The Shortest-Path Problem with Resource Constraints and k-Cycle Elimination for $k \geq 3$

Stefan Irnich
RWTH Aachen University, Deutsche Post Lehrstuhl für Optimierung von Distributionsnetzwerken, Templergraben 64, 52062 Aachen, Germany, sirnich@or.rwth-aachen.de

Daniel Villeneuve
AD OPT Technologies Inc., 3535 Queen Mary, Suite 650, Montréal, Québec, Canada H3V 1H8, danielv@ad-opt.com

The elementary shortest-path problem with resource constraints (ESPPRC) is a widely used modeling tool in formulating vehicle-routing and crew-scheduling applications. The ESPPRC often occurs as a subproblem of an enclosing problem, where it is used to generate implicitly the set of all feasible routes or schedules, as in the column-generation formulation of the vehicle routing problem with time windows (VRPTW). The ESPPRC problem being NP-hard in the strong sense, classical solution approaches are based on the corresponding non-elementary shortest-path problem with resource constraints (SPPRC), which can be solved using a pseudo-polynomial labeling algorithm. While solving the enclosing problem by branch-and-price, this subproblem relaxation leads to weak lower bounds and sometimes impractically large branch-and-bound trees. A compromise between solving ESPPRC and SPPRC is to forbid cycles of small length. In the SPPRC with k-cycle elimination (SPPRC-k-cyc), paths with cycles are allowed only if cycles have length at least $k+1$. The case $k = 2$ forbids sequences of the form $i - j - i$ and has been successfully used to reduce integrality gaps. We propose a new definition of the dominance rule among labels for dealing with arbitrary values of $k \geq 2$. The numerical experiments on the linear relaxation of some hard VRPTW instances from Solomon's benchmark show that k-cycle elimination with $k \geq 3$ can substantially improve the lower bounds of vehicle-routing problems with side constraints. The new algorithm has proven to be a key ingredient for getting exact integer solutions for well-known hard problems from the literature.

Key words: shortest paths; cycle elimination; column generation; vehicle routing
History: Accepted by William J. Cook, Area Editor for Design and Analysis of Algorithms; received August 2003; revised May 2004; accepted August 2004.

1. Introduction

The elementary shortest-path problem with resource constraints (ESPPRC) is a widely used modeling tool in formulating vehicle-routing and crew-scheduling applications, see e.g. Desaulniers et al. (1998). The ESPPRC consists of finding shortest paths that do not contain cycles from a source to all other nodes of a network. The ESPPRC often occurs as a subproblem of an enclosing problem, where it is used to generate implicitly the set of all feasible routes or schedules, as in the column-generation formulation of the vehicle-routing problem with time windows (VRPTW), see Cordeau et al. (2002). The ESPPRC problem being NP-hard in the strong sense (Dror 1994), classical solution approaches are based on the corresponding non-elementary shortest-path problem with resource constraints (SPPRC), which can be solved using a pseudo-polynomial labeling algorithm (Desrochers and Soumis 1988). While solving the enclosing problem by branch-and-price (see Barnhart et al. (1998) for an introduction to the methodology), this subproblem relaxation leads to weak lower bounds and sometimes impractically large branch-and-bound trees.

A compromise between solving ESPPRC and SPPRC is to forbid cycles of small length. In the SPPRC with k-cycle elimination (SPPRC-k-cyc), paths with cycles are allowed only if cycles have length at least $k+1$. The case $k=2$ forbids sequences of the form $i-j-i$, is well known (Houck et al. 1980), and has been successfully used to reduce integrality gaps for the VRPTW (Kolen et al. 1987, Desrochers et al. 1992). We propose a new definition of the dominance rule among labels for dealing with arbitrary values of $k \geq 2$. This new rule can be embedded in any algorithm for finding the minima of a set of vectors, such as a naive quadratic algorithm (comparing all pairs of labels) or a multi-dimensional divide-and-conquer algorithm, in order to identify the set of irrelevant labels. The multi-dimensional divide-and-conquer algorithm was first presented by Kung et al. (1975) to find the maxima of a set of vectors, and was later generalized by Bentley (1980) to solve a wide variety of problems, e.g. domination problems, maxima, range-searching, closest-pair, and nearest-neighbor problems. The numerical experiments on the linear relaxation of some hard VRPTW instances from the Solomon's benchmark (see Solomon 1987) show that k-cycle elimination with $k \geq 3$ can substantially improve the lower bounds. Using well-known techniques for branching and cutting (Kohl 1995, Kohl et al. 1999, Rich 1999), the new algorithm has proved to be a key ingredient for getting exact integer solutions for well-known hard problems from the literature.

We start with some basic notation on graphs, resource constraints, and the definition of the shortest-path problems with resource constraints in Section 2. Section 3 deals with pareto optimality and the concept of useful paths. Section 4 defines a template for labeling algorithms for solving SPPRC and SPPRC-k-cyc and identifies the main building blocks of such a labeling algorithm. In Section 5 we review cycle elimination, its basic definitions, and the progress that has been made to adapt dominance rules to incorporate cycle-elimination constraints. Section 6 focuses on the building blocks of a labeling algorithm for the general case of k-cycle elimination with $k \geq 2$. We will introduce the concept of hole sets, generalize dominance rules, describe the corresponding data structures and algorithms, and derive bounds necessary for the final worst-case complexity results. Section 7 discusses various extensions. Computational results for the VRPTW are presented in Section 8 and the paper ends with final conclusions in the last section.

2. Shortest-Path Problems with Resource Constraints

The shortest-path problems with resource constraints that we consider in this paper are extensions of the classical one-to-all shortest-path problem, where the cost is replaced by multi-dimensional resource vectors that are accumulated along paths and constrained at intermediate nodes. The objective is to find all pareto-optimal paths from a source node to all other nodes. The problem is stated on a simple digraph $G = (V, A)$, with V being the non-empty set of nodes and A being the set of arcs. Let $s \in V$ be designated as the *source* node. Minimal resource consumptions are associated to arcs, while lower and upper bounds on cumulative consumption from the source node are imposed at each node.

We state the resource constraints by considering individual paths. A *path* $P = (a_1, \ldots, a_p)$ is a finite sequence of arcs (some arcs may occur more than once) where the head node of a_i is identical to the tail node of a_{i+1} for all $i = 1, \ldots, p-1$. The graph being simple, such a path can be written as $P = (v_0, v_1, \ldots, v_p)$ with the understanding that $(v_{i-1}, v_i) \in A$ for all $i \in \{1, \ldots, p\}$. The number of arcs p is the length of the path. An *elementary path* is a path in which all nodes are different.

A *cycle* is a path $(v_0, v_1, \ldots, v_p)$ of length $p > 1$ having $v_0 = v_p$. For simplicity, we name any cycle of length less than or equal to k as a *k-cycle* or *small cycle*. To refer to its exact length p, it can be denominated as a *cycle of length p*.

The resources are represented as vectors in $\mathbb{R}^R$, where R is the number of different resources. The minimal resource consumptions associated to an arc $(i, j) \in A$ are denoted by $t_{ij} = (t_{ij}^1, \ldots, t_{ij}^R)$ and the resource intervals associated to a node $i \in V$ are denoted by $[a_i, b_i]$ with $a_i = (a_i^1, \ldots, a_i^R)$ and $b_i = (b_i^1, \ldots, b_i^R)$. We assume for now that $t_{ij}^1 > 0$ holds for all arcs $(i, j) \in A$. This condition will be relaxed in Section 7.

A vector $T = (T^1, \ldots, T^R) \in \mathbb{R}^R$ is not greater than (i.e. dominates) a vector $S = (S^1, \ldots, S^R) \in \mathbb{R}^R$ if for all components $i = 1, \ldots, R$, the inequality $T^i \leq S^i$ holds. We denote this fact by $T \leq S$. The relation $\leq$ provides a partial ordering of the vector space $\mathbb{R}^R$. A vector $T = (T^1, \ldots, T^R) \in \mathbb{R}^R$ is lexicographically less than or equal to a vector $S = (S^1, \ldots, S^R) \in \mathbb{R}^R$, which we write $T \preceq_{lex} S$, if they are equal or if there exists an index $i^* \in \{1, \ldots, R\}$ with $T^i = S^i$ for all $1 \leq i < i^*$ and $T^{i^*} < S^{i^*}$. The lexicographical ordering provides a total ordering of the vector space $\mathbb{R}^R$. Furthermore, the $\leq$ relation is included in the lexicographical relation, i.e. $T \leq S$ implies $T \preceq_{lex} S$.

A path $P = (v_0, v_1, \ldots, v_p)$ is *resource-feasible* if there exist vectors $T_i \in \mathbb{R}^R$ for all positions $i = 0, 1, \ldots, p$ such that $T_i \in [a_{v_i}, b_{v_i}]$ holds for all $i = 0, 1, \ldots, p$ and $T_{i-1} + t_{v_{i-1},v_i} \leq T_i$ holds for all $i = 1, \ldots, p$.

Assume that an arbitrary path $P = (v_0, v_1, \ldots, v_p)$ is given. Checking whether P is resource-feasible or not can be done by computing a minimal cumulative resource consumption at each node along the path and validating these consumptions against the upper bounds. For each node position $i = 0, 1, \ldots, p$, a vector T_i in $\mathbb{R}^R$ can be computed recursively from the lower bounds at the start node v_0

$$T_0 := a_{v_0} \tag{1}$$

and from the minimal consumption along the path given by

$$T_i := \max\{a_{v_i}, T_{i-1} + t_{v_{i-1},v_i}\} \quad \text{for all } i \in \{1, \ldots, p\}. \tag{2}$$

The path P is resource-feasible if and only if $T_i \leq b_i$ for all $i \in \{0, 1, \ldots, p\}$ defined by (1) and (2). We associate to each resource-feasible path $P = (v_0, v_1, \ldots, v_p)$ the unique *resource vector* $res(P) = T_p$ computed by (1)–(2).

Using the above notation, the shortest-path problem with resource constraints (SPPRC) consists of finding the set of all pareto-optimal resource vectors and a corresponding set of resource-feasible paths starting at the source node s. Whenever there are cycles of negative length for some resource $r \in \{1, \ldots, R\}$, some pareto-optimal paths could contain cycles. The existence of negative length cycles occurs naturally, for example, in the "cost" resource when using the SPPRC as a subproblem in a column-generation or Lagrangean-relaxation framework.

If cycles are not allowed in the solution to the SPPRC, then one needs to include cycle-elimination constraints, leading to the formulation of the elementary shortest-path problem with resource constraints (ESPPRC). In particular, integer feasibility of node-partitioning problems into several paths implies that the chosen subsets correspond to elementary paths (e.g., VRPTW).

In this paper, we focus on the controlled introduction of cycle-elimination constraints, by specifying the minimum length $k+1$ of the allowed cycles in the solution. This defines the shortest-path problem with resource constraints and k-cycle elimination (SPPRC-k-cyc). In problems that need elementary shortest paths as a part of the solution, two extreme approaches are, on the one hand, solving ESPPRC subproblems and on the other hand, solving SPPRC subproblems. Usually, lower bounds obtained with the former approach are tighter than the ones obtained with the latter. These tighter lower bounds are expected to lead to smaller branch-and-bound trees. Nonetheless, branching rules are needed to obtain integer solutions in both cases. Moreover, the complexity of the ESPPRC limits the size of the instances solvable in practice. We propose the SPPRC-k-cyc as a means to get better lower bounds than SPPRC while keeping the computational effort tractable.

