European Journal of Operational Research 238 (2014) 415–426

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

The shortest-path problem with resource constraints with $(k,2)$-loop elimination and its application to the capacitated arc-routing problem

Claudia Bode, Stefan Irnich *

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

ARTICLE INFO

Article history:
Received 15 April 2013
Accepted 4 April 2014
Available online 16 April 2014

Keywords:
Constrained shortest paths
Cycle elimination
Capacitated arc-routing problem

ABSTRACT

In many branch-and-price algorithms, the column generation subproblem consists of computing feasible constrained paths. In the capacitated arc-routing problem (CARP), elementarity constraints concerning the edges to be serviced and additional constraints resulting from the branch-and-bound process together impose two types of loop-elimination constraints. To fulfill the former constraints, it is common practice to rely on a relaxation where loops are allowed. In a k-loop elimination approach all loops of length k and smaller are forbidden. Following Bode and Irnich (2012) for solving the CARP, branching on followers and non-followers is the only known approach to guarantee integer solutions within branch-and-price. However, it comes at the cost of additional task-2-loop elimination constraints. In this paper, we show that a combined $(k,2)$-loop elimination in the shortest-path subproblem can be accomplished in a computationally efficient way. Overall, the improved branch-and-price often allows the computation of tighter lower bounds and integer optimal solutions for several instances from standard benchmark sets.

1. Introduction

In thiAlgorithms paper, we extend the works of Irnich and Villeneuve (2006) and Bode and Irnich (2012). The first paper considered k-cycle elimination for shortest-path problems with resource constraints (SPPRC, Irnich & Desaulniers, 2005, chap. 2). The elementary SPPRC (ESPPRC) is the subproblem of many column-generation formulations of routing problems. Since the ESPPRC is $\mathcal{NP}$-hard in the strong sense (Dror, 1994), early column-generation approaches solved the SPPRC, i.e., the corresponding non-elementary problem, or SPPRC with 2-cycle elimination (see, e.g. Houck, Picard, Queyranne, & Vemuganti, 1980, 1999), which are both relaxations, with the consequence that the lower bounds computed by the column-generation master program often deteriorate. The elimination of k-cycles, i.e., cycles with up to k edges, can be seen as a mean to gradually strengthen the linear relaxation of the column-generation master program while keeping the computational effort acceptable. Both from a practical and a worst-case point of view, k-cycle elimination is computationally attractive because there exist pseudo-polynomial labeling algorithms (Irnich & Villeneuve, 2006). Applied to the vehicle-routing problem with time windows some knowingly hard instances were solved for the first time. Nowadays, it seems that approaches based on solving ESPPRC (e.g. Jepsen, Petersen, Spoorendonk, & Pisinger, 2008) or ng-path relaxations (Baldacci, Mingozzi, & Roberti, 2011a) are superior due to the extremely tight lower bounds produced. However, when routes become very long, solving even very few ESPPRC subproblems can become extremely time consuming (see e.g. Desaulniers, Lessard, & Hadjar, 2008).

We apply loop-elimination for solving the capacitated arc-routing problem (CARP) with the branch-and-price algorithm of Bode and Irnich (2012). The CARP is the basic multiple-vehicle arc-routing problem. It has applications in waste collection, postal delivery, winter services and more (Corberán & Prins, 2010; Dror, 2000). For a general overview on exact algorithms for the CARP we refer to the survey (Belenguer, Benevent, & Irnich, 2014, chap. 9). In the paper (Bode & Irnich, 2012), the first full-fledged branch-and-price algorithm for the CARP was presented. It can be characterized by the idea of exploiting sparsity of the underlying CARP network. The advantage of sparse networks is that new CARP tours can be priced out efficiently using the sparse network only (see also Letchford & Oukil, 2009). Bode and Irnich (2012) discussed that the sparse network however comes at the cost of a more intricate branching and they

* Corresponding author. Tel.: +49 61313922007; fax: +49 61313922097.
E-mail addresses: claudia.bode@uni-mainz.de (C. Bode), irnich@uni-mainz.de (S. Irnich).

http://dx.doi.org/10.1016/j.ejor.2014.04.004
0377-2217/

developed an effective branching scheme based on follower and non-follower constraints. It means that for every two required edges e and e', one determines whether or not these two edges are serviced by the same vehicle and with no other intermediate service in between. Follower constraints, i.e., two edges e and e' have to be serviced consecutively, can be enforced by network modifications so that these constraints preserve the structure of the pricing subproblem. However, non-follower constraints, i.e., two edges e and e' must not be served consecutively, can only be implemented efficiently as a task-2-loop elimination constraint: The trick is to associate the same task to both edges e and e' (see also Irnich & Desaulniers (2005, chap. 2) for a discussion of tasks and task cycles). Hence, this branching scheme seemed limited to the case that an SPPRC subproblem with 2-loop elimination is employed (the term loop refers to cycles w.r.t. services edges; a 2-loop is the repetition of the same service). Moreover, Bode and Irnich (2012) showed that pricing relaxations based on k-loop elimination can produce better column-generation lower bounds. However, a key question remained unclear: How can the branching scheme with branching on followers/non-followers be combined with k-loop elimination for $k \geqslant 3$.

A positive answer will be given here because we can show that a combined $(k,2)$-cycle elimination accomplishes the above problem. More precisely, k-loops w.r.t. tasks associated with services on required edges and 2-loops w.r.t. tasks implied by non-follower constraints can be handled by a labeling approach presented here. Note that the companion paper (Bode & Irnich, in press) studies several alternative relaxations including, e.g., extensions of the afore-mentioned *ng*-path relaxations. Its focus is the empirical comparison of these relaxations, the analysis of the impact of possible acceleration techniques, and the overall comparison of the different branch-and-price algorithms. Please note that the article at hand and the paper (Bode & Irnich, in press) were originally one long technical report (submitted and available online on our website in November 2012). Due to its length the paper was not publishable so that we decided to split it into two parts. We (re)submitted both parts in April 2013 when a new paper by Bartolini, Cordeau, and Laporte (2013) became online available. Therefore, we consider results presented here and in Bartolini et al. (2013) as developed independently; the computational results section however compares also explicitly with their (very good) results.

