 6 of the main paper).

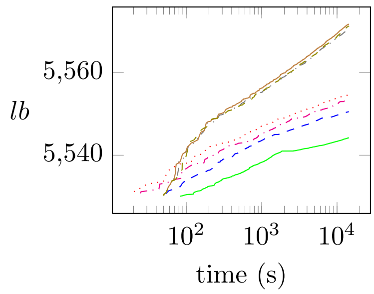
Solved to optimality: `egl-e1-a`, `egl-e1-b`



(a) `egl-e1-a`

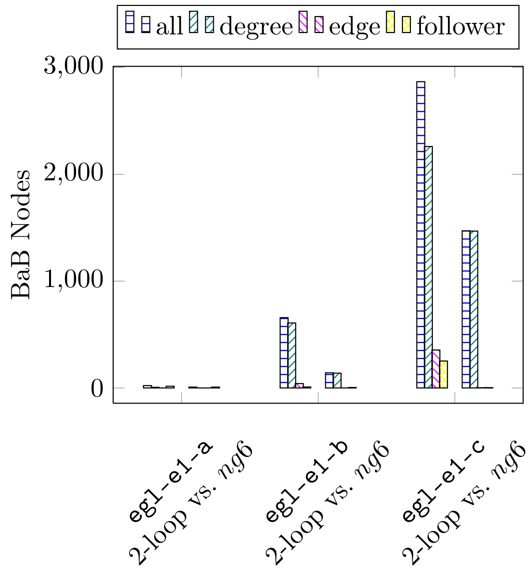


(b) `egl-e1-b`

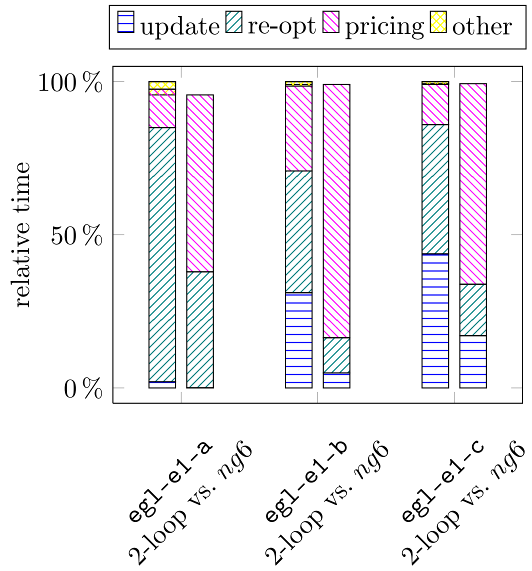


(c) `egl-e1-c`

Figure 7: Lower bounds over Time

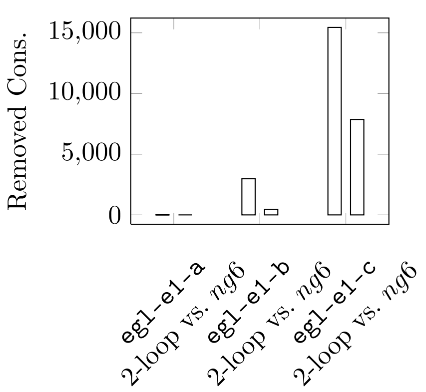


(d) Num. B&B Nodes

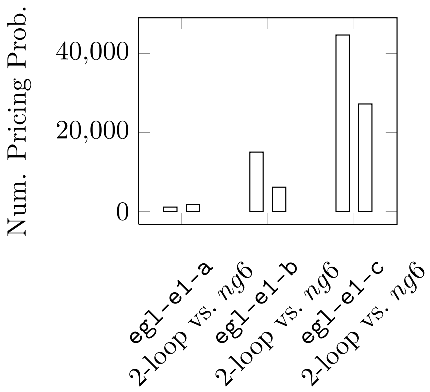


(e) Relative Times

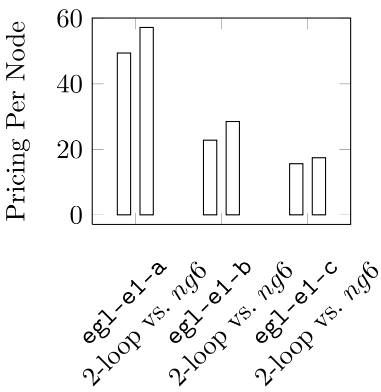
Figure 8: Number of Branch-and-Bound Nodes/Decisions and Relative Times spent in Components