3. Pareto Optimality and Useful Paths

Let $\mathcal{F}(u, v)$ be the set of all resource-feasible paths from a node u to a node v and $\mathcal{S}$ be the set of all paths feasible with respect to cycle-elimination constraints, i.e. $\mathcal{S} = \{\text{all paths}\}$ for SPPRC, $\mathcal{S} = \{\text{elementary paths}\}$ for ESPPRC, and $\mathcal{S} = \{k\text{-cycle free paths}\}$ for SPPRC-k-cyc. For a given path $P \in \mathcal{S}$, we denote by $\mathcal{E}(P)$ the set of all possible extensions Q such that $(P, Q) \in \mathcal{S}$, i.e. $\mathcal{E}(P) = \{Q \in \mathcal{S} : (P, Q) \in \mathcal{S}\}$.

The SPPRC, SPPRC-k-cyc, and ESPPRC have been stated as the problems of finding the set of all pareto-optimal resource vectors and corresponding resource-feasible paths in $\mathcal{S}$ starting at the source node s. Note that a vector $T \in \mathbb{R}^R$ in a set is *pareto-optimal* if no other vector $S \in \mathbb{R}^R$ from the set satisfies $S \leq T$. The set of pareto-optimal resource vectors at node $v \in V$ is denoted as $PO(v)$. Obviously, the set of solutions at node s is $PO(s) = \{a_s\}$. The task of the SPPRC, SPPRC-k-cyc, or ESPPRC is to compute the sets $PO(v)$ for $v \in V \setminus \{s\}$. The corresponding set of pareto-optimal paths $POP(v)$ is given by $POP(v) = \{P \in \mathcal{S} \cap \mathcal{F}(s, v) : res(P) \in PO(v)\}$. Obviously, there can exist more pareto-optimal paths than resource vectors, i.e. $|POP(v)| \geq |PO(v)|$, but we need to compute only one path for each pareto-optimal resource vector.

In the following, we will analyze the relation between $PO(v)$, $POP(v)$, and a set of *useful paths* that need to be considered for constructing $PO(v)$ in a (labeling) algorithm. In order to construct the set of solutions $PO(v)$, paths from the sets of solutions under consideration at predecessor nodes of v can be extended according to (2). For efficiency considerations, one should restrict this extension step to small subsets of paths at predecessor nodes that will nonetheless produce all the pareto-optimal resource vectors at node v. We will show that for solving SPPRC one can consider a single path from $POP(v)$ for each element of $PO(v)$ while for solving SPPRC-k-cyc and ESPPRC other useful paths, not necessarily in $POP(v)$, also have to be considered.

Lemma 1 *Let $P_1, \ldots, P_t$ and P be paths in $\mathcal{F}(s, v) \cap \mathcal{S}$. Let*

(D) *$res(P_i) \leq res(P)$ for all $i \in \{1, \ldots, t\}$*

(E) *$\mathcal{E}(P) \subseteq \mathcal{E}(P_1) \cup \ldots \cup \mathcal{E}(P_t)$.*

Let $Q \in \mathcal{S}$ be an arbitrary path ending at node w.

If P can be feasibly extended in direction to Q, i.e. $(P, Q) \in \mathcal{F}(s, w) \cap \mathcal{S}$, then the same holds for at least one P_i, $i \in \{1, \ldots, t\}$, i.e. $(P_i, Q) \in \mathcal{F}(s, w) \cap \mathcal{S}$, with $res(P_i, Q) \leq res(P, Q)$.

Lemma 1 implies that a path P fulfilling the above conditions is not useful and does not need to be considered for constructing the sets $PO(v)$ for any $v \in V$.

In the case of the SPPRC, the equality $\mathcal{E}(P) = \mathcal{S}$ holds for all paths P so that condition (E) is always true. For solving SPPRC, Lemma 1 can be interpreted in the following way:

1. Consider only pareto-optimal paths.

2. For any set of pareto-optimal paths ending at the same node and having the same resource vector, keep only one of these arbitrarily.

One possible way to formalize the latter idea of choosing one path among paths with the same resource vector is to restrict the $\leq$ relation between their resource vectors by an acyclic dominance relation $\prec_{dom}$ comparing paths. For resolving the ambiguity, we can devise any total order on paths, for example, by comparing their finite node sequences or more efficiently, by attaching to each path a unique arbitrary number (an identifier "id") at path construction time. Using the latter idea, we have for two different paths P_1 and P_2 either $id(P_1) < id(P_2)$ or $id(P_1) > id(P_2)$. For paths $P_1, P_2 \in \mathcal{F}(s, v)$ the relation $P_1 \prec_{dom} P_2$ holds iff $res(P_1) < res(P_2)$ or $(res(P_1) = res(P_2)$ and $id(P_1) < id(P_2))$.

In the SPPRC-k-cyc (resp. ESPPRC), the feasibility of extending a path depends on two aspects, namely, the resource vector and the last k nodes (resp. all visited nodes) of the path. Because of this latter aspect, given distinct paths P and Q with $P \prec_{dom} Q$ ending at

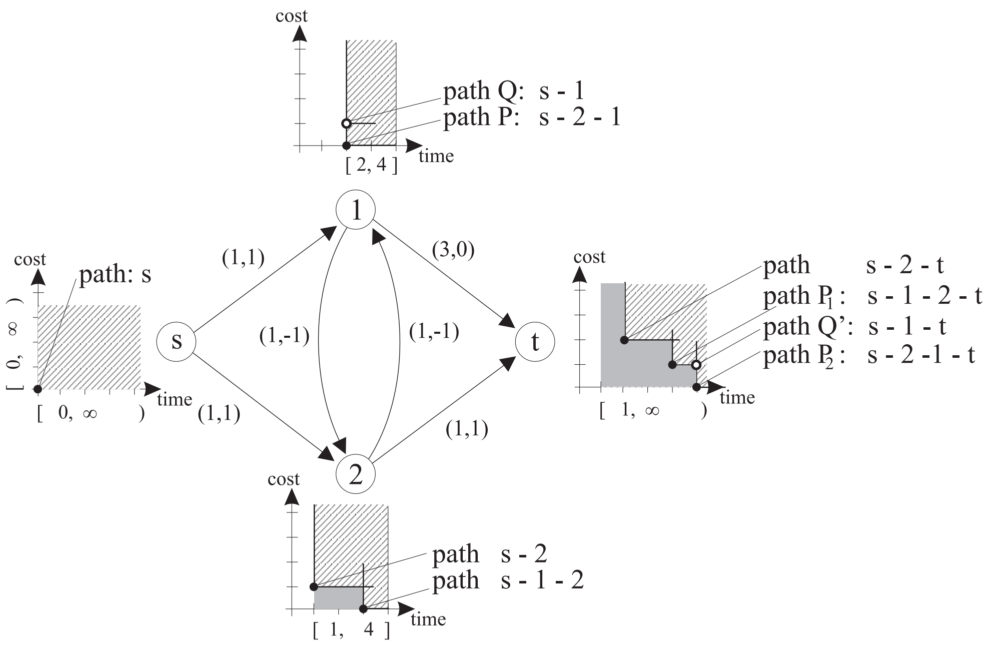


Figure 1: Example of an SPPRC-k-cyc

the same node, there could exist feasible extensions of Q that are not feasible for P. This situation for an SPPRC-2-cyc with two resources is depicted in Figure 1 for paths P and Q at node 1 and the extension to node 2. (Figure 1 shows for each node i the resource space, which is given as the cartesian product of the resource windows $[a_i^{time}, b_i^{time}] \times [a_i^{cost}, b_i^{cost}]$. Each arc (i, j) shows the minimum resource consumption $(t_{ij}^{time}, t_{ij}^{cost})$. The resource vectors $res(\cdot)$ of all k-cycle free paths are shown in the resource space. Pareto-optimal paths are depicted by the symbol $\bullet$, non-pareto-optimal paths by $\circ$.) To cover all feasible extensions $\mathcal{E}(Q)$ of Q, one needs a set $\{P_1, \dots, P_l\}$ of paths with $P_i \prec_{dom} Q$ for all $i \in \{1, \dots, l\}$. This means that for each feasible extension of Q, there has to be at least one path P_i that can be extended in the same way (an example for this is also depicted in Figure 1 at node t with paths Q', P_1, and P_2 having $P_1, P_2 \prec_{dom} Q$). Note that many of these paths P_i could have identical resource vectors, even identical to $res(Q)$. As long as some feasible extensions of Q are not covered, Q cannot be discarded. Such a path Q could also be used to cover feasible extensions of another path R for which $res(Q) \leq res(R)$.

The example of Figure 1 also shows that the usefulness of paths strongly depends on the value of k. At node t the path Q' is not pareto-optimal because paths P_1 and P_2 have resource vectors $res(P_1), res(P_2) \lneqq res(Q')$. Assume that the figure just shows a small portion of a digraph and that node t is an intermediate node that connects back to node 1, 2, and some other nodes. In the case of $k = 2$, the two paths P_1 and P_2 cover all possible extensions of Q', so Q' can be discarded. In the case of $k = 3$ neither P_1 nor P_2 can be extended back to node 1 or node 2 while Q' is allowed to be extended to node 2. For $k \geq 4$, all paths of the form $(\cdot 1)$, $(\cdot 2)$ are additional invalid extensions of P_1 and P_2 ("$\cdot$" is a wildcard for any node $v \in V$) while Q' is allowed to be extended to $(x2)$ with $x \notin \{1, 2\}$. Therefore, for the purpose of computing k-cycle-free pareto-optimal sets at other nodes, Q' cannot be discarded if $k \geq 3$, although Q' is not in the set of pareto-optimal solutions at node t for any k.

For all cases (SPPRC, ESPPRC, SPPRC-k-cyc), we can now describe a set of paths that have to be considered for constructing $PO(v)$ at each node $v \in V$. This set of *useful paths* is defined as

$$U(v) = \{Q \in \mathcal{S} \cap \mathcal{F}(s,v) : \mathcal{E}(Q) \not\subseteq \bigcup_{P \in \mathcal{S} \cap \mathcal{F}(s,v) : P \prec_{dom} Q} \mathcal{E}(P)\}. \tag{3}$$

For the SPPRC, the set $U(v)$ contains exactly $|PO(v)|$ different pareto-optimal paths because $\bigcup_{P \in \mathcal{S} \cap \mathcal{F}(s,v) : P \prec_{dom} Q} \mathcal{E}(P)$ is either $\mathcal{S}$ or $\varnothing$ depending on whether there exists a path P with $P \prec_{dom} Q$ or not (note that the union over an empty index set is the empty set).