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

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

The remainder of this paper is structured as follows. Section 2 presents the new labeling algorithm and its theoretical analysis for SPPRC with combined $(k,2)$-loop elimination. Section 3 presents computational results for using the respective relaxations as subproblems in a branch-and-price for the CARP. The paper ends with final conclusions in Section 4.

2. Labeling algorithm for SPPRC with combined $(k,2)$-loop elimination

A general discussion of SPPRC solution approaches including a detailed discussion of dynamic programming labeling algorithms, at the moment the generally best performing methods, can be found in Irnich and Desaulniers (2005, chap. 2). Solution approaches tailored to ESPPRC are presented in Feillet, Dejax, Gendreau, and Gueguen (2004), Boland, Dethridge, and Dumitrescu (2006), Righini and Salani (2006, 2008).

2.1. Basic SPPRC and generic labeling algorithm

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

In standard SPPRC, the problem is to compute a minimum-cost feasible path ending at each destination node $t \in V$. In case of non-decreasing REFs, a path $P' = (P,j)$ is *feasible* (w.r.t. resources) if $r(P') \in [a_{v(P')}, b_{v(P')}]$ holds and the predecessor path P is also feasible. Then, for solving SPPRC, one typically computes, for each node, a set of paths $\{P_1,\ldots,P_q\}$ with $\{r(P_1),\ldots,r(P_q)\}$ forming the Pareto-optimal resource values. The particular importance of non-decreasing REFs is stressed in Desaulniers et al. (1998, chap. 3), Irnich and Desaulniers (2005, chap. 2), Irnich (2008).

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

- Service arcs (i,j) and (j,i) correspond to proving service to a required edge $\{i,j\}$. These arcs consume a positive amount $q_{ij} > 0$ of the vehicle's capacity and have a (reduced) cost $\tilde{c}^{serv}_{ij}$ (not restricted in sign).
- Deadheading arcs (i,j) and (j,i) correspond to traversing an arbitrary edge $\{i,j\}$ without providing service. These arcs consume nothing from the vehicle's capacity, i.e., $q_{ij} = 0$. Even with valid inequalities present in the column-generation master program, their (reduced) cost $\tilde{c}_{ij}$ can be guaranteed to be non-negative (for details see Bode & Irnich, 2012).

Bode and Irnich (2012) explained how the column-generation restricted master program (RMP) provides with its dual solution the reduced costs $\tilde{c}^{serv}_{ij}$ and $\tilde{c}_{ij}$. Summing up, the resources $r(P)$ in the CARP case consumed along a path P are given by $r(P) = (q(P), \tilde{c}(P))$, where $q(P)$ is restricted to integer values $0,1,\ldots,C$ (C is the capacity of the vehicle) and $\tilde{c}(P)$ is the accumulated reduced costs.

The outline of a generic labeling approach for solving SPPRC is presented in Algorithm 1. Herein, each path P is represented by a label $L(P)$, i.e., a data structure that allows the reconstruction of the associated path via labels of the predecessor path, provides additional information such as the resource consumption $r(P)$, and allows to invoke a dominance algorithm. The set $\mathcal{U}$ is the set of unprocessed labels $L(P)$, i.e., paths P that are not extended along all arcs $(v(P), j) \in A$ of the forward star of node $v(P)$. The set $\mathcal{L}$ contains those labels that need to be kept.

Algorithm 1. Generic SPPRC Dynamic Programming Labeling Algorithm

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

Depending on the path selection rule in the path extension step, different label processing procedures result such as label setting and label correcting algorithms. The invocation of a dominance algorithm is optional in the sense that otherwise the algorithm enumerates all feasible paths starting at node s. Dominance is however crucial in the design of *efficient* labeling algorithms, and we devote Section 2.3 for the detailed presentation of this basic component. In general, the intension of the dominance algorithm is to identify those paths that do not need to be extended, i.e., one or several other paths still allow finding (Pareto-) optimal paths. It can be applied at any time in the course of the algorithm and might be delayed to a point where several new paths with identical end node have been generated and stored in $\mathcal{U}$. Any reasonable strategy for invoking the dominance algorithm will optimize the tradeoff between the computational effort and the risk that a path is extended before one finds out that it is dominated.

In the presence of additional path-structural constraints (such as cycle- or loop-elimination constraints discussed in Section 2.2, see also Section 2.2 of Irnich & Desaulniers (2005, chap. 2)), the set $\mathcal{L}$ must generally include additional labels for paths that are not necessarily Pareto-optimal. It this case, a final filtering step is needed to identify a Pareto-optimal subset. Efficient algorithms for that purpose can be found in Bentley (1980).

2.2. Task-loops and loop elimination

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

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

The rationale behind these seeming confusing definitions is that 2-cycle elimination and task-2-loop elimination can be handled with almost identical algorithmic approaches: Dominance rules require that only a best and a second-best path with different last task need to be kept in $\mathcal{L}$ (see Algorihm 1). The dominance rules were first presented by (Houck et al., 1980) for 2-cycle elimination in the node-routing context and by (Benavent, Campos, CorberÃ¡n, & Mota, 1992) for task-2-loop elimination and the CARP.

2.3. Dominance rules in combined (k_1, k_2)-loop elimination

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

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

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

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

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

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

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

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

$$r(P_1),\ldots,r(P_g) \leqslant r(P), \tag{1}$$

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

The fundamental idea for (k_1,k_2)-loop elimination is to efficiently encode the *set of possible extensions* of a path. For this purpose, let $\mathcal{E}(P)$ denote the set of loop-free extensions of the path P. $\mathcal{E}(P)$ solely considers task loops and not resource consumptions. The second condition (2.) above is fulfilled for $P_1,\ldots,P_g$ and P if (1) and

$$\bigcup_{i=1}^{g}\mathcal{E}(P_i) \supseteq \mathcal{E}(P) \tag{2}$$

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

2.3.1. Encoding the possible extensions by self-hole sets

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

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

and

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

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

We are able to express the above condition as

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

and

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

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

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

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

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

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

Equivalently, the self-hole set of P is

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

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

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

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

Now we consider a dominance situation where (1) and (2) are fulfilled for dominating paths $P_1,\ldots,P_g$ and a dominated path P. By de Morgan's law, we get

$$\bigcup_{i=1}^{g}\mathcal{E}(P_i) \supseteq \mathcal{E}(P) \iff \bigcap_{i=1}^{g}H(P_i) \subseteq H(P) \tag{3}$$

so that the condition (2) for loop-free extensions can be equivalently stated with the help of self-hole sets. The point is now that any intersection of the self-hole sets, resulting on the right hand side, can be calculated and represented as a union of set forms again. The following theorem summarizes the result:

Theorem 1. *Let $P_1,P_2,\ldots,P_g$ and P be different paths ending at the same node $v(P_1)=\ldots=v(P_g)=v(P)$ with $r(P_1),\ldots,r(P_g) \leqslant r(P)$, and (3) is fulfilled.*

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

Example 2. (Example 1 continued) Let P' be another path with $\mathcal{T}^E(P')=(a,d)$ (just two edges serviced along P') and $\mathcal{T}^B(P')=(\beta)$. The self-hole set of P' is

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

Then, the intersection of the self-hole sets is

$H(P) \cap H(P') = (a,\cdot,\cdot)(\alpha) \cup (\cdot,a,\cdot)(\alpha) \cup (d,\cdot,\cdot)(\alpha) \cup (\cdot,d,\cdot)(\alpha) \cup (\cdot,\cdot,d)(\alpha) \cup$
$(a,\cdot,\cdot)(\beta) \cup (b,\cdot,\cdot)(\beta) \cup (\cdot,b,\cdot)(\beta) \cup (c,\cdot,\cdot)(\beta) \cup (\cdot,c,\cdot)(\beta) \cup (\cdot,\cdot,c)(\beta) \cup$
$(a,d,\cdot)(\cdot) \cup (a,\cdot,d)(\cdot) \cup (b,a,\cdot)(\cdot) \cup (b,d,\cdot)(\cdot) \cup (b,\cdot,d)(\cdot) \cup (a,b,\cdot)(\cdot) \cup (d,b,\cdot)(\cdot) \cup$
$(\cdot,b,d)(\cdot) \cup (c,a,\cdot)(\cdot) \cup (c,d,\cdot)(\cdot) \cup (c,\cdot,d)(\cdot) \cup (a,c,\cdot)(\cdot) \cup (d,c,\cdot)(\cdot) \cup (\cdot,c,d)(\cdot) \cup$
$(a,\cdot,c)(\cdot) \cup (\cdot,a,c)(\cdot) \cup (d,\cdot,c)(\cdot) \cup (\cdot,d,c)(\cdot)$

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

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

With the above definition of the intersection of two set form, we are able to formally define the intersection of two hole sets H_1 and H_2. Algorithm 2 is similarly stated in Irnich and Villeneuve (2006, p. 398). Note that the first loop is included for the purpose of accelerating the subsequent steps, i.e., to produce fewer set forms s and t in the preliminary result set H having s included in t or vice versa. Such included set forms are eliminated in the last loop. Overall complexity of Algorithm 2 $\mathcal{O}$(kpq), where p and q is the number of set forms in the respective hole set.

Algorithm 2. Intersection of hole sets

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

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

2.3.2. Upper bound on the number of set forms in an intersection of self-hole sets

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

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

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

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

Example 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,

$$H(P_1) = (\cdot,\cdot)(\alpha) \cup (a,\cdot)(\cdot) \cup (b,\cdot)(\cdot) \cup (\cdot,b)(\cdot)$$
$$H(P_2) = (\cdot,\cdot)(\beta) \cup (c,\cdot)(\cdot) \cup (d,\cdot)(\cdot) \cup (\cdot,d)(\cdot)$$
$$H(P_3) = (\cdot,\cdot)(\gamma) \cup (e,\cdot)(\cdot) \cup (f,\cdot)(\cdot) \cup (\cdot,f)(\cdot)$$

giving rise to

$$H(P_1) \cap H(P_2) \cap H(P_3) = (a,d)(\gamma) \cup (b,d)(\gamma) \cup (c,b)(\gamma) \cup (d,b)(\gamma) \cup (c,f)(\alpha) \cup (d,f)(\alpha) \cup (e,d)(\alpha) \cup (f,d)(\alpha) \cup (a,f)(\beta) \cup (b,f)(\beta) \cup (e,b)(\beta) \cup (f,b)(\beta).$$

These are twelve set forms which is the maximum number $(k_1-1)!^2 \cdot (k_2-1)!^2 \cdot \left(\genfrac{}{}{0pt}{}{k_1-1+k_2-1}{k_1-1}\right) = (3-1)!^2 \cdot (2-1)!^2 \cdot \binom{(3-1)+(2-1)}{3-1} = 4 \cdot 1 \cdot 3 = 12$.

2.3.3. Upper bound on the number of paths with identical state

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

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

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

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

holds. Irnich and Villeneuve (2006) have shown that a path P_t can be discarded if $I_t = I_{t-1}$ holds. The reason is that $I_t = I_{t-1}$ implies $H(P_1) \cap \ldots \cap H(P_{t-1}) \subseteq H(P_t)$ so that conditions (3) hold. Therefore, the *maximum length of a properly decreasing chain of intersections of self-hole sets* is a bound on the maximum number of labels to consider with identical state.

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

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

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

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

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

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

For the labeling algorithm, it means that the running-hole set is stored within the label for bookkeeping already identified dominance relations. Whenever one or several new labels are created (by the path extension step, see Section 2.1), they are compared for dominance with the existing (old) labels that are already present at the same node. If a new label $L(P)$ dominates an existing label $L(P')$, i.e., $L(P) \prec_{dom} L(P')$, the running-hole set of P' is replaced by $H^{run}(P') := H^{run}(P') \cap H(P)$. Conversely, if an old label $L(P')$ dominates a new label $L(P)$, i.e., $L(P') \prec_{dom} L(P)$, the running-hole set of P is replaced by $H^{run}(P) := H^{run}(P) \cap H(P')$. Additional algorithmic tricks (to improve on the average run time) for storing the intersection and checking the above condition were discussed in Irnich and Villeneuve (2006, p. 399).