(a) Removed Constraints



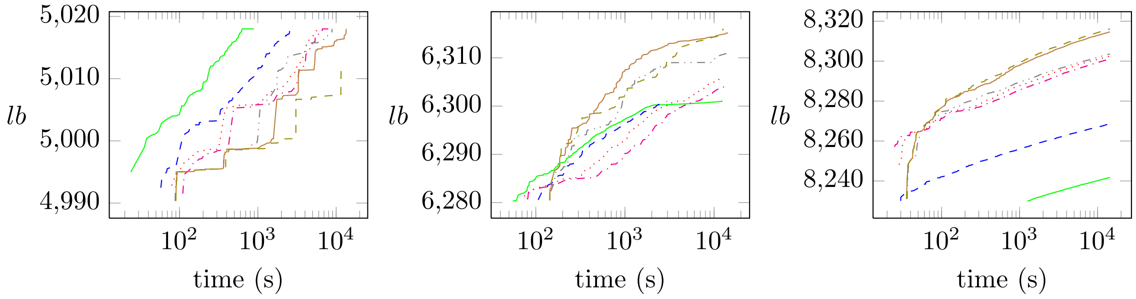
(b) Num Pricing Problems



(c) Pricings Per Node

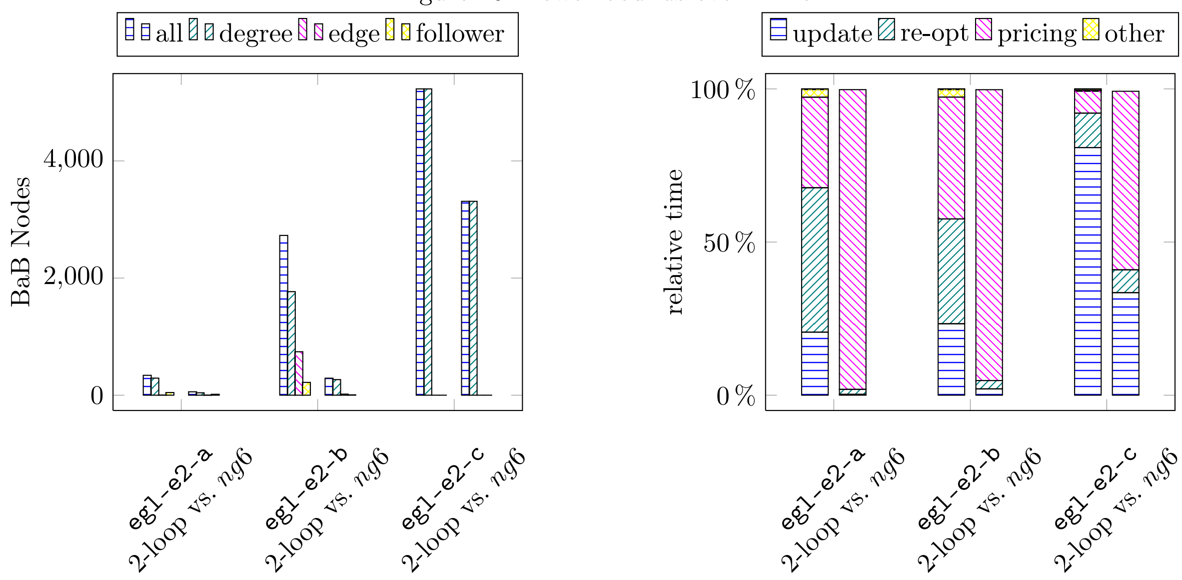
Figure 9: Number of Pricing Problems overall/per Node

Solved to optimality: `egl-e2-a`



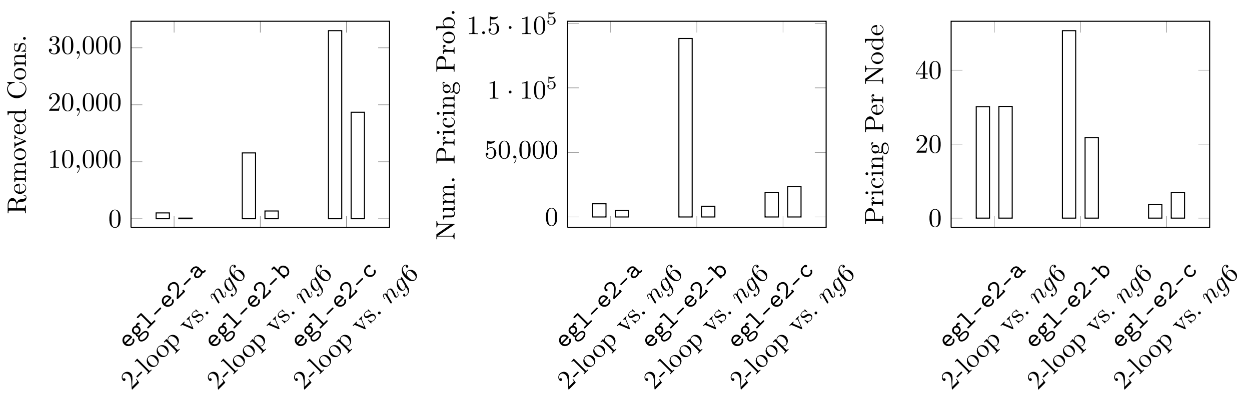
(a) `egl-e2-a` (b) `egl-e2-b` (c) `egl-e2-c`

Figure 10: Lower bounds over Time



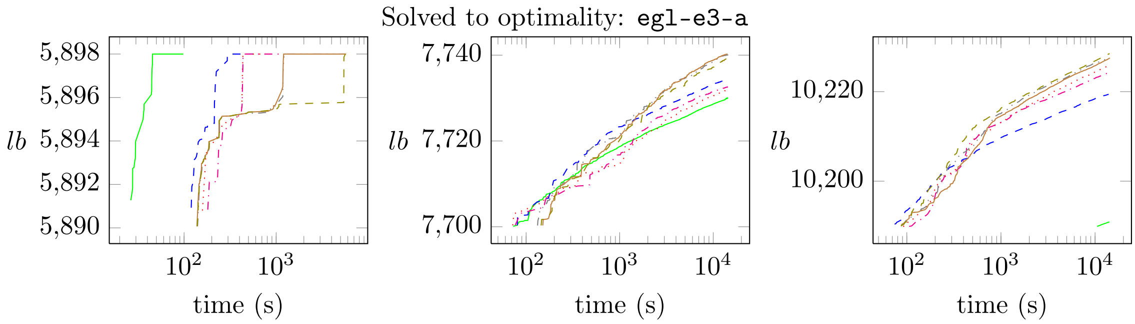
(d) Num. B&B Nodes (e) Relative Times

Figure 11: Number of Branch-and-Bound Nodes/Decisions and Relative Times spent in Components