One of the main contributions of this paper is to develop an efficient way of encoding $\mathcal{E}(Q)$ and $\bigcup_{P \in \mathcal{S} \cap \mathcal{F}(s,v) : P \prec_{dom} Q} \mathcal{E}(P)$ and to check inclusion among these sets. The labeling algorithm for SPPRC — presented in the next section — based on the idea of useful paths will end up with at least one path with pareto-optimal resource vector, i.e. one element $P \in POP(v)$ for each resource vector in $PO(v)$. In addition to this we will show for SPPRC-k-cyc that the maximum number of useful paths to consider, i.e. $|U(v)|$, grows by a factor $\alpha(k)$ (independent of the size of the digraph) compared to $|PO(v)|$ in the SPPRC.

4. Labels and Labeling Algorithms

This section presents the remaining concepts and notation that are needed to discuss the literature on cycle elimination and our labeling algorithm.

The concept of labels has been introduced to store efficiently a large number of different paths and their corresponding resource vectors. Whenever two paths P_1 and P_2 share a subset of arcs as a prefix, this common part is stored as a single chain of labels. For a comprehensive introduction to the ideas of labeling algorithms for shortest-path problems, the reader is referred to Ahuja et al. (1993), and for labeling algorithms for the SPPRC to Desrochers and Soumis (1988) and Powell and Chen (1998).

Let $P = (v_0, v_1, \ldots, v_p)$ be any resource-feasible path starting at the source node $v_0 = s$. Each subpath $P_i = (v_0, \ldots, v_i)$ for $i = 0, \ldots, p$ is also resource-feasible by definition.

Instead of storing all these paths separately, a label L_i is associated to each path P_i. A label $L = L(P)$ associated to a path $P = (v_0, v_1, \ldots, v_p) \in \mathcal{F}(s, v_p)$ is consequently defined as the resource vector $res(P) = T_p = (T_p^1, T_p^2, \ldots, T_p^R) \in \mathbb{R}^R$ and (a link to) its predecessor label corresponding to the subpath $(v_0, v_1, \ldots, v_{p-1})$. Since there is a one-to-one correspondence between labels and paths, we write $L = L(P)$ (resp. $P = P(L)$) to refer to the corresponding label (path) and $res(L)$, $\mathcal{E}(L)$, etc. instead of $res(P)$, $\mathcal{E}(P)$. The last node of the corresponding path $P = P(L)$ is named the *resident node* of L and is denoted by $v(L) = v_p$. The predecessor node, when it exists, is denoted by $pred(L) = v_{p-1}$. The initial label L at the source node s corresponding to the path $P = (s)$ of length 0 does not have a predecessor node, and $pred(L)$ is then defined as $\perp$.

In SPPRC labeling algorithms (see e.g. Desrochers and Soumis 1988, Powell and Chen 1998), new labels need to be produced from existing ones before being compared and either being kept or discarded. *Extending* a label L resident at node $v_p = v(L)$ *in direction to node* $v_{p+1} \in V$ (resp. *along the arc* $(v_p, v_{p+1}) \in A$) means to check whether the vector T_{p+1} defined by (2) satisfies the upper bounds, i.e. the condition $T_{p+1} \leq b_{v_{p+1}}$.

We use the fact that efficient labeling algorithms exist for solving the SPPRC to derive an algorithm for the SPPRC-k-cyc. Because of our hypothesis that $t_{ij}^1 > 0$, these algorithms can use a label-setting approach and only extend pareto-optimal, or more generally *useful*, labels. We assume that two algorithmic steps can be identified and isolated in the base SPPRC algorithm:

1. *Dominance rule*: discard a label L if it can be proved that this label is not needed to produce any pareto-optimal path at any node $v \in V$;

2. *Path-extension step*: produce a new path at node w from an existing pareto-optimal resource-feasible path at a node v with $(v, w) \in A$, such that the new path is resource-feasible.

The dominance rule is meant to be invoked from a general dominance algorithm, which could be a naive quadratic all-pairs comparison algorithm or a more sophisticated multi-dimensional divide-and-conquer algorithm as described in Kung et al. (1975) and Bentley (1980). A *template for labeling algorithms* for the SPPRC family of problems is given by Algorithm 1 in Figure 2.

The use of an intermediate set $\mathcal{L}_v$ of potentially permanent labels, made possible by the hypothesis that $t_{ij}^1 > 0$, allows batch processing of several labels in the dominance and path-extension steps (see Desrochers and Soumis 1988 for better bucket criteria than

Algorithm 1 (* Template for a Labeling Algorithm for (E)SPPRC, SPPRC-k-cyc *)

```
1:  (INPUT:)
    Digraph $(V, A)$ with resources $R$, resource windows $[a_i, b_i]$, minimal consumptions $t_{ij}$, $t^1_{ij} > 0$
2:  (Initialization)
    LET $U(v) := \varnothing$ for all $v \in V$.
3:  LET $\mathcal{L} := \{L\}$ be the set of unprocessed labels, with $v(L) = s$, $res(L) = a_s$ and $pred(L) = \perp$.
4:  LET $t_{min} := \min_{(ij)\in A} t^1_{ij}$.
5:  (Main Loop)
    WHILE $\mathcal{L} \neq \varnothing$
6:      LET $v$ be a node associated to a label $L_v \in \mathcal{L}$ with $res^1(L_v)$ being minimum over all labels in $\mathcal{L}$.
7:      IDENTIFY a subset of labels $\mathcal{L}_v \subset \mathcal{L}$ resident at node $v$ with "small" resource $res(L)^1$, i.e.
        $\mathcal{L}_v := \{L \in \mathcal{L} : v(L) = v, res(L)^1 < res(L_v) + t_{min}\}$.
8:      LET $\mathcal{L} := \mathcal{L} \setminus \mathcal{L}_v$.
9:      (Dominance Algorithm)
        APPLY a dominance algorithm to all labels resident at node $v$, i.e. to $U(v) \cup \mathcal{L}_v$.
10:     LET $U'(v)$ be the subset of $\mathcal{L}_v$ that was not discarded when applying the dominance rule within
        the dominance algorithm.
11:     LET $U(v) := U(v) \cup U'(v)$.
12:     (Path-Extension Step)
        FOR EACH label $L \in U'(v)$
13:         PRODUCE feasible extensions $L'$ of $L$ for each $w \in V$ such that $(P(L), w) \in \mathcal{F}(s, w) \cap \mathcal{S}$.
14:         ADD these feasible extensions $L'$ to $\mathcal{L}$.
15:     ENDFOR
16: ENDWHILE
17: (FILTER:)
    EXTRACT the sets $PO(v)$ of pareto-optimal resource vectors from $\{res(P) : P \in U(v)\}$ for all
    $v \in V$
18: (OUTPUT:)
    Sets of pareto-optimal labels corresponding to $PO(v)$ for each node $v \in V$
```

Figure 2: Template for a Labeling Algorithm for (E)SPPRC and SPPRC-k-cyc

our expository use of t_{min}). The sets $PO(v)$ can be extracted from $U(v)$ by using any vector-dominance algorithm, as in the case of the plain SPPRC algorithm. By refining the *dominance rules* to discard non-useful paths instead of all non-pareto-optimal paths and by adapting the *path-extension step*, the same algorithms solve the SPPRC, SPPRC-k-cyc, or ESPPRC.

The path-extension step is easily modified to take into account the specificity of SPPRC-k-cyc and ESPPRC. The feasibility criterion has to include the validation of cycle-elimination constraints. For the SPPRC-k-cyc, this is done by not extending a label L to nodes identical to $v(L)$ or any of the $k-1$ predecessor nodes $pred^1(L), \ldots pred^{k-1}(L)$. For the ESPPRC, the label L cannot be extended to any of its predecessor nodes.

The dominance rule needs to leave on each node v not only one pareto-optimal label corresponding to each vector in $PO(v)$ but also other useful labels that will have to be extended to incident nodes. Section 5 presents the criteria for ESPPRC and for the special case of SPPRC-2-cyc. Section 6 explains the details of dealing with k-cycle elimination for general values of $k \geq 2$.

5. A Review of Cycle Elimination

The dominance relation can be used to identify pareto-optimal paths when the resource vectors of labels are distinct (as is the case when resource vectors are taken from a set). In this case, labels that are dominated are not pareto-optimal. For the SPPRC, this means that dominated labels can be discarded, since they correspond neither to useful intermediate subpaths nor to final pareto-optimal paths. However, labels with identical resource vectors might arise when extending several labels from predecessor nodes. In the context of the SPPRC, only one arbitrary label with a given resource vector has to be considered, because all such labels share the same feasible extensions. For the SPPRC-k-cyc, we pointed out in Section 3 that many paths might be needed to cover all the possible extensions of a single path. This means that dominated labels cannot be discarded without considering their possible extensions. In particular, it is no longer true that we can choose an arbitrary representative label for a given resource vector and discard the others. Therefore, when considering cycle elimination, the dominance relation still identifies pareto-optimal paths but fails to provide a criterion to eliminate useless intermediate subpaths.

In the first Section 5.1, we propose a simple but inefficient dominance rule for k-cycle elimination that groups labels according to the sequence of their k last nodes before applying the usual elimination rule based on dominance. Sections 5.2 and 5.3 present more efficient algorithms for specific values of k. If $k \geq |V|$, then the SPPRC-k-cyc becomes equivalent to the ESPPRC and Beasley and Christofides (1989) as well as Guéguen et al. (1998) have proposed handling cycle elimination by encoding the already-visited nodes as additional binary resources. If $k = 2$, Houck et al. (1980) provide refined elimination rules that allow discarding a label if there are two dominating labels with different predecessors. Kohl (1995) and Larsen (1999) extend these rules further by analyzing resource consumption in the neighborhood of each node.

5.1. A Simple Dominance Rule for k-Cycle Elimination

A correct rule for using dominance in order to eliminate a label L is to ensure that the dominating label L^{dom} has the same feasible extensions as L:

> Let labels L and L^{dom} have identical predecessor vectors $pred(L) = pred(L^{dom})$ and the same resident node $v(L) = v(L^{dom})$. Then the usual elimination criterion for SPPRC based on dominance ($L^{dom} \prec_{dom} L$) applies.

This approach works because labels with identical predecessor vectors and the same resident node share the same feasible extensions. But there is one important drawback with such a simple approach: in the worst case, we expect the number of labels to grow by a factor of $\mathcal{O}(n^{k-1})$ (for example, take a complete digraph (V, A) with all resource consumptions equal to one and resource intervals $[0, k]$).

Efficient cycle elimination requires a reduction in the number of labels, which can be obtained by allowing labels with different predecessor vectors to contribute parts of the feasible extensions that are needed to eliminate a given label.

5.2. Dominance Rules for the Elementary SPPRC

For large values of k (i.e. $k \geq |V|$), the SPPRC with k-cycle elimination becomes equivalent to the ESPPRC. In this case, it is not necessary to store the predecessor nodes of a label L as a *sequence*. Since it is not allowed to visit a node more than once, its position in the vector $pred(L)$ is irrelevant and $pred(L)$ can be considered as a *set*. Then, labels can be grouped according to their predecessor set and the usual elimination criterion for SPPRC based on dominance applies to each group. This is an improvement over the strategy presented in the previous subsection because more labels are grouped together and all the permutations of predecessor nodes being considered as identical.