2.4. Specifics and complexity of the CARP pricing problem

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

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

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

1. In an outer loop over possible values $q = 0, 1, 2, \ldots, C$ of the load resource paths P with $q(P) = q$ are extended.
2. The extension is split into two parts, the extension along all deadheading arcs first and the extensions along all service arcs second.
3. The first extension (deadheading) produces only labels with identical load q. All extensions can be handled together using a Dijkstra-type of labeling. Note that for the CARP, the only relevant resource is cost whenever load is fixed. By pre-assigning minimum-cost labels at all nodes, the time complexity $D(n,m)$ can be reached.
4. The latter extensions (service) produce not more than $\mathcal{O}(2m)$ new labels $L(P)$, all with load $q(P) > q$.
5. The overall complexity of all extensions is therefore dominated by the complexity of the Dijkstra algorithm. Taking the outer loop over all load values into account implies an overall complexity of $\mathcal{O}(CD(n,m))$.

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

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

We have the following final result:

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

3. Computational results

This section reports computational results of the branch-and-price algorithm for the CARP first presented in Bode and Irnich (2012) when $(k,2)$-loop free relaxations for $k \in \{2,3,4\}$ are used. We quantify the impact of the different $(k,2)$-loop free relaxations on the computation time and the overall best lower bound achieved at the end of the branch-and-price. The branching scheme presented in Bode and Irnich (2012) consists of three levels of branching decisions: First branching on non-even node degrees, and second branching on edges with fractional edge flow. Both decisions have no impact on the structure of the pricing problem. The third decision is branching on follower information, whenever the information if two edges are serviced consecutive is fractional. This branching rule, however, modifies the network of the underlying graph of the pricing problem. In particular, it requires a second task set to be handled in the SPPRC labeling algorithm that solves the pricing subproblem.

For the branch-and-price, no initial upper bound is given and the node selection rule in branch-and-bound is best-bound first. Note that the same formulation of the (restricted) master problem is used as in Bode and Irnich (2012), while for the pricing subproblem the following modifications are made: Whenever possible, the simple k-loop elimination pricing is used. If, however, any non-follower constraints is active, the simple k-loop elimination pricing is replaced by $(k,2)$-loop elimination pricing. Moreover, we use standard heuristic pricing procedures and acceleration techniques for exact pricing as presented in the companion paper (Bode & Irnich, in press). The two acceleration techniques applied are bounding with the 2-loop elimination relaxation and bi-direction labeling; for details we refer to Mingozzi, Bianco, and Ricciardelli (1997), Baldacci, Mingozzi, and Roberti (2011b, 2011a), Righini and Salani (2006).

The computational study uses two standard benchmark sets from the literature: The first benchmark set `egl` was introduced by Eglese and Li (1992) and can be downloaded from http://www.uv.es/~belengue/carp/. This set consists of 24 instances based on the road network of Cumbria. The first 12 instances have 77 nodes and 98 edges, whereas the remaining 12 instances are larger and have 140 nodes and 190 edges. Instances with the same graph size further differ in the number of required edges and the

Table 1
Integer results for `egl` instances.

Instance	ub_{best} or opt	2-loop				3-loop				4-loop				lb^{own}_{best}	lb^{known}_{best}	lb^{BCL}_{best}
		lb^{root}	lb^{tree}	Time	B&B nodes	lb^{root}	lb^{tree}	Time	B&B nodes	lb^{root}	lb^{tree}	Time	B&B nodes			
e1-a	3548	3545	OPT	176	23	3546	OPT	516	19	3546	OPT	589	11	3548	3548	3548
e1-b	4498	4464	OPT	1343	659	4465	OPT	1512	374	4467	OPT	3827	311	4498	4498	4498
e1-c	5595	5523	5545	4 h	3326	5528	5551	4 h	3057	5532	5555	4 h	2271	5555	5595	5583
e2-a	5018	4996	OPT	892	340	4996	OPT	2955	227	4999	OPT	6345	102	5018	5018	5018
e2-b	6317	6273	6301	4 h	3293	6280	6301	4 h	1376	6283	**6306**	4 h	779	**6306**	6301	**6306**
e2-c	8335	8202	8242	4 h	5601	8227	8269	4 h	5104	8263	8303	4 h	3907	8303	8335	8298
e3-a	5898	5894	OPT	106	26	5895	OPT	562	22	5895	OPT	761	19	5898	5898	5898
e3-b	7775	7684	7730	4 h	3431	7699	**7735**	4 h	1519	7704	7732	4 h	607	**7735**	7728	7728
e3-c	10,292	10,145	10,191	4 h	5396	10,176	10,220	4 h	4490	10,182	10,226	4 h	3479	10,226	10,244	10,225
e4-a	6444	6389	6408	4 h	3361	6389	6405	4 h	280	6389	6399	4 h	66	6408	6408	6408
e4-b	8961	8852	8892	4 h	4392	8862	8899	4 h	1627	8865	8900	4 h	1096	8900	8935	8919
e4-c	**11,529**	11,411	11,456	4 h	5924	11,438	11,488	4 h	5045	11,463	**11,502**	4 h	4316	**11,502**	11,493	11,491
s1-a	5018	5011	OPT	11,683	97	5012	OPT	7762	43	5013	OPT	4312	14	5018	5018	5018
s1-b	6388	6370	6386	4 h	210	6373	OPT	13,072	130	6376	OPT	12,250	49	6388	6388	6388
s1-c	8518	8418	8440	4 h	354	8457	8476	4 h	314	8468	8500	4 h	310	8500	8518	8518
s2-a	9884	9791	9805	4 h	847	9795	9806	4 h	257	9795	9804	4 h	114	9806	9825	**9838**
s2-b	13,100	12,949	12,970	4 h	2320	12,955	12,978	4 h	1548	12,960	12,982	4 h	1054	12,982	13,017	12,999
s2-c	16,425	16,314	16,351	4 h	2041	16,332	16,377	4 h	2189	16,338	16,380	4 h	1949	16,380	16,425	16,395
s3-a	10,220	10,144	10,160	4 h	547	10,145	10,154	4 h	66	10,145	10,150	4 h	13	10,160	10,160	**10,161**
s3-b	13,682	13,598	13,630	4 h	1515	13,604	13,629	4 h	800	13,605	13,627	4 h	274	13,630	13,648	13,637
s3-c	17,188	17,058	17,096	4 h	3102	17,089	17,122	4 h	2505	17,090	17,125	4 h	2217	17,125	17,188	17,136
s4-a	12,268	12,126	12,149	4 h	1617	12,129	12,147	4 h	271	12,129	12,142	4 h	62	12,149	12,149	**12,159**
s4-b	16,283	16,066	16,104	4 h	2366	16,071	16,106	4 h	1449	16,073	16,105	4 h	473	16,106	16,105	**16,114**
s4-c	20,481	20,340	20,374	4 h	2797	20,362	20,397	4 h	3556	20,375	20,406	4 h	3157	20,406	20,430	20,405
Num lb^{best}_{own}			9				9				17					
Num opt			5				6				6					
Avg %gap		0.84	0.54			0.74	0.46			0.68	0.43					