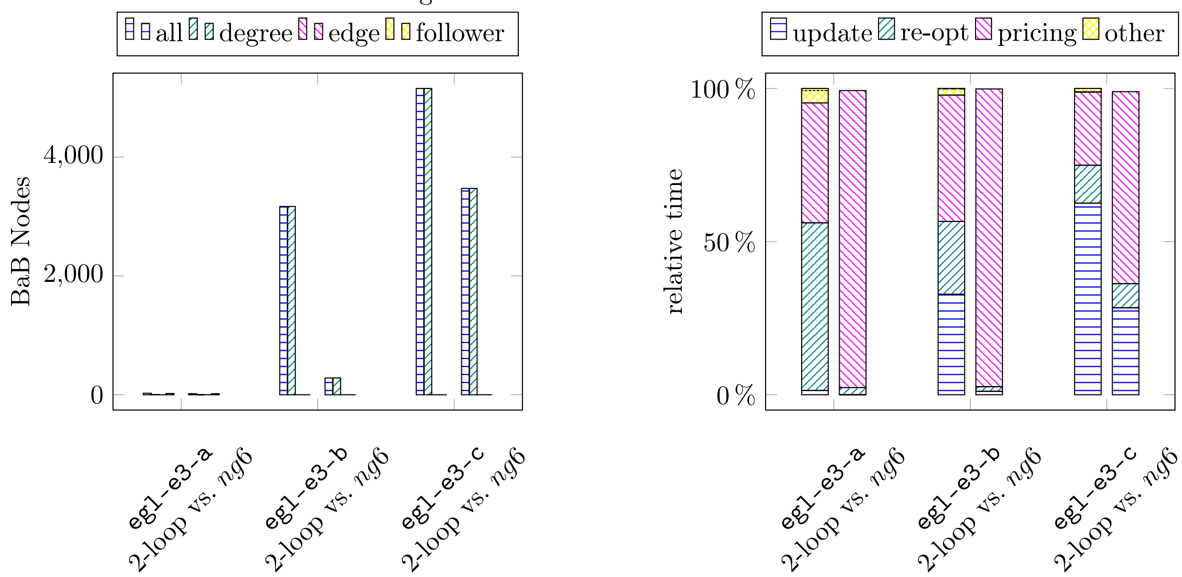
(a) Removed Constraints (b) Num Pricing Problems (c) Pricings Per Node

Figure 12: Number of Pricing Problems overall/per Node



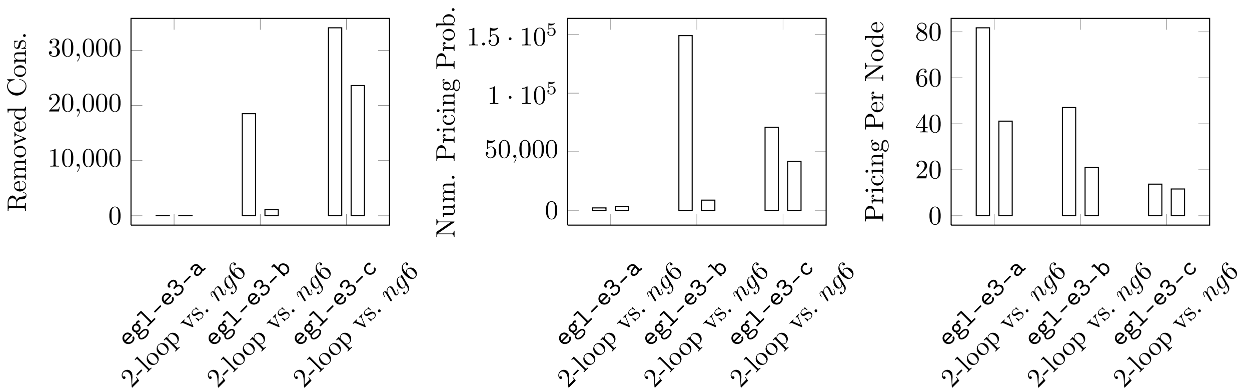
(a) `egl-e3-a` (b) `egl-e3-b` (c) `egl-e3-c`

Figure 13: Lower bounds over Time



(d) Num. B&B Nodes (e) Relative Times

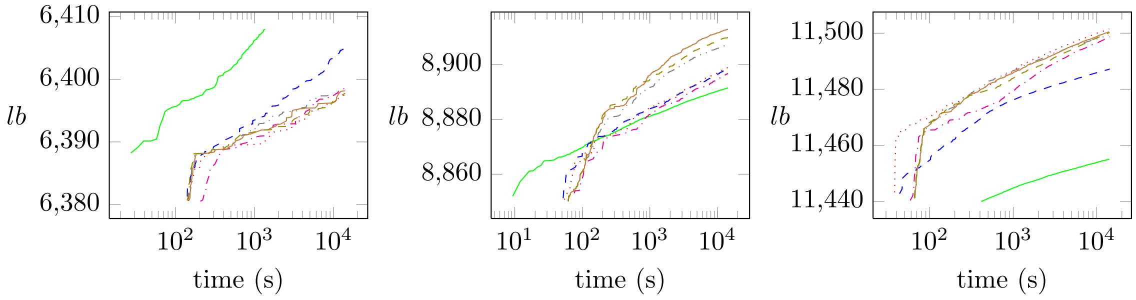
Figure 14: Number of Branch-and-Bound Nodes/Decisions and Relative Times spent in Components