Beasley and Christofides (1989) model the ESPPRC by defining one additional binary resource for each node in the graph. The minimum resource consumption on an arc $(i, j) \in A$ with respect to the additional resource j is 1 while the other minimum resource consumptions are set to 0. This approach allows them to solve the ESPPRC as an SPPRC with many resources. Instead of considering additional resources, their approach can be interpreted as a reformulation of the elimination criterion taking into account subsets of already-visited nodes:

> Let labels L_{dom} and L have predecessor *sets* satisfying $pred(L_{dom}) \subseteq pred(L)$ and the same resident node $v(L_{dom}) = v(L)$. Then the usual elimination criterion for SPPRC based on dominance ($L_{dom} \prec_{dom} L$) applies.

Guéguen et al. (1998) improve the idea of Beasley and Christofides by a change in the point of view on the predecessor set $pred(L)$. The predecessor set $pred(L)$ originally represents the *"set of nodes that have been visited"* by the path $P(L)$. Another interpretation is that $pred(L)$ is the *"set of nodes that cannot be visited any more."* Their improvement is to look at the resource vector $res(L)$ of a label L to identify nodes that cannot be visited anymore (e.g. because of time-window constraints and non-negative travel times). These nodes are added to the set $pred(L)$. As a result, the usual elimination criterion for SPPRC can eliminate more labels.

5.3. Enhanced Dominance Rules for 2-Cycle Elimination

The SPPRC with 2-cycle elimination was first studied by Houck et al. (1980) in the context of solving the traveling-salesman problem. The presentation of this section is taken from Kohl (1995) and Larsen (1999), which covers and extends the label-elimination rules of Houck et al. (1980).

The rules are sensitive to the occurrence of labels with identical resource vectors. The ambiguity resulting from having both $L_1 \prec_{dom} L_2$ and $L_2 \prec_{dom} L_1$ for two labels L_1 and L_2 with the same resource vector $res(L_1) = res(L_2)$ can be avoided by using the same strategy as described in Section 3.

The following three definitions are the key elements for the 2-cycle case.

- A label L^{dom} with resource vector $(T^1_{dom}, \ldots, T^R_{dom})$ is called *strongly dominant* if it is not dominated and at least one of its resources $r^* \in \{1, \ldots, R\}$ satisfies

$$T^{r^*}_{dom} + t^{r^*}_{v(L^{dom}),pred(L^{dom})} > b^{r^*}_{pred(L^{dom})}. \tag{4}$$

The inequality (4) means that L^{dom} cannot be extended to its predecessor node. As a consequence, any label L dominated by a strongly-dominant label L^{dom} cannot be extended to $pred(L^{dom})$ and can therefore be discarded.

- A label L^{dom} is called *semi-strongly dominant* if it is not dominated and it is not strongly dominant.

  Semi-strongly dominant labels L have the potential of being extended to the second node $v = v(L)$ of a 2-cycle $w - v - w$.

- A label L is called *weakly dominant* if it is only dominated by semi-strongly dominant labels, these semi-strongly dominant labels $L_1^{dom}, \dots, L_p^{dom}$ have the same predecessor node $pred(L_1^{dom}) = \cdots = pred(L_p^{dom})$ and the predecessor of L is different, i.e. $pred(L) \neq pred(L_1^{dom})$.

All labels that are neither strongly dominant nor semi-strongly dominant nor weakly dominant can be discarded.

A weakly dominant label is not allowed to be discarded. Instead, it has to be extended to *at least* the predecessor node of its dominating labels. Efficient algorithms for SPPRC-2-cyc *only extend* weakly dominant labels to the predecessor node of its dominating labels, although this is not a logical requirement. For further details about implementing a SPPRC-2-cyc algorithm, the reader is referred to Kohl (1995) and Larsen (1999).

6. Dominance for an SPPRC-k-cyc Labeling Algorithm

As pointed out in Section 4, the main building blocks of a labeling algorithm are the path-extension step and the dominance rule, regardless of the problem at hand, SPPRC, SPPRC-k-cyc, or ESPPRC. In this section, we focus on the dominance aspects for SPPRC-k-cyc for a given value of $k \geq 2$. We first propose an efficient encoding of each set $\mathcal{E}(P)$ of possible extensions and of unions $\mathcal{E}(P_1) \cup \cdots \cup \mathcal{E}(P_t)$ of *covered* extensions. We then derive worst-case bounds on the size of such an encoding and on the number of useful paths with the same resource vectors that might be kept at each node.

6.1. Encoding the Possible Extensions by Hole Sets

It is possible to encode efficiently (possibly infinite) sets of paths for which a finite set of positions are fixed. We call such a set of paths with some positions being assigned to fixed nodes a *set form*. A set form can be encoded as a single finite vector s of elements

$v \in V \cup \{\cdot\}$, with the meaning that this vector represents all paths $Q \in \mathcal{S}$ whose ith node is v if $f_i = v \in V$.

For a label L corresponding to a path $P = P(L) \in \mathcal{S} \cap \mathcal{F}(s, v)$ the following property holds: $Q \in \mathcal{E}(P)$ if and only if $Q = (v_1, \ldots, v_k, \ldots, v_q) \in \mathcal{S}$ and $(v_1, \ldots, v_k) \in \mathcal{E}(P)$. Therefore, the *relevant information* for paths $Q \in \mathcal{E}(P)$ can be encoded by considering only the first k nodes of Q. By taking the complement of $\mathcal{E}(P)$ with respect to $\mathcal{S}$, we can encode $\mathcal{E}(P)$ as the implicit complement of a finite union of set forms, each set form being encoded as a vector of length k. By the *self-hole set* of the path P we mean the finite union $H(P)$ of set forms identified by the set of paths Q, which, when appended to P, produce a path $(P, Q) \notin \mathcal{S}$ that is infeasible with respect to k-cycle elimination constraints.

As a 4-cycle-elimination example, a label L with last 4 nodes $(pred^3(L), pred^2(L), pred^1(L), v(P)) = (a, b, c, v)$ cannot be extended to any path $Q \in H(L)$ included in one of the following set forms: $(v, \cdot, \cdot, \cdot)$, $(\cdot, v, \cdot, \cdot)$, $(\cdot, \cdot, v, \cdot)$, $(\cdot, \cdot, \cdot, v)$, $(c, \cdot, \cdot, \cdot)$, $(\cdot, c, \cdot, \cdot)$, $(\cdot, \cdot, c, \cdot)$, $(b, \cdot, \cdot, \cdot)$, $(\cdot, b, \cdot, \cdot)$, $(a, \cdot, \cdot, \cdot)$.

The representation of $H(L)$ as the union of set forms is quadratic in k, i.e. up to $\binom{(k+1)}{2} = k(k+1)/2$ different set forms are necessary. Of course, for a label L with less than k predecessors (i.e. $pred^k(L) = \perp$), $H(L)$ needs fewer set forms. Also, a set form can be discarded from the representation of $H(L)$ if it corresponds to a set of paths that is already included by another set form in $H(L)$, e.g. $(a, \cdot, b)$ encodes a subset of $(a, \cdot, \cdot)$ and can therefore be discarded.

Let L be a label resident at node v and let $L_1, \ldots, L_t$ be a collection of labels all resident at the same node $v = v(L_i)$ for which $L_i \prec_{dom} L$. Lemma 1 says that L does not need to be considered for constructing the set $PO(u)$ for any $u \in V$ if all its possible extensions are covered by the union of the possible extensions of labels L_i, $i \in \{1, \ldots, t\}$. The existence of a set of t labels satisfying the above criterion implies that the union can also be taken over all the labels $L_i \prec_{dom} L$, i.e. the specific subset of t labels is irrelevant to the validity of the lemma. From de Morgan's law it follows that

$$\mathcal{E}(L) \subseteq \bigcup_{L_i \prec_{dom} L} \mathcal{E}(L_i) \iff \bigcap_{L_i \prec_{dom} L} H(L_i) \subseteq H(L).$$

We can therefore implement the dominance criterion defined by Lemma 1 using our self-hole sets, for which we have to provide the implementation of the intersection and subset operations.

6.2. Intersection of Self-Hole Sets

In this section, we present an algorithm to compute the intersection of two collections of set forms $\{s^1, \ldots, s^p\}$ and $\{t^1, \ldots, t^q\}$, corresponding to the intersection of the self-hole sets associated to two labels. Its complexity (with respect to time and space) is bounded by $\mathcal{O}(kpq)$. We take care not to keep in the resulting collection any set form that explicitly forms a k-cycle.

We recommend storing each hole set as a *list* of set forms. The check as to whether a set form $s = (s_1, \ldots, s_k)$ is *included* in the set form $t = (t_1, \ldots, t_k)$ can be performed in $\mathcal{O}(k)$ time. For each position $i = 1, \ldots, k$ the condition $t_i \in \{\cdot, s_i\}$ must be satisfied. For example, $(\cdot b \cdot acd) \subseteq (\cdot b \cdots d)$ but $(\cdot b \cdot acd) \nsubseteq (\cdot b \cdot \cdot ad)$ and $(\cdot b \cdot acd) \nsubseteq (\cdot b \cdot ecd)$.

The intersection $s \cap t$ of two set forms $s = (s_1, \ldots, s_k)$ and $t = (t_1, \ldots, t_k)$ can either be the empty set $\varnothing$ or a new set form $u = (u_1, \ldots, u_k)$. Both cases can be computed in $\mathcal{O}(k)$ amortized time. The subtle problem is to identify a k-cycle in $\mathcal{O}(k)$ time, i.e. identical nodes $\neq \cdot$ at different positions in s and t resulting in $s \cap t = \varnothing$. Therefore, a field $f[\ ]$ indexed by the characters in V with values $\{0, 1\}$ has to be initialized once with $f[w] := 0$ for all $w \in V$. For each position $i = 1, \ldots, k$ three different cases have to be considered. First, if $\{s_i, t_i\} = \{w, \cdot\}$ for a node $w \in V$ then set $u_i := w$ and invert $f[w] := 1 - f[w]$. If the result is $f[w] = 0$ then a k-cycle is found and $s \cap t = \varnothing$. The two other cases are $u_i := \cdot$ if $\{s_i, t_i\} = \{\cdot\}$ and $s \cap t = \varnothing$ if $\cdot \neq s_i \neq t_i \neq \cdot$. After the computation of $s \cap t$ one has to reset the field $f[\ ]$ by setting $f[s_i] := 0$ and $f[t_i] := 0$ for each $i \in \{1, \ldots, k\}$. The overall complexity of h intersection operations is $\mathcal{O}(|V| + h \cdot k)$, i.e. $\mathcal{O}(k)$ amortized time for a single intersection operation. Note further that the resulting intersection $u := s \cap t$ is the empty set if there exists a position $i \in \{1, \ldots, k\}$ with $(u_i, u_{i+1}) \notin A$. If the digraph is represented by an adjacency matrix this check can be performed in constant time; for a representation with adjacency lists, the check requires $\mathcal{O}(|V|)$ effort.

Algorithm 2 in Figure 3 for computing the intersection of hole sets consists of three blocks: one to identify set forms in different hole sets including each other, a second to perform the intersection on all remaining pairs, and a third to check for inclusion in the resulting collection of set forms.