Table 2
Integer results for `bmcv` instances, subset `C`.

Instance	ub_{best} or opt	2-loop				3-loop				4-loop				New opt or lb^{own}_{best}	lb^{known}_{best}	New opt or lb^{BCL}_{best}
		lb^{root}	lb^{tree}	Time	B&B nodes	lb^{root}	lb^{tree}	Time	B&B nodes	lb^{root}	lb^{tree}	Time	B&B nodes			
C01	4150	4086	4144	4 h	2834	4087	4140	4 h	1435	4090	4140	4 h	862	4144	4105	**4150**
C02	3135	3135	OPT	6	5	3135	OPT	28	10	3135	OPT	60	6	3135	3135	3135
C03	2575	2529	OPT	6494	3746	2542	OPT	171	115	2546	OPT	252	120	2575	2575	2575
C04	3510	3474	**OPT**	2163	1317	3474	**OPT**	2678	773	3474	**OPT**	4403	532	**3510**	3480	**3510**
C05	5365	5320	OPT	59	81	5321	OPT	169	103	5323	OPT	272	81	5365	5365	5365
C06	2535	2508	OPT	163	310	2509	OPT	314	157	2510	OPT	196	53	2535	2535	2535
C07	4075	4019	OPT	278	456	4019	OPT	561	465	4019	OPT	724	401	4075	4075	4075
C08	4090	4025	OPT	751	474	4026	OPT	778	347	4029	OPT	634	178	4090	4090	4090
C09	5260	5219	5244	4 h	2989	5220	5242	4 h	1922	5220	5242	4 h	1454	5244	5235	**5255**
C10	4700	4606	OPT	1363	1604	4614	OPT	1344	1087	4617	OPT	1493	810	4700	4700	4700
C11	4635	4571	4608	4 h	2564	4571	4608	4 h	1332	4572	4607	4 h	612	4608	4585	**4620**
C12	4240	4175	4234	4 h	4356	4176	4231	4 h	2072	4176	4226	4 h	1211	4234	4210	**4240**
C13	2955	2907	OPT	288	612	2909	OPT	456	411	2910	OPT	589	356	2955	2955	2955
C14	4030	3982	4010	4 h	5189	3986	4021	4 h	2810	3986	4024	4 h	1610	4024	4030	4030
C15	4940	4887	4918	4 h	1620	4888	4915	4 h	977	4891	4916	4 h	670	4918	4915	**4935**
C16	1475	1470	OPT	6	13	1470	OPT	42	13	1470	OPT	196	13	1475	1475	1475
C17	3555	3547	OPT	17	26	3548	OPT	20	16	3550	OPT	23	11	3555	3555	3555
C18	5620	5557	5570	4 h	1958	5557	5568	4 h	691	5557	5563	4 h	292	5570	5580	**5590**
C19	3115	3074	**OPT**	2311	1324	3076	**OPT**	3204	902	3076	**OPT**	6706	978	**3115**	3100	**3115**
C20	2120	2120	OPT	12	20	2120	OPT	136	26	2120	OPT	392	9	2120	2120	2120
C21	3970	3956	**OPT**	9947	3113	3956	**OPT**	1007	130	3956	**OPT**	3284	117	**3970**	3960	**3970**
C22	2245	2245	OPT	32	16	2245	OPT	60	11	2245	OPT	256	13	2245	2245	2245
C23	4085	4032	**4073**	4 h	2752	4032	4072	4 h	1078	4032	4069	4 h	400	**4073**[a]	4035	**4075**
C24	3400	3377	**OPT**	1358	454	3380	**OPT**	975	124	3380	**OPT**	2325	130	**3400**	3385	**3400**
C25	2310	2310	OPT	6	10	2310	OPT	10	9	2310	OPT	20	4	2310	2310	2310
Num lb^{best}_{own}			24				18				18					
Num opt			17				17				17					
Avg time				5619				5086				5481				
Avg %gap		0.97	0.13			0.92	0.13			0.89	0.14					

[a] Since all routing costs are multiples of 5, the lower bound of 4075 results.

Table 3
Integer results for `bmcv` instances, subset `E`.