(a) Removed Constraints (b) Num Pricing Problems (c) Pricings Per Node

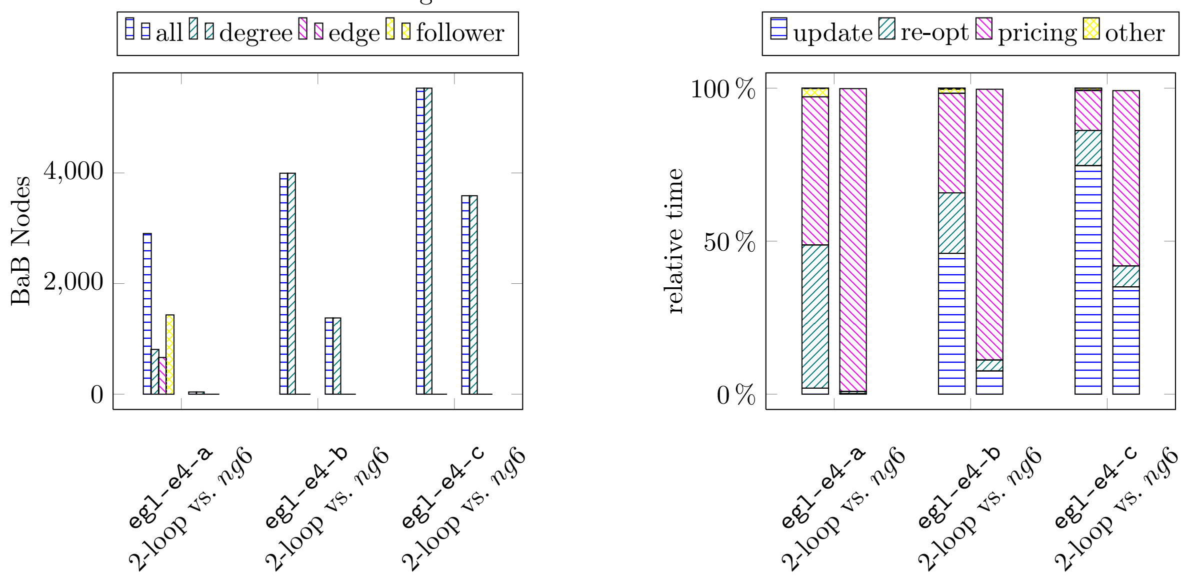
Figure 15: Number of Pricing Problems overall/per Node

Solved to optimality:



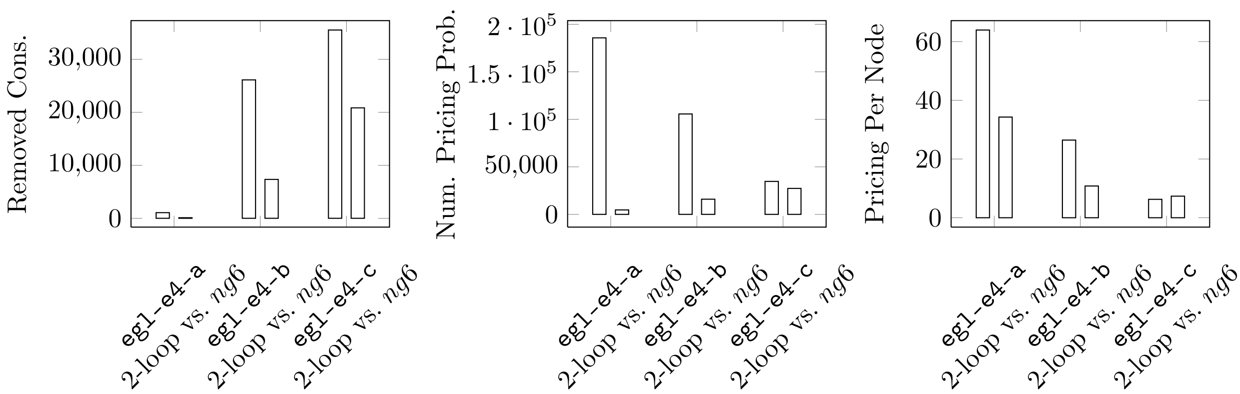
(a) `egl-e4-a` (b) `egl-e4-b` (c) `egl-e4-c`

Figure 16: Lower bounds over Time



(d) Num. B&B Nodes (e) Relative Times

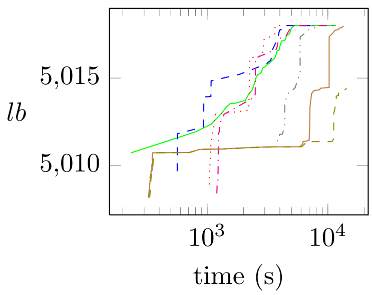
Figure 17: Number of Branch-and-Bound Nodes/Decisions and Relative Times spent in Components



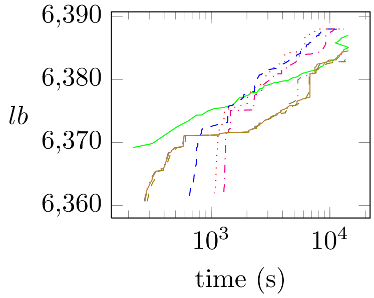
(a) Removed Constraints (b) Num Pricing Problems (c) Pricings Per Node