Steps 3–6 are included for the purpose of accelerating the subsequent steps, i.e. to insert fewer set forms s and t into the preliminary *result* set with $s \subseteq t$. In both cases (with or without steps 3–6), the overall complexity of Algorithm 2 is bounded by $\mathcal{O}(kpq)$.

Computing the intersection of more than two hole sets can be done iteratively. In the course of the dominance step of Algorithm 1, the result of these intersections needs to be

```
Algorithm 2 (* Intersection of hole sets *)
1:  (INPUT:) two hole sets encoded as two collections $\{s^1, \ldots, s^p\}$ and $\{t^1, \ldots, t^q\}$ of set forms.
2:  (Initialization)
    let $H_1 := \{s^1, \ldots, s^p\}$, $H_2 := \{t^1, \ldots, t^q\}$, $result := \varnothing$.
3:  (Check for inclusion in input sets)
    for all $s \in H_1$ and all $t \in H_2$ do
4:      if $s \subseteq t$ (resp. $t \subseteq s$) then
5:          let $result := result \cup \{s\}$ (resp. $result := result \cup \{t\}$).
6:          let $H_1 := H_1 \setminus \{s\}$ (resp. $H_2 := H_2 \setminus \{t\}$).
7:  (Compute intersection)
    for all $s \in H_1$ and all $t \in H_2$ do
8:      if $s \cap t \neq \varnothing$ then
9:          let $result := result \cup \{s \cap t\}$.
10: (Check for inclusion in result set)
    for all $\{s, t\} \subseteq result$, $s \neq t$ do
11:     if $s \subseteq t$ (resp. $t \subseteq s$) then
12:         let $result := result \setminus \{s\}$ (resp. $result := result \setminus \{t\}$).
13: (OUTPUT:) $result$.
```

Figure 3: Computing the Intersection of Hole Sets

"accumulated" separately for each potentially discarded label. We define $H^{run}(L)$ as the *running-hole set* of label L, which starts as $\mathcal{S}$ and is reduced by intersection with each self-hole set $H(L_i)$ of each label $L_i \prec_{dom} L$ considered during the dominance step. Using running hole sets, a label L can be discarded as soon as $H^{run}(L) \subseteq H(L)$.

Let $H^{run}(L) = \{s^1, \ldots, s^p\}$ be the current running-hole set of label L and $H(L) = \{t^1, \ldots, t^q\}$ be its self-hole set. Checking whether $H^{run}(L) \subseteq H(L)$ holds can be performed using a one-by-one subset comparison of set forms s^i with the set forms t^j.

It is easy to see that there exists a more efficient way of checking $H^{run}(L) \subseteq H(L)$. The running-hole set of a label L just after its creation is represented by the single set form $(\cdot \ldots \cdot)$. Since $H(L)$ does not change in the course of the algorithm, the check for $s^i \subseteq t^j$ has to be performed only when set forms s^i of $H^{run}(L)$ are created or modified. Whenever we find $s^i \subseteq t^j$, the set form s^i is obviously included in $H(L)$. Furthermore, any intersection operation undertaken to modify $H^{run}(L)$ can only replace s^i by one or several subsets of s^i or eliminate s^i. In these cases, the result of the intersection is still a subset of $H^{run}(L)$. Therefore, set forms s of $H^{run}(L)$ with $s \subseteq H(L)$ can be eliminated. This processing, i.e. to check a set form s of $H^{run}(L)$ directly after its creation and possibly eliminating it, creates a *modified running-hole set* $H^{mod-run}(L) = H^{run} \setminus H(L)$.

As a result, checking $H^{run}(L) \subseteq H(L)$ reduces to checking $H^{mod-run}(L) = \varnothing$. Moreover, all the labels resident at a given node v will have set forms forbidding the occurrence of node v at any of the next k positions, and that node v is the only node that forces us to encode set forms using k components. By using $H^{mod-run}(L)$ to accumulate intersections

Table 1: Intersection of Self-Hole Sets of Two Labels, $k = 3$

label L_1 with $(pred^2(L_1), pred^2(L_1))$	label L_2 with $(pred^2(L_2), pred^1(L_2))$	set forms in $H(L_1) \cap H(L_2)$
(ab)	(ab)	$\{(a\cdot), (b\cdot), (\cdot b)\}$
(ab)	(ba)	$\{(a\cdot), (b\cdot)\}$
(ab)	(ac)	$\{(a\cdot), (bc), (cb)\}$
(ab)	(bc)	$\{(ac), (b\cdot), (cb)\}$
(ab)	(cb)	$\{(b\cdot), (\cdot b)\}$
(ab)	(ca)	$\{(a\cdot), (ba), (cb)\}$
(ab)	(cd)	$\{(ad), (bd), (cb), (db)\}$

Note. The intersection of the self-hole sets of two labels L_1 and L_2 depends on the predecessors $pred(L_1)$ and $pred(L_2)$. All nodes $a, b, c, d \in V$ are assumed to be different.

of hole sets, we can therefore dispense with encoding any set form related to the resident node, and we can use vectors of only $k-1$ elements to encode all the relevant dominance information. For the remainder of the paper, we will use this quadratic encoding with only $k(k-1)/2 = \binom{k}{2}$ set forms.

For example, in the case of $k = 3$, if $H(L_1) = \{(a\cdot), (b\cdot), (\cdot b)\}$ and $H(L_2) = \{(c\cdot), (d\cdot), (\cdot d)\}$ with four different nodes $a, b, c, d \in V$, then $H(L_1) \cap H(L_2) = \{(ad), (bd), (cb), (db)\}$. Table 1 shows all different cases of intersections of the self-hole sets of two labels L_1 and L_2 depending on the predecessor vectors $pred(L_1)$ and $pred(L_2)$. It is evident that for k-cycle elimination with $k > 3$ the number of different cases grows rapidly.

6.3. Upper Bound on the Number of Set Forms in an Intersection of Hole Sets

The following theorem is a key result for bounding the increase in complexity that comes from the adjunction of k-cycle elimination constraints to SPPRC. We henceforth use the fact that only $k-1$ elements are necessary to encode set forms. In order to simplify the notation we define $\bar{k} := k-1$ and $\bar{K} := \{1, 2, \ldots, k-1\}$.

Theorem 1 *The maximum number of different set forms needed to represent any intersection of hole sets $H(L_1) \cap H(L_2) \cap \cdots \cap H(L_p)$ of any set of p labels $L_1, \ldots, L_p$ is $(k-1)!^2$. This bound is tight in the sense that one can construct a set of $k-1$ labels $\{L_1, \ldots, L_{k-1}\}$ needing $(k-1)!^2$ different set forms to describe the intersection of their hole sets.*

Proof: First of all, it is helpful to define the *type* $I(s)$ of an arbitrary set form

$s = (s_1, \ldots, s_{\bar{k}}) \in (V \cup \{\cdot\})^{\bar{k}}$:

$$I(s) := \{i \in \bar{K} : s_i = \cdot\} \subseteq \bar{K}.$$

(For example, $I(\cdot a \cdot c) = \{1, 3\}$ and $I(adcb) = \varnothing$.)

In the following, we want to describe the *maximum number of different set forms* that can be generated from a given set form s by intersecting it with several (arbitrarily chosen) hole sets $H(L_1), H(L_2), \ldots$. It is obvious that this maximum number of different set forms created from s is determined by the type $I = I(s)$ of s. Therefore, we denote by $n_k(I)$ the *maximum number of different set forms that can be generated from a set form of type I by intersection with arbitrarily chosen holes sets.* n_k is defined on all subsets $I \subseteq \bar{K}$. The following recurrences are valid for n_k:

$$\begin{aligned} n_k(\varnothing) &= 1; \\ n_k(I) &= \sum_{i \in I} (k-i) \cdot n_k(I \setminus \{i\}) \quad \text{for all } \varnothing \neq I \subseteq \bar{K}. \end{aligned}$$

The first equation is trivial. The second equation is implied by the intersection operation, i.e. at the ith position of the intersection, one can only place those predecessor $pred^j(L)$ having $j \geq i$. For each position i, there are $k-i$ different possibilities. (For example, when intersecting $s = (\cdot b \cdot a)$ with a hole set $H(L)$ then there are $k-1=4$ possibilities to place predecessors of L at position 1, and $k-3=2$ possibilities to place them at position 3.)

This recurrence is solved by

$$n_k(I) = |I|! \prod_{i \in I} (k-i). \tag{5}$$

The following proof is by induction on the cardinality of I. For $I = \varnothing$, (5) gives $n_k(\varnothing) = 1$, which is correct. Now assume that (5) is true for all subsets of $\bar{K}$ with cardinality $|I|-1$. It follows

$$\begin{aligned} n_k(I) &= \sum_{i \in I} ((k-i)\, n_k(I \setminus \{i\})) = \sum_{i \in I} \left((k-i) \left((|I|-1)! \prod_{j \in I \setminus \{i\}} (k-j) \right) \right) \\ &= \left(\sum_{i \in I} (|I|-1)! \right) \left((k-i) \prod_{j \in I \setminus \{i\}} (k-j) \right) = |I|! \left(\prod_{j \in I} (k-j) \right). \end{aligned}$$

This proves (5). In particular, we get $n_k(\bar{K}) = \bar{k}!\bar{k}! = \bar{k}!^2$. This means that we can generate at most $\bar{k}!^2$ different set forms starting from the trivial set form $(\cdot \ldots \cdot)$ by intersecting with several hole sets $H(L_1), H(L_2), \ldots$.

To show that the bound $\bar{k}!^2$ is tight, we can choose any $\bar{k}$ different labels $L_1, \dots, L_{\bar{k}}$ and predecessor vectors $pred(L_i)$ with completely disjoint node sets, i.e. $\bar{k}^2$ different nodes $pred^j(L_i)$ for $i, j \in \{1, \dots, \bar{k}\}$. The intersection $H(L_1) \cap H(L_2) \cap \dots \cap H(L_{\bar{k}})$ consists of exactly $\bar{k}!^2$ different set forms s and each one is of type $I(s) = \varnothing$. $\diamond$

6.4. Upper Bound on the Number of Labels with Identical Resource Vectors

In this section, we show that the contribution of a self-hole set to a running-hole set cannot be split in arbitrarily fine pieces. There exists an upper bound $\alpha(k)$ on the number of labels that can properly contribute to the intersections of hole sets that are computed as the running-hole sets of dominated labels. In particular, the number of useful labels with identical resource vector is also bounded by $\alpha(k)$.