Instance	ub_{best} or $\underline{opt}$	2-loop				3-loop				4-loop				New $\underline{opt}$ or lb^{own}_{best}	lb^{known}_{best}	New $\underline{opt}$ or lb^{BCL}_{best}
		lb^{root}	lb^{tree}	Time	*B&B* nodes	lb^{root}	lb^{tree}	Time	*B&B* nodes	lb^{root}	lb^{tree}	Time	*B&B* nodes			
E01	4910	4857	**4898**	4 h	3269	4857	4896	4 h	2064	4859	4896	4 h	1517	**4898**	4885	4890
E02	$\underline{3990}$	3960	3971	4 h	4363	3960	3985	4 h	1835	3966	OPT	862	126	3990	3990	3990
E03	$\underline{2015}$	2015	OPT	3	1	2015	OPT	19	3	2015	OPT	23	2	2015	2015	2015
E04	$\underline{4155}$	4218	OPT	1593	914	4128	OPT	2584	707	4131	OPT	2789	449	4155	4155	4155
E05	$\underline{4585}$	4562	OPT	506	448	4568	OPT	128	61	4573	OPT	68	17	4585	4585	4585
E06	$\underline{2055}$	2055	OPT	5	7	2055	OPT	25	13	2055	OPT	54	10	2055	2055	2055
E07	$\underline{4155}$	4068	4137	4 h	4508	4073	4149	4 h	4079	4079	OPT	6684	2053	4155	4155	4155
E08	$\underline{4710}$	4671	OPT	208	131	4674	OPT	160	55	4680	OPT	198	53	4710	4710	4710
E09	5820	5771	5802	4 h	1028	5772	5800	4 h	935	5772	5798	4 h	642	5802	5780	**5810**
E10	$\underline{3605}$	3605	OPT	9	9	3605	OPT	21	6	3605	OPT	48	5	3605	3605	3605
E11	4650	4630	**4650**	4 h	2218	4630	**OPT**	4244	279	4630	**4650**	4 h	363	$\underline{\mathbf{4650}}$	4640	$\underline{\mathbf{4650}}$
E12	$\underline{4180}$	4109	4167	4 h	3623	4111	4169	4 h	2435	4112	4170	4 h	1653	4170	4180	4180
E13	$\underline{3345}$	3309	OPT	155	220	3310	OPT	264	240	3311	OPT	437	249	3345	3345	3345
E14	$\underline{4115}$	4091	4108	4 h	5195	4091	OPT	1996	1453	4091	OPT	2089	1052	4115	4115	4115
E15	4205	4182	4199	4 h	1819	4182	4196	4 h	712	4182	4194	4 h	292	4199	4190	$\underline{\mathbf{4205}}$
E16	3775	3747	**OPT**	1287	793	3750	**OPT**	246	103	3751	**OPT**	380	82	$\underline{\mathbf{3775}}$	3755	$\underline{\mathbf{3775}}$
E17	$\underline{2740}$	2740	OPT	4	3	2740	OPT	8	2	2740	OPT	10	2	2740	2740	2740
E18	3835	3825	3825	4 h	1245	3825	3825	4 h	446	3825	3825	4 h	146	3825	3825	3825
E19	3235	3204	**OPT**	7855	993	3205	**OPT**	5905	591	3205	**OPT**	13,935	639	$\underline{\mathbf{3235}}$	3225	$\underline{\mathbf{3235}}$
E20	2825	2789	2815	4 h	5261	2792	2820	4 h	2175	2793	**OPT**	3112	464	$\underline{\mathbf{2825}}$	2805	$\underline{\mathbf{2825}}$
E21	$\underline{3730}$	3725	3730	4 h	5396	3728	3730	4 h	1434	3728	3730	4 h	263	3730	3730	3730
E22	$\underline{2470}$	2461	OPT	32	24	2466	OPT	74	33	2466	OPT	74	17	2470	2470	2470
E23	3710	3684	3704	4 h	548	3686	3703	4 h	385	3687	3699	4 h	223	3704	3690	$\underline{\mathbf{3710}}$
E24	4020	3992	**OPT**	9489	2123	3993	**4020**	4 h	1933	3997	**OPT**	13,135	1130	$\underline{\mathbf{4020}}$	4005	$\underline{\mathbf{4020}}$
E25	$\underline{1615}$	1615	OPT	1	4	1615	OPT	3	1	1615	OPT	2	1	1615	1615	1615
Num lb^{best}_{own}			20				17				21					
Num opt			14				15				18					
Avg time				7758				6963				6364				
Avg %gap		0.65	0.12			0.61	0.08			0.57	0.07					

vehicle capacity. The second benchmark set `bmcv` consisting of 100 instances is obtained from the road network of Flanders, Belgium (Beullens, Muyldermans, Cattrysse, & van Oudheusden, 2003). These instances range from 26 to 97 nodes and 35 to 142 edges, where only a subset of the edges is required. The instances were kindly provided by Muyldermans (2012) and the transformed instances into the standard format can be downloaded from http://logistik.bwl.uni-mainz.de/Dateien/bmcv.zip. These instances comprise four subsets, where the underlying graph for individual instances of subset `C` and `E` is identical, but the vehicle capacity is 300 for the `C` set and 600 for the `E` set. The same holds for the subsets of instances named `D` and `F`.

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

To shorten the notation, we will skip the second entry in $(k, 2)$ so that, in the following, k-loop is a shortcut for $(k, 2)$-loop-free. Since a comprehensive study of linear relaxation results for k-loop elimination with and without activated acceleration techniques are presented in Bode and Irnich (2012), Bode and Irnich (in press), this section focuses on integer results obtained when the branch-and-bound ends (either with an optimal solution or when the given time limit is reached). Tables 1–5 present the integer results for the `egl` and `bmcv` instances. The header entries in all tables have the following meaning:

instance	name of the instance (for `egl` instances the prefix `egl-` is omitted for the sake of brevity)
ub_{best} or $\underline{opt}$	the best known upper bound (not underlined) or the optimum (underlined) reported in Beullens et al. (2003) or Bartolini, Cordeau, and Laporte (2012)
lb^{root}	lower bound that results from solving the linear relaxation
lb^{tree}	lower bound provided by the branch-and-price algorithm within the time limit of four hours; (rounded up to the next integer) 'OPT' indicates that the instance is solved to proven optimality within four hours if the value of lb^{tree} matches the best known upper bound the gap was closed, but no integer optimal solution was computed within the time limit
time	computation time in seconds; if the time limit is reached it is indicated by 4 h
B&B nodes	report the number of solved branch-and-bound nodes
lb^{best}_{own} or new $\underline{opt}$	best lower bound over all relaxations tested here; underlined if optimality gap is closed
lb^{best}_{known}	best lower bounds round to a multiple of five reported in Beullens et al. (2003) or Bartolini et al. (2012)
lb^{best}_{BCL} or new $\underline{opt}$	best lower bounds reported in Bartolini et al. (2013); underlined if optimality gap is closed

The following additional information is given for the respective relaxation:

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

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

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

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

Overall, we are able to obtain 19 new best lower bounds out of 33 previously unsolved `bmcv` instances (5 for subset `C`, 4 for subset `D`, 6 for subset `E` and 4 for subset `F`). Thereof, 15 instances (`C04`, `C19`, `C21`, `C24`, `D08`, `D14`, `D19`, `E11`, `E16`, `E19`, `E20`, `E24`, `F04`, `F08` and `F12`) are solved to optimality for the first time. Bartolini et al. (2012) mentioned that the objective value is always a multiple of five. Using this fact, optimality can be proven also for instances `D23` and `F23`.