Figure 18: Number of Pricing Problems overall/per Node

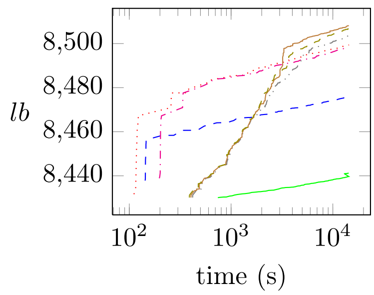
Solved to optimality: egl-s1-a



(a) egl-s1-a

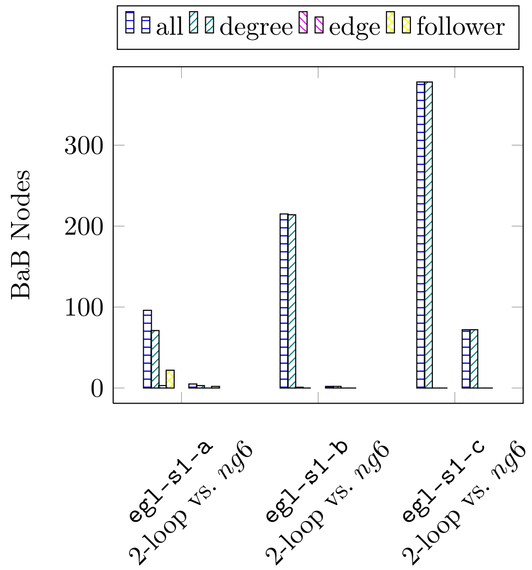


(b) egl-s1-b

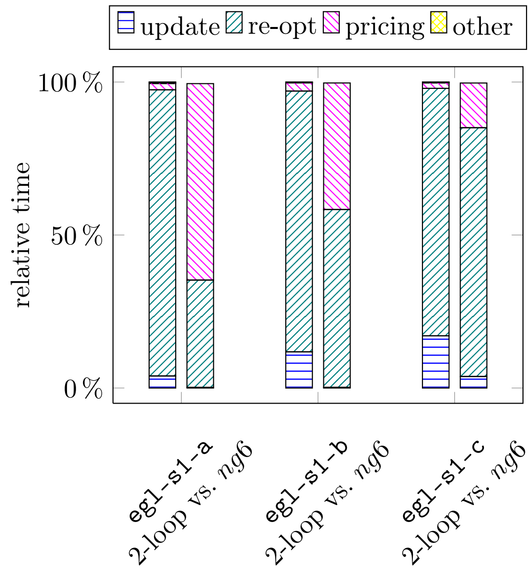


(c) egl-s1-c

Figure 19: Lower bounds over Time

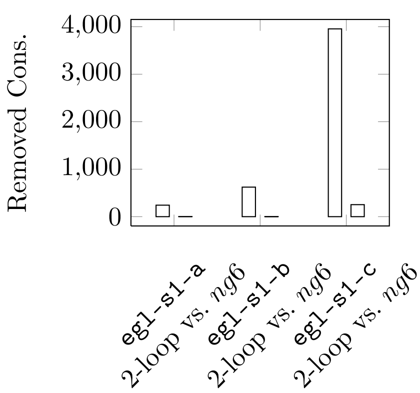


(d) Num. B&B Nodes

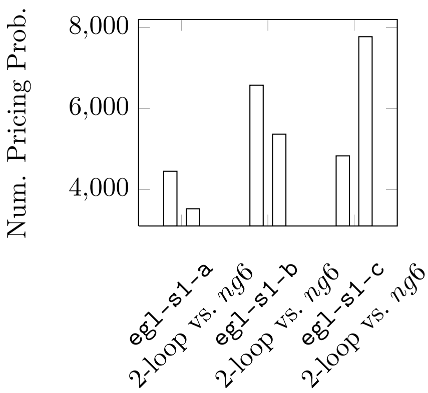


(e) Relative Times

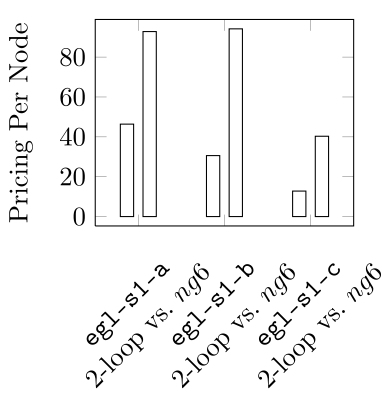
Figure 20: Number of Branch-and-Bound Nodes/Decisions and Relative Times spent in Components



(a) Removed Constraints



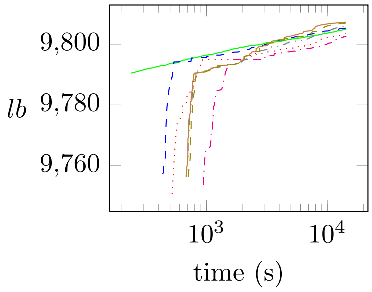
(b) Num Pricing Problems



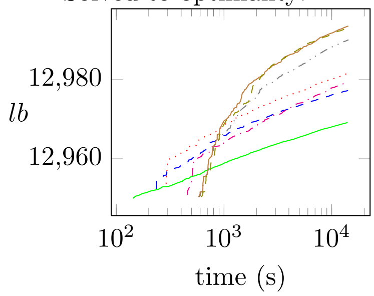
(c) Pricings Per Node

Figure 21: Number of Pricing Problems overall/per Node

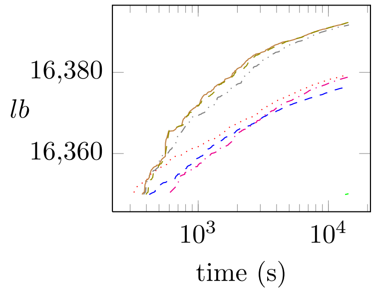
Solved to optimality:



(a) `egl-s2-a`

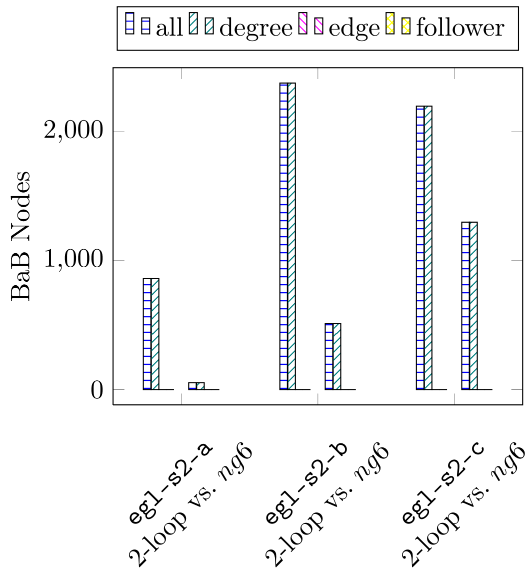


(b) `egl-s2-b`

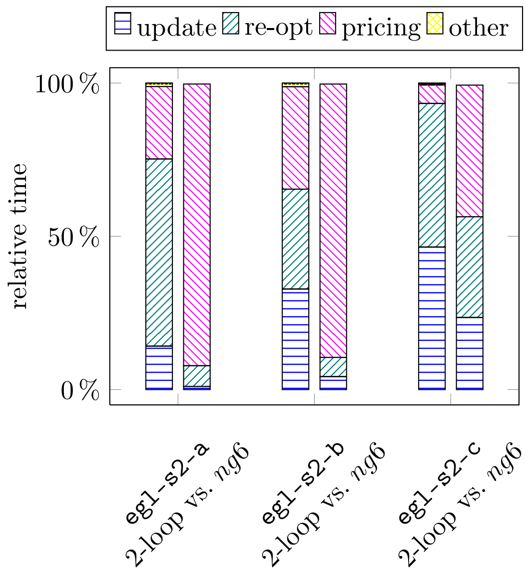


(c) `egl-s2-c`

Figure 22: Lower bounds over Time

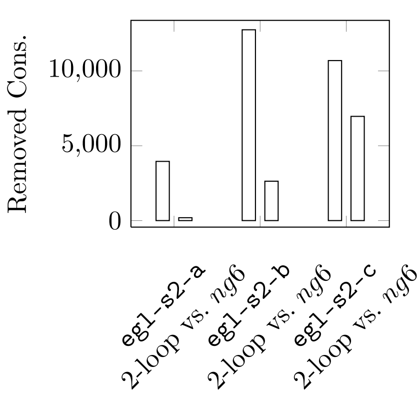


(d) Num. B&B Nodes

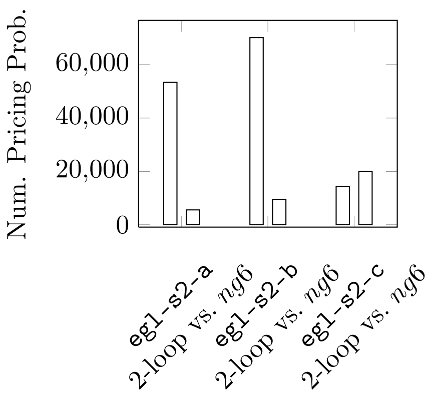


(e) Relative Times

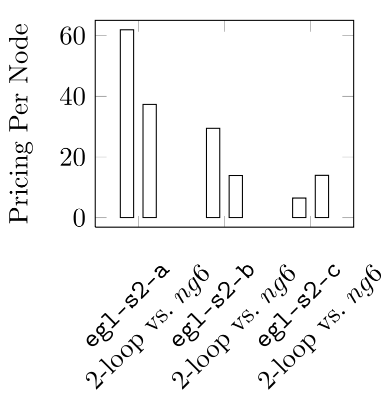
Figure 23: Number of Branch-and-Bound Nodes/Decisions and Relative Times spent in Components