Given any collection of labels $L_1, \dots, L_q$ with identical resource vectors $res = res(L_i)$ and $v = v(L_i)$, $i = 1, \dots, q$, this collection can be (uniquely) sorted according to the adopted resource-dominance rule, i.e.

$$L_1 \prec_{dom} L_2 \prec_{dom} \dots \prec_{dom} L_q,$$

and any other label L^{dom} dominating L_i will also dominate L_{i+1}. Therefore, their running-hole sets fulfill

$$H^{run}(L_1) \supseteq H^{run}(L_2) \supseteq \dots \supseteq H^{run}(L_q). \tag{6}$$

By the definition of the running-hole set, we see that $H^{run}(L_{i+1})$ is identical to $H(L_i) \cap H^{run}(L_i)$. Therefore, the condition $H^{run}(L_{i+1}) = H^{run}(L_i)$ is equivalent to $H(L_i) \cap H^{run}(L_i) = H^{run}(L_i)$, i.e. $H^{run}(L_i) \subseteq H(L_i)$. We see that L_i can be discarded if $H^{run}(L_i) = H^{run}(L_{i+1})$, with $i < q$. Equivalently, instead of $H^{run}(L_{i+1})$ we might consider the intersection of self-hole sets of all dominating labels with $H(L_{i+1})$. Therefore, L_t can be discarded if $I_t = I_{t-1}$ where I_t is defined by $I_t := H^{run}(L_1) \cap \left(\bigcap_{i=1}^{t} H(L_i)\right)$.

The preceding analysis shows that we can remove some of the labels $L_1, \dots, L_q$ with identical resource vectors within the chain (6). Since $H^{run}(L_j)$ and $H^{run}(L_{j+1})$ (or equivalently I_{j-1} and I_j) differ only by a single intersection with one self-hole set (i.e. $H(L_j)$), we will study *longest chains of intersections of running-hole sets with self-hole sets.* Clearly, the maximum length of a proper chain of intersections of hole sets bounds the maximum number of labels with identical resource vectors that we have to consider.

Theorem 2 *Let $L_1, L_2 \ldots, L_s$ be labels with self-hole sets $H(L_1), H(L_2), \ldots, H(L_q)$ with the property that their intersections*

$$I_t := \bigcap_{i=1}^{t} H(L_i)$$

for $t = 1, \ldots, q$ defines a properly decreasing chain $I_1 \supset I_2 \supset \ldots \supset I_q$. Then, the length q of this chain is bounded by $\alpha(k)$, which depends on k but not *on the size of the graph or the width of the resource windows.*

The bound $\alpha(k) = k(k-1)!^2$ is a valid upper bound. This bound $\alpha(k)$ is not always tight.

Proof: We assume that $I_1 \supset I_2 \supset \cdots \supset I_q$ is a decreasing chain of hole sets constructed by intersection operations with some self-hole sets $H(L_1), H(L_2), \ldots, H(L_q)$. Consecutive intersections I_{j-1} and I_j are different and their difference is caused by a single intersection with the self-hole set $H(L_j)$, i.e. $I_j = I_{j-1} \cap H(L_j)$.

By Theorem 1, each intersection I_j consists of r_j different set forms $s_j^1, \ldots, s_j^{r_j}$ with $1 \leq r_j \leq (k-1)!^2$. Therefore, a maximum of $(k-1)!^2$ different set forms each having $k-1$ components gives rise to $(k-1)(k-1)!^2$ different components within these set forms. The intersection operation $I_{j-1} \cap H(L_j)$ performs modifications on the set forms $s_{j-1}^1, \ldots, s_{j-1}^{r_{j-1}}$ of I_{j-1} to compute the set forms of I_j. It either
(i) generates one or more additional set forms that are constructed from existing set forms of I_{j-1} by replacing a component with entry "$\cdot$" by a value $v \in V$, or
(ii) removes one or more set forms of I_{j-1}.
The number of operations of type (i) is, therefore, bounded by the maximum number of components, i.e. $(k-1)(k-1)!^2$, while the number of operations of type (ii) is obviously bounded by the maximum number of different set forms, i.e. $(k-1)!^2$. Each intersection performs at least one operation of type (i) or (ii). Altogether, this yields an upper bound of $\alpha(k) = (k-1)(k-1)!^2 + (k-1)!^2 = k(k-1)!^2$. $\diamond$

Remark 1 For the cases of $k = 2, 3$, we can show that the bounds $\alpha'(2) = 2$ and $\alpha'(3) = 6$ are tight. The bound for $k = 3$ was computed using an enumeration procedure to compute all possible chains of intersection of self-hole sets of labels. The result is shown in Figure 4. The longest chain of intersections consists of $k! = 6$ non-empty hole sets. The intersection graph is defined as a graph in which each node corresponds to an intersection I_j (i.e. a *type*

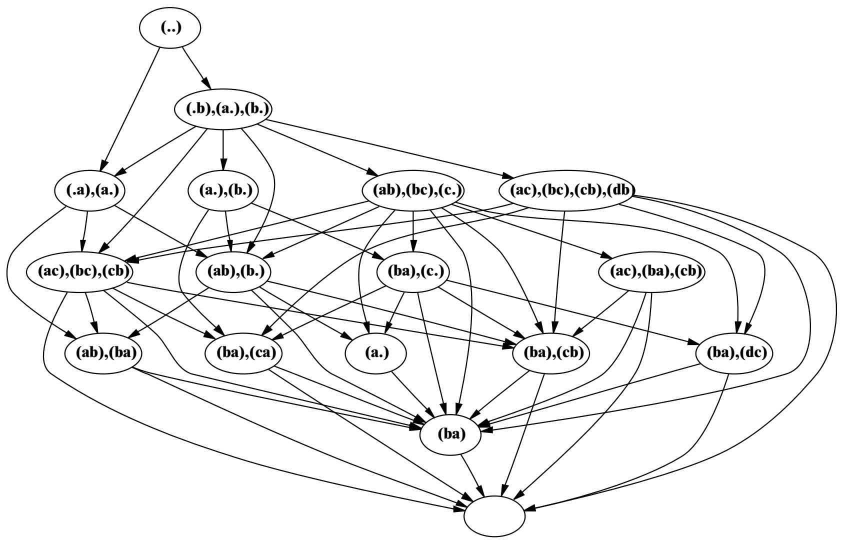


Figure 4: Intersections of Hole Sets with Self-Hole Sets for $k = 3$

of running-hole set $H^{run}(L)$) and arcs $(H^{run}(L^1), H^{run}(L^2))$ exist if there is a self-hole set $H(L)$ such that $H^{run}(L^2) = H^{run}(L^1) \cap H(L)$. In order to avoid many symmetric cases, running-hole sets that differ only by a permutation of the node names are depicted only once. Therefore, nodes of the graph represent only *types of running-hole sets* and node names $a, b, c, \ldots$ possibly have to be permuted. For example, the arc $(\{(ba), (c.)\}, \{(a.)\})$ corresponds to the fact that the running-hole set $H(L) = \{(ba), (c.)\}$ can be intersected with $H(L) = \{(cd)\}$ resulting in the running-hole set $\{(c.)\}$ that is of the same type as $H(L^2) = \{(a.)\}$.

For the general case of $k > 4$, we conjecture that $\alpha'(k) = k!$ is a valid and tight upper bound. It is easy to construct chains of labels with different predecessor vectors that generate a chain of $k!$ properly decreasing intersections. So, if $\alpha'(k) = k!$ is a valid bound, then it is also tight. A proof of this conjecture is still an open problem.

There exist examples showing that, by ordering the labels $L_1, \ldots, L_t$ with identical resource vectors in different ways, one might obtain sets of undominated labels of different cardinality. However, we do *not* try to adapt the ordering (find a good permutation) of labels with identical resource vectors to produce small or minimum-cardinality label sets, for the following two reasons. On the one hand, we might have to solve a set-covering problem to find one label set of minimum cardinality. On the other hand, performing operations of complexity $\Omega(n \log n)$ (where n is the number of labels) is already computationally too costly.

6.5. Strongly, Semi-Strongly, and Weakly Dominant Labels in k-Cycle Elimination

Kohl's definitions of *(semi-)strongly* and *weakly dominant* labels from Section 5.3 can be extended to the k-cycle elimination case with $k \geq 3$. The extension is based on a modification of the definition of the *self-hole set* $H(L)$ of a label L.

A set form $s = (\cdot \ldots \cdot w \cdot \ldots \cdot)$ in $H(L)$ represents a set of forbidden extensions of label L. The extensions of label L along each path included in s would involve a minimum resource consumption at the terminal node w. The idea of strongly dominant labels is that their resource consumption along all paths of the form s is too big, i.e. the extension does not result in a new resource-feasible label. In this case, the set form s is not necessary to represent $H(L)$ because neither L nor a label dominated by L can be feasibly extended along any path in s.

To make the idea and notation more precise, let the set form $s = s(q, w)$ of $H(L)$ have the entry $w \in V$ at position q, $1 \leq q \leq \bar{k}$. Let the label L be resident at node v, let $Q = (w_1, \ldots, w_q)$ be an arbitrary path of length $q - 1$ with last node $w_q = w$ and let $P(L, Q) = (P(L), Q)$ be the concatenation of the path corresponding to label L and path Q. Furthermore, let $r^* \in \{1, \ldots, R\}$ be any resource and $T^{r*}_{P(L,Q)}$ be the minimum resource consumption of path $P(L, Q)$ computed by (2). If for a set form $s = s(q, w)$

$$\min\{T^{r*}_{P(L,Q)} : Q \text{ path of length } q-1 \text{ with last node } w\} > b^{r^*}_w \tag{7}$$

then there exist no feasible extensions of L along paths included in s. As a consequence, s is *unnecessary*, i.e. s might be removed from the description of $H(L)$. Let $H^{mod-self}(L)$ denote the *modified self-hole set* of label L constructed from the self-hole set $H(L)$ by removing all unnecessary set forms.

For the case $k = 2$, the above inequality (7) has a simplified formulation given by condition (4). In general, the condition (7) involves a huge set of paths and it is computationally too costly to be checked. Instead of this, we propose use of a weaker condition. For resources r^* with non-decreasing resource consumption $t^{r^*}_{ij} > 0$, we a priori compute the minimum resource consumption $\bar{t}^{r^*}_{vw}$ between each pair of nodes v, w (the nodes v, w are not necessarily connected through an arc $(v, w) \in A$). Instead of condition (7), we check $res(L)_{r^*} + \bar{t}^{r^*}_{vw} > b^{r^*}_w$.

Summing up, the dominance rules of this section and the modification of the self-hole set $H^{mod-self}(L)$ allow the following interpretations:

- *Strongly dominant* labels L^{dom} are those undominated labels having $H^{mod-self}(L^{dom}) = \varnothing$. If any other label L is dominated by a strongly-dominant label L^{dom} it can be discarded (because $H^{run}(L) \subseteq H^{mod-self}(L^{dom}) = \varnothing$ implies $H^{run}(L) \subseteq H(L)$).
- *Semi-strongly dominant* labels L^{dom} are those undominated labels that fulfill $H^{mod-self}(L^{dom}) \neq \varnothing$.
- *Weakly dominant* labels L are dominated labels with $H^{run}(L) \nsubseteq H^{mod-self}(L)$.

From the dominance rules of Section 6.1 it follows that all other labels, which are not strongly dominant, semi-strongly, or weakly dominant, can be discarded.