In comparison to the recent results presented by Bartolini et al. (2013) (developed in parallel to our algorithm), their approach produces some impressive results: They improve five lower bounds for the `egl` instances where we improve three. Interestingly, the algorithms are complementary, since improvements of both approaches are made on seven different instances. For the `bmcv` instances, they improve the lower bounds for 11 + 8+5 + 5 = 29 instances, where we improve 4 + 6+4 + 4 = 18. Moreover, they

Table 4
Integer results for `bmcv` instances, subset `D`.

Instance	ub_{best} or $\underline{opt}$	2-loop				3-loop				4-loop				New $\underline{opt}$ or lb^{own}_{best}	lb^{known}_{best}	New $\underline{opt}$ or lb^{BCL}_{best}
		lb^{root}	lb^{tree}	Time	$B\&B$ nodes	lb^{root}	lb^{tree}	Time	$B\&B$ nodes	lb^{root}	lb^{tree}	Time	$B\&B$ nodes			
D01	$\underline{3215}$	3215	OPT	50	13	3215	OPT	992	17	3215	OPT	4117	16	3215	3215	3215
D02	$\underline{2520}$	2520	OPT	26	22	2520	OPT	86	14	2520	OPT	286	15	2520	2520	2520
D03	$\underline{2065}$	2065	OPT	43	15	2065	OPT	172	9	2065	OPT	1472	9	2065	2065	2065
D04	$\underline{2785}$	2785	OPT	105	33	2785	OPT	600	33	2785	OPT	9022	26	2785	2785	2785
D05	$\underline{3935}$	3935	OPT	24	15	3935	OPT	145	19	3935	OPT	166	17	3935	3935	2935
D06	$\underline{2125}$	2125	OPT	20	18	2125	OPT	97	5	2125	OPT	1615	15	2125	2125	2125
D07	$\underline{3115}$	3028	3108	4 h	3069	3032	3102	4 h	893	3035	3098	4 h	403	3108	3115	3115
D08	3045	2982	**OPT**	3730	736	2982	3041	4 h	411	2982	3027	4 h	113	**$\underline{3045}$**	2995	**$\underline{3045}$**
D09	$\underline{4120}$	4120	OPT	106	36	4120	OPT	929	47	4120	OPT	1654	36	4120	4120	4120
D10	$\underline{3340}$	3332	OPT	33	21	3332	OPT	195	23	3332	OPT	493	17	3340	3340	3340
D11	$\underline{3745}$	3745	3745	4 h	1061	3745	OPT	1945	21	3745	OPT	11,009	28	3745	3745	3745
D12	$\underline{3310}$	3310	OPT	123	17	3310	OPT	539	21	3310	OPT	198	4	3310	3310	3310
D13	$\underline{2535}$	2535	OPT	997	741	2535	OPT	61	15	2535	OPT	605	15	2535	2535	2535
D14	3280	3272	**3280**	4 h	3513	3272	**OPT**	564	35	3272	**OPT**	1804	30	**$\underline{3280}$**	3275	**$\underline{3280}$**
D15	$\underline{3990}$	3990	OPT	602	41	3990	OPT	3347	17	–	–	–	–	3990	3990	3990
D16	$\underline{1060}$	1060	OPT	7	7	1060	OPT	66	8	1060	OPT	677	10	1060	1060	1060
D17	$\underline{2620}$	2620	OPT	11	18	2620	OPT	31	14	2620	OPT	42	8	2620	2620	2620
D18	$\underline{4165}$	4165	OPT	2951	48	–	–	–	–	4165	4165	4 h	2	4165	4165	4165
D19	2400	2372	**OPT**	552	225	2373	**OPT**	3174	186	2373	**OPT**	13,090	195	**$\underline{2400}$**	2395	**$\underline{2400}$**
D20	$\underline{1870}$	1870	OPT	15	22	1870	OPT	149	21	1870	OPT	2004	20	1870	1870	1870
D21	3050	2967	3005	4 h	2615	2969	2988	4 h	261	2969	2982	4 h	75	3005	2985	**3015**
D22	$\underline{1865}$	1865	OPT	36	15	1865	OPT	251	11	1865	OPT	3200	15	1865	1865	1865
D23	3130	3111	**3126**	4 h	341	3111	3114	4 h	13	3111	3111	4 h	1	**3126**[a]	3115	3125
D24	2710	2666	2704	4 h	884	2666	2691	4 h	132	2667	2679	4 h	45	2704	2680	**$\underline{2710}$**
D25	$\underline{1815}$	1815	OPT	10	13	1815	OPT	25	4	1815	OPT	155	8	1815	1815	1815
Num lb^{best}_{own}			25				20				20					
Num opt			19				19				18					
Avg time				3834				4567				6673				
Avg %gap		0.46	0.08			0.45	0.15			0.44	0.20					

[a] Since all routing costs are multiples of 5, this proves optimality $opt = 3130$.

Table 5
Integer results for `bmcv` instances, subset `F`.