(a) Removed Constraints



(b) Num Pricing Problems



(c) Pricings Per Node

Figure 24: Number of Pricing Problems overall/per Node

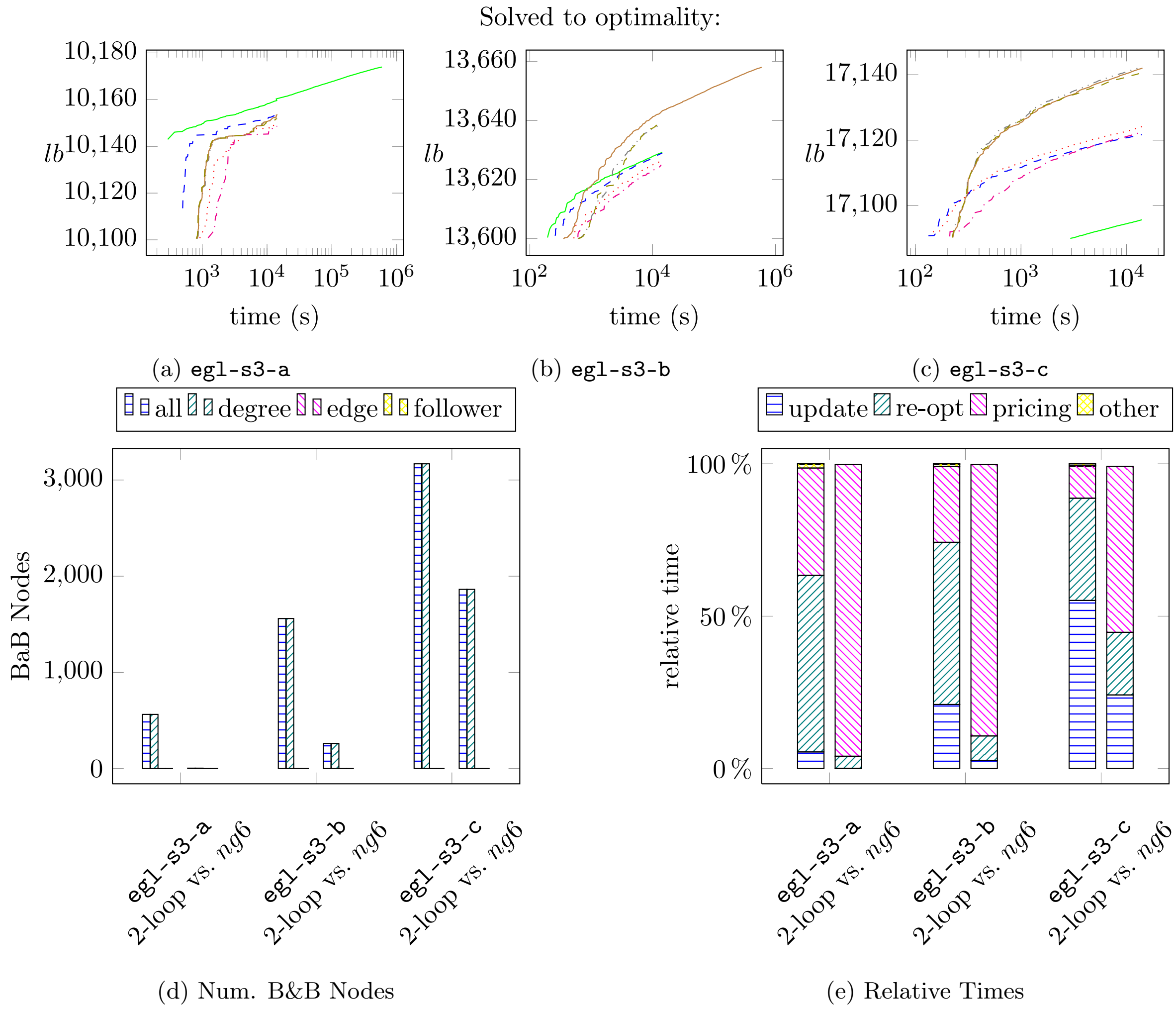


Figure 25: Number of Branch-and-Bound Nodes/Decisions and Relative Times spent in Components