7. Extensions

In this short section, we present three possible extensions to our definition of SPPRC-k-cyc, that can be accomodated with few modifications to our algorithm: *Non-positive resource consumptions* $t^1_{ij} \leq 0$, the generalization of the resource update $T^i := \max\{a_{v_i}, T_{i-1} + t_{v_{i-1},v_i}\}$ by *resource extension functions* and task-cycle elimination with *sequences of tasks associated to nodes and arcs.*

7.1. Non-Positive Resource Consumption

The assumption that all resource consumptions t^1_{ij} for $(i,j) \in A$ are strictly positive (see Section 2) enables the use of a label-setting algorithm for treating labels. In fact, the weaker condition that $t_{ij} \succ_{lex} 0$ is readily supported by our algorithm. This hypothesis allows label-setting algorithms to process labels in increasing order. In our labeling algorithm, Algorithm 1, the choice of labels with a "small" resource consumption $res(L)^1$ under all unprocessed labels L with the above assumption guarantees that all labels that are extended (steps 12–14) are never identified as discardable in later iterations. In essence, the defining property of label-setting algorithms is that those labels chosen to be extended are kept without modification until the end of the labeling process.

The general ideas of *label-setting* as well as *label-correcting algorithms* are, for instance, explained in Ahuja et al. (1993). Label-correcting algorithms are applicable to shortest-path instances where negative arc lengths occur. The presence of non-positive resource consumptions t^r_{ij} for all resources $r = 1, \ldots, R$ does not affect the validity of the dominance rules (presented in Section 6). The existence of a negative t^1_{ij} means that the strategy of treating labels in a strictly increasing order has to be replaced by a more flexible label-processing strategy. The well-known Bellman-Ford label-correcting algorithms for the one-dimensional SPP adds newly generated labels to the end of a queue (data structure) and expands labels L one by one starting with the label currently at the top of the queue. A more sophisticated generalized label correcting strategy for the SPPRC was used in Powell and Chen (1998) and is directly applicable for the SPPRC-k-cyc label processing.

7.2. Resource-Extension Functions

Desaulniers et al. (1998) point out that resource-extension functions (REFs) allow more flexible modeling of the dependencies between resource variables T_i^r than what is possible with the simple update formula (2), i.e. $T_j := \max\{a_j, T_i + t_{ij}\}$, when arc (i, j) is used. Recall that resource vectors at nodes i and j are denoted by $T_i = (T_i^1, \dots, T_i^R)$, resp. $T_j = (T_j^1, \dots, T_j^R)$. Desaulniers et al. discuss REF as functions of the form $f_{ij}^r(T_i)$ defined for each arc $(i, j) \in A$ and each resource $r = 1, \dots, R$. REF may be linear or nonlinear. The resource-update formula (2) is then replaced by

$$T_j^r \geq f_{ij}^r(T_i) = f_{ij}^r(T_i^1, \dots, T_i^R), \tag{8}$$

which means that the resource variable T_j^r might depend on the values of all the resource variables of node i. Practically relevant examples include load-dependent cost functions Dumas et al. (1991), the VRP(TW) with simultaneous pickups and deliveries Min (1989), and various applications in airline and crew scheduling.

Furthermore, it is also shown in Desaulniers et al. (1998) that for the case of *non-decreasing REF* (including all resources, in particular the cost resource) the dominance principle of SPPRC remains valid. For two given resource vectors $T_i \leq T_i'$ at node i, the extension along arc (i, j), computed by (8) yields two resource vectors T_j and T_j' with $T_j \leq T_j'$.

In this sense, the dominance rules of Section 6 and the SPPRC-k-cyc algorithm are fully compatible with the use of non-decreasing REF.

7.3. k-Cycle Elimination for Task Sequences Associated to Nodes and Arcs

Let $\mathcal{T}$ be a *set of tasks*. Associate a sequence S_i, resp. S_{ij}, of tasks to each node $i \in V$, resp. arc $(i, j) \in A$. Node-cycle elimination is just a special case of task-cycle elimination where the task set coincides with the node set and each sequence S_i consists of only node i, i.e. $\mathcal{T} = V$, $S_i = (i)$ for all $i \in V$, and $S_{ij} = ()$ for all $(i,j) \in A$. For each path P, the concatenation of the node and arc task sequences naturally defines a corresponding task sequence $S(P)$. The SPPRC with *task-k-cycle elimination* consists of finding the set of all pareto-optimal resource vectors $res(P)$ paths P without sequences of tasks containing k-cycles.

Only slight modifications of the definitions given in Section 6 are necessary to handle this extension. The set $\mathcal{E}(P)$ of possible extensions of path P has to take task sequences into account. The self-hole set $H(L)$ of a label L resident at node v has to consider the last k tasks implied by the task sequences associated to the predecessor arcs and nodes of $P(L)$. All labels resident at node v share the last $|S_v|$ tasks. Because of this, the encoding of set forms needs to use vectors of length $k - |S_v|$, compared to $\bar{k} = k - 1$ when node cycles are forbidden. Finally, the extension step of the SPPRC-k-cyc algorithm has to check for k-cycles of tasks instead of nodes.

The main scope of application for eliminating k-cycles of tasks are *aggregated networks* and *discretized networks*. In the first case, parts of an original network $G_o = (V_o, A_o)$ are shrunken together yielding a smaller, aggregated network $G = (V, A)$. A single node of G might correspond to a set of nodes and arcs in G_o and, therefore, node-cycle elimination in G_o is equivalent to task-cycle elimination in G.

In a discretized network, several nodes $i_1, \ldots, i_p \in \mathit{copy}(i)$ correspond to a single node i of an original network. For instance, for the discretization of the SPPTW, nodes $i \in V$ are duplicated into $p = b_i - a_i + 1$ nodes according to the time windows $[a_i, b_i]$ to represent the distinct points in the resource space. In order to eliminate k-cycles, the task set $\mathcal{T}$ consists of the nodes of the original network and each node $i_p \in \mathit{copy}(i)$ of the discretized network has the associated task sequence $S_{i_p} = (i)$.

8. Application: Lower Bounds for Vehicle-Routing Problems with Side Constraints

This section illustrates application of the new SPPRC-k-cyc algorithm as a subproblem solver within a branch-and-price algorithm. One of the most prominent vehicle routing problems with side constraints is the VRPTW. Solomon's (1987) benchmark VRPTW instances have attracted numerous researchers to develop both exact and heuristic solution procedures. For a comprehensive review of VRPTW heuristics see, e.g., Rochat and Taillard (1995), Chiang and Russell (1997), Gambardella et al. (1999), and Taillard et al. (2001).

For our implementation of improving lower bounds in exact branch-and-price methods for the VRPTW, we have chosen the same setup as in Kohl (1995), Larsen (1999), Rich (1999), Kohl et al. (1999), and Kallehauge et al. (2001), i.e. 168 instances with 25, 50, and 100 customers where time and distances are rounded with precision 0.1. The objective is to minimize the total cost, i.e. the traveled distance. We use several standard techniques to improve a standard branch-and-price approach for VRPTWs that have been published, e.g. pre-processing Desrochers et al. (1992), 2-path cuts Kohl (1995), Kohl et al. (1999), and nearest-neighbor networks for *partial pricing* Gamache et al. (1999), in Larsen (1999) called *limited subsequence*).

In the following short computational analysis, the values $lb_1(k)$ for $k \geq 2$ describe the lower bound of the VRPTW instance given by the solution of the LP-relaxation of the master program when k-cycle elimination is applied (i.e. the solution of the root node of the search tree). The value $lb_1(1)$ is the lower bound implied by solving the subproblem as a plain SPPRC. The integrality gap $[lb_1(1), opt]$ is given with respect to this value and the optimal solution opt. After adding the 2-path cutting planes, the improved lower bound is denoted as $lb_2(k)$ (cutting planes are added only to the restricted master program at the root node).

8.1. Three Examples

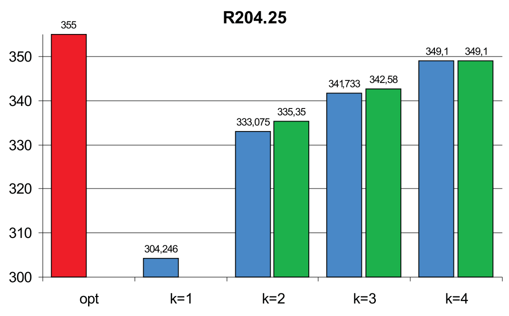
Figures 5(a)–(c) depict the increasing lower bounds for three selected instances (r204.25, r209.50, and rc205.100). The value $lb_1(k)$, resp. $lb_2(k)$, is the lower bound before, resp. after, adding 2-path cuts to the master problem (first bar, resp. second bar, in each section of the diagram).

Part (a) shows the results for the instance R204 with 25 customers. With increasing values of $k = 2, 3, 4$, the size of the branch-and-bound tree and the CPU times decrease to trees with 622, 104, and 35 nodes, resp. 3524, 385, and 123 CPU seconds.

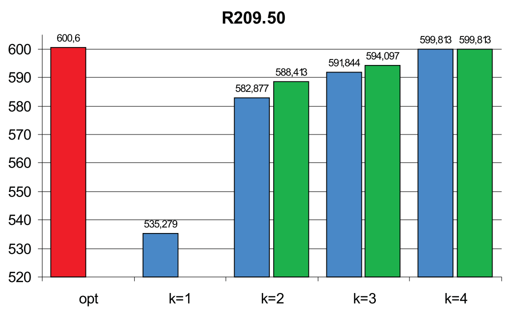
Part (b) shows the results for the instance R209 with 50 customers, which was previously unsolved with 2-cycle elimination. It is now solved by 3-cycle elimination with a search tree of 59 nodes in 703 seconds. With 4-cycle elimination the same problem instance has a reduced search tree with six nodes and can be solved in 142 seconds CPU time.

Part (c) shows the results for the instance RC205 with 100 customers. This instance has only been solved with 5-cycle elimination.

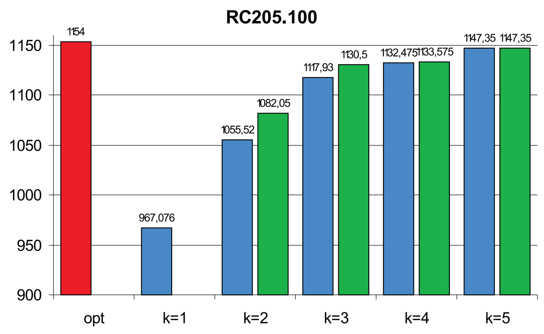
For these cases, k-cycle elimination for $k \geq 3$ obviously helps to increase drastically the lower bound. It leads to significantly smaller branch-and-bound search trees and, therefore,



(a)



(b)



(c)

Figure 5: Lower Bounds of the LP-Relaxation of the Master Program.

reduced overall computation times. These and all other computations (see Table 2 and Section 8.2) were performed on a standard PC with an Intel Pentium III processor at 600MHz with 512MB of main memory. The algorithm was coded in C++ and the callable library of CPLEX 7.0 CPLEX (1997) was used to solve the restricted master problem (RMP).