Instance	ub_{best} or opt	2-loop				3-loop				4-loop				New opt or lb^{own}_{best}	lb^{known}_{best}	New opt or lb^{BCL}_{best}
		lb^{root}	lb^{tree}	Time	B&B nodes	lb^{root}	lb^{tree}	Time	B&B nodes	lb^{root}	lb^{tree}	Time	B&B nodes			
F01	4040	4040	OPT	10,942	1103	4040	OPT	988	26	4040	OPT	2170	27	4040	4040	4040
F02	3300	3300	OPT	40	27	3300	OPT	166	13	3300	OPT	1957	77	3300	3300	3300
F03	1665	1665	OPT	14	17	1665	OPT	124	14	1665	OPT	507	11	1665	1665	1665
F04	3485	3476	**OPT**	198	48	3476	**OPT**	6933	160	3476	**OPT**	9654	39	**3485**	3480	**3485**
F05	3605	3605	OPT	61	25	3605	OPT	220	19	3605	OPT	1023	27	3605	3605	3605
F06	1875	1875	OPT	16	18	1875	OPT	73	19	1875	OPT	350	13	1875	1875	1875
F07	3335	3335	OPT	46	15	3335	OPT	190	15	3335	OPT	226	11	3335	3335	3335
F08	3705	3690	**OPT**	174	62	3690	**OPT**	531	40	3692	**OPT**	1927	46	**3705**	3695	**3705**
F09	4730	4730	OPT	526	38	4730	OPT	1533	34	4730	4730	4 h	42	4730	4730	4730
F10	2925	2925	OPT	10	13	2925	OPT	58	11	2925	OPT	373	15	2925	2925	2925
F11	3835	3835	OPT	209	33	3835	OPT	1473	30	3835	OPT	4889	26	3835	3835	3835
F12	3395	3386	**OPT**	2341	289	3386	**3395**	4 h	255	3387	3392	4 h	107	**3395**	3390	**3395**
F13	2855	2855	OPT	19	28	2855	OPT	86	27	2855	OPT	306	20	2855	2855	2855
F14	3330	3330	OPT	23	9	3330	OPT	130	14	3330	OPT	822	13	3330	3330	3330
F15	3560	3560	OPT	277	41	3560	3560	4 h	169	3560	3560	4 h	95	3560	3560	3560
F16	2725	2725	OPT	44	18	2725	OPT	147	10	2725	OPT	543	9	2725	2725	2725
F17	2055	2055	OPT	6	7	2055	OPT	26	10	2055	OPT	90	7	2055	2055	2055
F18	3075	3062	3065	4 h	962	3062	3065	4 h	72	3062	3065	4 h	36	3065	3065	3065
F19	2525	2488	2515	4 h	804	2488	2515	4 h	164	2489	2514	4 h	124	2515	2500	**2525**
F20	2445	2445	OPT	56	21	2445	OPT	820	27	2445	OPT	2210	22	2445	2445	2445
F21	2930	2930	OPT	112	33	2930	OPT	1213	29	2930	OPT	3582	37	2930	2930	2930
F22	2075	2075	OPT	25	17	2075	OPT	70	16	2075	OPT	141	19	2075	2075	2075
F23	3005	2989	**3003**	4 h	140	2989	2998	4 h	53	2989	2994	4 h	26	**3003**[a]	2995	**3005**
F24	3210	3210	OPT	251	29	3210	OPT	2575	32	3210	OPT	10,106	33	3210	3210	3210
F25	1390	1390	OPT	4	15	1390	OPT	20	10	1390	OPT	89	10	1390	1390	1390
Num lb^{best}_{own}			25				24				22					
Num opt			22				20				19					
Avg time				2344				3575				5095				
Avg %gap		0.13	0.03			0.13	0.04			0.13	0.05					

[a] Since all routing costs are multiples of 5, this proves optimality $opt = 3005$.

deliver 6 + 7+4 + 5 = 22 optimality proofs, while we prove optimality in 4 + 5+4 + 4 = 17 cases.

At the end, 12 `egl` instances and 10 `bmcv` instances remain open.

4. Conclusions

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

We have applied the new labeling algorithm for SPPRC with combined task-$(k, 2)$-loop elimination for solving pricing subproblems in a branch-and-price algorithm for the CARP. It was known from the work of Bode and Irnich (2012) that task-k-loop elimination can significantly improve bounds of the linear relaxation of the column-generation master program. However, branching, i.e., a genuine branch-and-price was not possible due to the branching rule implying 2-loop elimination constraints on a new task set. The new results using the SPPRC subproblem relaxation with task-$(k, 2)$-loop elimination allow for a comparison of overall computation times and lower bounds when the branch-and-price algorithm terminates. Using standard benchmark set, we have shown that the approach is competitive: Several new best lower bounds were presented and some knowingly hard instances were solved to proven optimality for the first time.

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 2.3. Note that the proofs follow similar ideas as discussed in the first article on k-cycle elimination (focused on node-routing applications) and we refer the reader to this (Irnich & Villeneuve, 2006) for a more detailed motivation.

A.1. Proof of theorem on maximum number of set forms

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* $\omega(k_1,k_2):=(k_1-1)!^2\cdot(k_2-1)!^2\cdot\binom{(k_1-1)+(k_2-1)}{k_1-1}$*. This bound* $\omega(k_1,k_2)$ *is tight.*

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

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

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

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

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

$$\forall I_1\subseteq\{1,\ldots,k_1-1\}\text{ and }I_2\subseteq\{1,\ldots,k_2-1\}\text{ and }I\neq(\emptyset,\emptyset)$$

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

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

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

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

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

A.2. Proof of theorem on maximum number of set forms in an intersection of self-hole sets

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

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

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

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

References

Ahuja, R., Magnanti, T., & Orlin, J. (1993). *Network flows: Theory, algorithms, and applications*. Englewood Cliffs, New Jersey: Prentice Hall.

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

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

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

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

Belenguer, J.-M., Benevent, E., & Irnich, S. (2014). The capacitated arc routing problem: Exact algorithms. In A. Corberán & G. Laporte (Eds.), *Arc routing: Problems, methods and applications. MOS-SIAM series on optimization*. Philadelphia: SIAM. in press.

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

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

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

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

Bode, C., & Irnich, S. (in press). In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. *Transportation Science*. http://dx.doi.org/10.1287/trsc.2013.0507. in press.

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

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

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., & Villeneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic & G. Laporte (Eds.), *Fleet management and logistics* (pp. 57–93). Boston, Dordrecht, London: Kluwer Academic Publisher.

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

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

Dror, M. (Ed.). (2000). *Arc routing: Theory, solutions and applications*. Boston: Kluwer.

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

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

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

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

Irnich, S., & Desaulniers, G. (2005). Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, & M. Solomon (Eds.), *Column generation* (pp. 33–65). New York, NY: Springer.

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

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

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

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

Mingozzi, A., Bianco, L., & 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. (July 2012). Personal Communication.

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

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

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

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