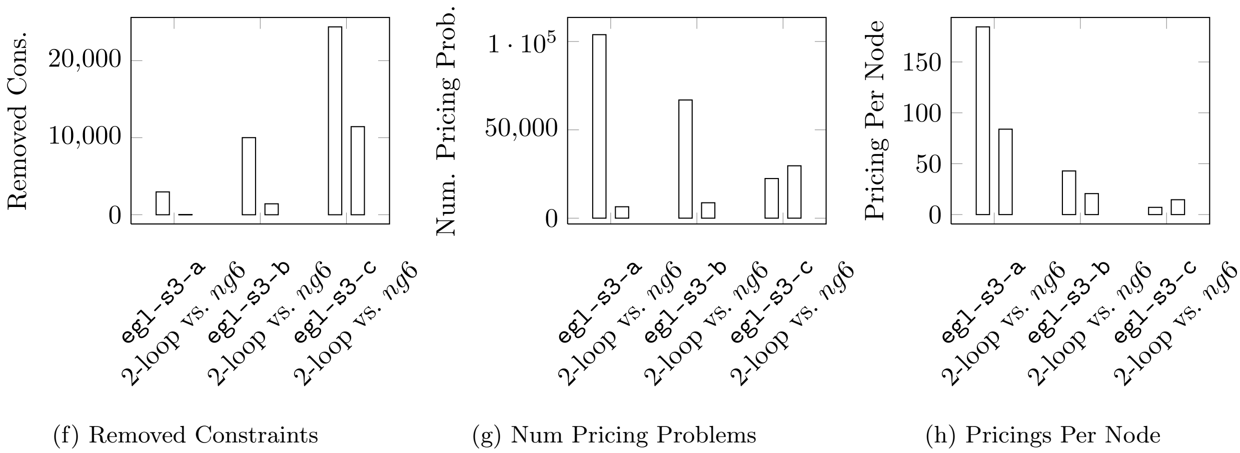
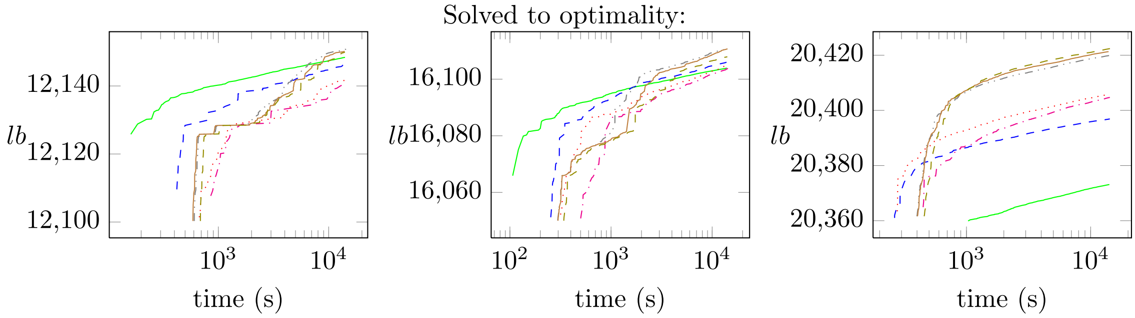
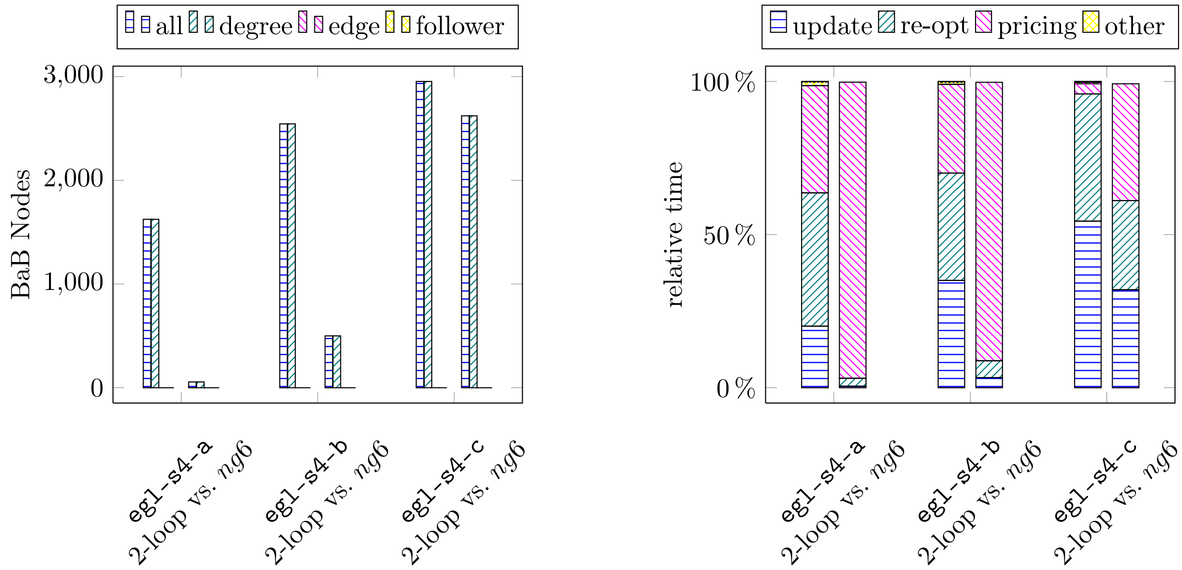


Figure 26: Number of Pricing Problems overall/per Node



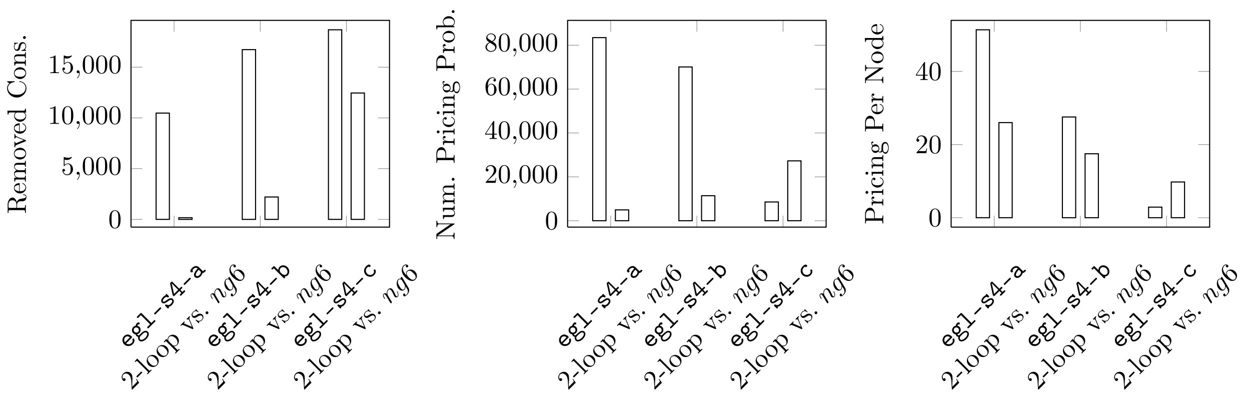
(a) `egl-s4-a` (b) `egl-s4-b` (c) `egl-s4-c`

Figure 27: Lower bounds over Time



(d) Num. B&B Nodes (e) Relative Times

Figure 28: Number of Branch-and-Bound Nodes/Decisions and Relative Times spent in Components



(a) Removed Constraints (b) Num Pricing Problems (c) Pricings Per Node

Figure 29: Number of Pricing Problems overall/per Node