8.2. Summary of an Extended Computational Test

Detailed computational results for all 168 Solomon problem instances can be found in the Online Supplement to this paper on the journal's website. It contains a comparison of the lower bounds $lb_1(k)$ (and $lb_2(k)$) for $k \in \{2, 3, 4\}$, which are the result of solving the RMP without (with) adding 2-path cuts at the root node. The corresponding computing times $T(k)$ give the time needed to solve an instance to proven optimality (the computation time for each instance was restricted to one hour (3600s) CPU time). An analysis of the detailed computational test can be aggregated to the following characteristic numbers:

Table 2: Optimal Solutions for Previously Unsolved VRPTW Instances

Instance	Distance	#vehicles	k-cycle	Tree	Time (in s)
r104.100	971.5	11	3	5396	268106.0
rc104.100	1132.3	10	3	6757	986809.0
rc107.100	1207.8	12	3	1493	42770.7
rc108.100	1114.2	11	3	707	71263.0
c204.100	588.1	3	2	12	54254.4
r203.50	605.3	5	3	23	470.4
r204.25	355.0	2	4	35	231.7
r204.50	506.4	2	4	132	23749.5
r205.50*	690.1	4	4	137	1091.4
r206.50	632.4	4	3	1615	22455.3
r208.25*	328.2	1	3	16	363.5
r209.50	600.6	4	4	7	255.4
r210.50	645.6	4	3	960	11551.4
r211.50	535.5	3	3	1972	21323.0
rc202.50	613.6	5	4	28	503.3
rc202.100	1092.3	8	5	239	124018.0
rc203.25*	326.9	2	4	297	3455.3
rc203.50	555.3	4	5	38	54229.2
rc205.50*	630.2	5	4	5	82.4
rc205.100	1154.0	7	5	65	13295.9
rc206.50	610.0	5	4	62	934.9

How many instances are solved to optimality with $k = 2$?	112
How many instances are solved to optimality with $k = 3$?	117
How many instances are solved to optimality with $k = 4$?	117
How many instances are solved to optimality with $k \in \{2, 3, 4\}$?	124
How often is $k = 3$ optimal but $k = 2$ not optimal?	7
How often is $k = 4$ optimal but $k = 3$ not optimal?	5
How often is $k = 4$ optimal but $k = 2$ not optimal?	11
How often is $k = 2$ optimal but neither $k = 3$ nor $k = 4$ optimal?	2
How often is time $T(3)$ smaller than $T(2)$?	42
How often is time $T(4)$ smaller than $T(3)$?	17
How often is time $T(4)$ smaller than $T(2)$?	41
How often is $T(3)$ or $T(4)$ smaller than $T(2)$?	47

Comparing the results for $k = 3$ and $k = 4$ against $k = 2$ shows that 3-cycle and 4-cycle elimination are successful for only some instances. "Successful" means that improved lower bounds imply significantly smaller branch-and-bound trees and that smaller trees over-compensate the higher effort to solve SPPRC with k-cycle elimination (instead of solving SPPRC without or only with 2-cycle elimination). The success for only some of the instances had to be expected because not all fractional solutions of the RMPs contain routes with cycles. There are two instances (r204.50, c203.100) for which the new algorithms ($k = 3, 4$) failed to compute the optimal solution, which are solved by using 2-cycle elimination. It seems that k-cycle elimination with $k \geq 3$ is more successful for instances with long routes than for instances with short routes and more useful when customers are not clustered.

A comparison of the computing times shows that 47 of the 124 solved instances are solved faster by 3- or 4-cycle elimination than with 2-cycle elimination. Ten instances (rc203.25, rc206.25, rc207.25, r112.5, r203.50, r205.50, r209.50, rc202.50, rc205.50, rc206.50) could only be solved by the new algorithms ($k = 3, 4$) within an hour of computing time. Five of them (r203.50, r209.50, rc202.50, rc205.50, rc206.50) were previously unsolved. The previously unsolved instance r204.25 is now solved to optimality by all of the three algorithms ($k = 2, 3, 4$). This is due to the modified branching rule.

These results show that incorporating k-cycle elimination into the subproblem solution procedure is a promising and attractive method for attacking *hard-to-solve* instances of vehicle-routing problems with side constraints. With 3- and 4-cycle elimination, more than 15 previously unsolved VRPTW instances of the Solomon benchmark set (with 25, 50 and 100 customers) could be solved to proven optimality, see Table 2. (For instances marked with * different results were reported in Cordeau et al. (2002). The problem c204.100 was solved with 2-cycle elimination mainly because of extensive partial pricing.)

9. Conclusions

In this paper, we have presented variants of the shortest-path problem with resource constraints that incorporate cycle-breaking rules to eliminate cycles of length k or less for different values of $k \geq 2$. The contribution of this paper is the development and analysis of effective, pseudo-polynomial labeling algorithms for solving the corresponding SPPRC-k-cyc problems. Solving SPPRC-k-cyc is not only interesting from a theoretical point of view. It has been shown that the controlled introduction of cycle-elimination constraints is sometimes a key technique to improve lower bounds within price-directive decomposition approaches for constrained vehicle-routing and scheduling problems where the subproblem consists of finding negative-length (elementary) paths. Increasing k, i.e., specifying the minimum length $k+1$ of the allowed cycles in the subproblem solution, produces tighter lower bounds and these are expected to lead to smaller branch-and-bound trees. We do not propose cycle-elimination as a general means to solve all vehicle-routing problems faster but as a means to solve hard instances (e.g. with wide time windows) that tend to have many cycles in their LP-relaxation. Computational results have shown that there is clearly a trade-off between avoiding longer cycles within each branch-and-bound node and solving more tree nodes. In many VRPTW cases, 3-cycle elimination or 4-cycle elimination are preferable against solving plain SPPRC, SPPRC-2-cyc, and SPPRC-k-cyc with higher values of $k > 4$. As a result, more than 15 previously unsolved instances of Solomon's VRPTW benchmark problems with 25, 50, and 100 customers could be solved to proven optimality.

All practical success of applying SPPRC-k-cyc is based on the appropriate definition of dominance rules among labels. In contrast to solving plain SPPRC and SPPRC-2-cyc, the general case requires different and refined dominance rules. It has been pointed out that dominance based on the comparison of resource vectors is not sufficient to eliminate a resource-dominated label directly. The new dominance rule among labels is based on the concept of possible extensions. Special attention has to be given to the case of labels with identical resource vectors. In order to resolve ambiguity, we have proposed refining the dominance relation as well as the lexicographical order relation to make them non-reflexive and cycle-free.

The worst-case complexity of the new dominance rule increases faster than exponentially with k, but most applications of SPPRC-k-cyc are expected to use small fixed values of k. In the context of a fixed value of k, the new dominance rule gives rise to pseudo-polynomial dominance algorithms that are iteratively called within the SPPRC-k-cyc labeling algorithm. The resulting SPPRC-k-cyc algorithms have the following remarkable property: The worst-case complexity of solving SPPRC-k-cyc for a fixed network with different values of k grows by a factor $\alpha(k)$ depending only on the cycle length k. The reason for this is that the maximum number of labels after applying the new dominance rules is at most $\alpha(k)$ times the size of the state space of the corresponding SPPRC. This factor $\alpha(k)$ is independent of the size of the network measured by $|V|$, R, and the values of the resource consumptions and bounds. From the opposite point of view, when we compare solving SPPRC-k_1-cyc and solving SPPRC-k_2-cyc ($k_1, k_2 \geq 2$) for different networks, both are of the same worst-case complexity with respect to the size of the network, i.e. $|V|$, $|S|$, and R.

References

Ahuja, R.K., T.L. Magnanti, J.B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ.

Barnhart, C., E.L. Johnson, G.L. Nemhauser, M.W. Savelsbergh, P.H. Vance. 1998. Branch-and-price: column generation for solving huge integer programs. *Operations Research* **46** 316–329.

Beasley, J.E., N. Christofides. 1989. An algorithm for the resource constrained shortest path problem. *Networks* **19** 379–394.

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

Chiang, W.-C., R.A. Russell. 1997. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing* **9** 417–430.

Cordeau, J.-F., G. Desaulniers, J. Desrosiers, M.M. Solomon, F. Soumis. 2002. VRP with time windows. P. Toth, D. Vigo, eds. *The Vehicle Routing Problem.* SIAM, Philadelphia, PA. Chapter 7, 155–194.

CPLEX 1997. Using the CPLEX Callable Library, Version 5.0. Technical report, CPLEX, Division of ILOG, Incline Village, NV.

Desaulniers, G., J. Desrosiers, I. Ioachim, M.M. Solomon, F. Soumis, D. Villeneuve. 1998. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. T. Crainic, G. Laporte, eds. *Fleet Management and Logistics.* Kluwer Academic Publisher, Boston, MA. Chapter 3, 57–93.

Desrochers, M., J. Desrosiers, M.M. Solomon. 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research* **40** 342–354.

Desrochers, M., F. Soumis. 1988. A generalized permanent labelling algorithm for the shortest path problem with time windows. *Information Systems and Operations Research* **26** 191–212.

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

Dumas, Y., J. Desrosiers, F. Soumis. 1991. The pick-up and delivery problem with time windows. *European Journal of Operational Research* **54** 7–22.

Gamache, M., F. Soumis, G. Marquis. 1999. A column generation approach for large-scale aircrew rostering problems. *Operations Research* **47** 247–263.

Gambardella, L.M., É.D. Taillard, G. Agazzi. 1999. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. D. Corne, M. Dorigo, F. Glover, eds. *New Ideas in Optimization.* McGraw-Hill, London, UK. Chapter 5, 63–76.

Guéguen, C., P.J. Dejax, M. Dror, M. Gendreau. 1998. An exact algorithm for the elementary shortest path problem with resource constraints. Technical Report G98-04-A, Laboratoire Productique Logistique, École Centrale, Paris, France (revised July 1999, also with D. Feillet as co-author).

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

Kallehauge, B., J. Larsen, O.B.G. Madsen. 2001. Lagrangean duality applied on vehicle routing with time windows – experimental results. Technical Report IMM-TR-2001-9, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark.

Kohl, N. 1995. *Exact methods for Time Constrained Routing and Related Scheduling Problems.* PhD thesis, Department of Mathematiclal Modelling, Technical University of Denmark, Lyngby, Denmark.

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

Kolen, A.W.J., A.H.G. Rinnooy-Kan, H.W.J.M. Trienekens. 1987. Vehicle routing with time windows. *Operations Research* **35** 266–274.

Kung, H.T., F. Luccio, F.P. Preparata. 1975. On finding maxima of a set of vectors. *Journal of the ACM* **22** 469–476.

Larsen, J. 1999. *Parallelization of the Vehicle Routing Problem with Time Windows.* PhD thesis, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark.

Min, H. 1989. The multiple vehicle routing problem with simultaneous delivery and pick-up points. *Transportation Research* **23** 377–386.

Powell, W.B., Z.-L. Chen. 1998. A generalized threshold algorithm for the shortest path problem with time windows. P. Pardalos, D. Du, eds. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society. 303–318.

Rich, J. 1999. *A computational study of vehicle routing applications.* PhD thesis, Department of Computational and Applied Mathematics, Rice University, Houston, Texas.

Rochat, Y., É.D. Taillard. 1995. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics* **1** 147–167.

Solomon, M.M. 1987. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research* **35** 254–265.

Taillard, É.D., L.M. Gambardella, M. Gendreau, J.Y. Potvin. 2001. Adaptive memory programming: a unified view of metaheuristics. *European Journal of Operational Research* **135** 1–16.