Resource-robust valid inequalities for set covering and set partitioning models

Ymro N. Hoogendoorn*[a] and Kevin Dalmeijer[ba]

[a] *Econometric Institute, Erasmus University Rotterdam*

[b] *H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology*

January 12, 2021

EI 2020-08

*E-mail: y.n.hoogendoorn@ese.eur.nl; Phone: +31 10 408 1264; Address: PO Box 1738, 3000DR Rotterdam, The Netherlands; Corresponding author

Abstract

For a variety of routing and scheduling problems in aviation, shipping, rail, and road transportation, the state-of-the-art solution approach is to model the problem as a set covering type problem and to use a branch-price-and-cut algorithm to solve it. The pricing problem typically takes the form of a Shortest Path Problem with Resource Constraints (SPPRC). In this context, valid inequalities are known to be 'robust' if adding them does not complicate the pricing problem, and 'non-robust' otherwise. In this paper, we introduce 'resource-robust' as a new category of valid inequalities between robust and non-robust that can still be incorporated without changing the structure of the pricing problem, but only if the SPPRC includes specific resources. Elementarity-robust and *ng*-robust are introduced as widely applicable special cases that rely on the resources that ensure elementary routes and *ng*-routes, respectively, and practical considerations are discussed. The use of resource-robust valid inequalities is demonstrated with an application to the Capacitated Vehicle Routing Problem. Computational experiments show that replacing robust valid inequalities by *ng*-robust valid inequalities may result in better lower bounds, a reduction in the number of nodes in the search tree, and a decrease in solution time.

Keywords*: Resource-Robust, Valid Inequalities, Branch-Price-and-Cut.*

Note from the authors

This document is a preliminary version of the paper that includes computational results for the Capacitated Vehicle Routing Problem. We are currently applying the ideas in this paper to other valid inequalities and other problems, and we are performing additional numerical experiments to better showcase the potential of resource-robust valid inequalities.

1 Introduction

Set covering and set partitioning models play an important role in operations research and have been used to solve a broad range of vehicle routing and crew scheduling problems (Desaulniers et al., 2005). These problems are typically solved with a branch-price-and-cut (BPC) algorithm, in which the column generation subproblem (pricing problem) is a Shortest Path Problem with Resource Constraints (SPPRC).

In this paper, we are interested in set covering and set partitioning type problems that are solved with column generation and for which the pricing problem is an Elementary SPPRC (ESPPRC), i.e., only elementary paths are allowed (Irnich and Desaulniers, 2005). Even though the ESPPRC is the main focus, results are stated more generally for the SPPRC where possible.

This ESPPRC is common for vehicle routing problems, where the elementarity stems from the requirement that clients are visited only once. We refer to Toth and Vigo (2014) for an overview of common vehicle routing variants and to Costa et al. (2019) for a recent survey on BPC algorithms for vehicle routing.

The same methodology is used to solve a variety of problems in different areas. In aviation, applications include aircraft routing (Barnhart et al., 1998a), aircraft sequencing (Ghoniem et al., 2015), and airline schedule recovery (Eggenberg et al., 2010). In shipping, the ESPPRC appears as a pricing problem in, e.g., liner shipping service design (Plum et al., 2014) and berth allocation and quay crane assignment (Vacca et al., 2013). Railway applications include timetable design and engine scheduling (Bach et al., 2015)

and integrated timetabling and crew scheduling (Bach et al., 2016).

In BPC algorithms, valid inequalities are used to strengthen the Linear Programming (LP) relaxations and thereby reduce the number of nodes in the search tree. On the other hand, valid inequalities may complicate the pricing problem. For example, the well-known clique and odd-hole inequalities for the set partitioning polytope cannot straightforwardly be used in a BPC algorithm without changing the structure of the pricing problem dramatically (Jepsen et al., 2008).

Fukasawa et al. (2006) distinguish *robust* and *non-robust valid inequalities* based on how the inequalities impact the pricing problem. Valid inequalities are called robust if adding them does not change the structure of the pricing problem. This is the case when the associated reduced costs can be projected onto the arcs of the SPPRC pricing problem. Many valid inequalities fall into this category, including strengthened comb cuts, k-path cuts, subtour elimination constraints, and rounded capacity cuts (Costa et al., 2019).

Adding non-robust valid inequalities complicates the pricing problem. However, the overall performance of the BPC algorithm may improve if non-robust valid inequalities are used with care. This is demonstrated by Jepsen et al. (2008), who introduce the subset-row inequalities and apply them to the Vehicle Routing Problem with Time Windows. Other non-robust valid inequalities are surveyed by Costa et al. (2019), including elementary cuts, strengthened capacity cuts, clique cuts, k-cycle elimination cuts, and strong degree cuts.

In this paper, we introduce a new category of valid inequalities that lie somewhere between robust and non-robust: *resource-robust valid inequalities*. Resource-robust valid inequalities can be incorporated in the BPC algorithm without changing the structure of the pricing problem, but only if the SPPRC includes specific resources. By formalizing the concept of "changing the structure of the pricing problem", we formulate conditions under which a valid inequality is resource-robust.

Of particular interest are resource-robust valid inequalities that depend on resources associated with a *state-space relaxation*. State-space relaxation is an important tool for

speeding up the pricing problem at the cost of slightly weaker LP bounds. An example is the *ng-route relaxation* introduced by Baldacci et al. (2011), which is commonly used in state-of-the-art algorithms to relax the elementarity condition in vehicle routing problems. State-space relaxations are typically implemented through resources in the ESPPRC. As such, resource-robust valid inequalities that depend on these resources are widely applicable.

To demonstrate the benefit of adding resource-robust valid inequalities, we present an application to the Capacitated Vehicle Routing Problem (CVRP). We introduce the *ng-Capacity Cuts* (*ng*CC) as a stronger alternative to the well-known rounded capacity cuts. If the *ng*-route relaxation is used, the *ng*CCs can be added without changing the structure of the pricing problem. In Section 5 we detail why this is not a completely free lunch. However, exploiting the state-space relaxation allows for significantly reducing the impact that these stronger valid inequalities have on the pricing problem.

In our computational experiments, we compare a simple BPC algorithm with and without *ng*CCs. Our experiments show that using *ng*CCs improves the average performance of the BPC algorithm on clustered instances in which the customers have a relatively high demand. This proof of concept demonstrates that using resource-robust valid inequalities can be advantageous for solving set covering and set partitioning models.

In the literature, reducing the impact of non-robust valid inequalities is studied by Pecin et al. (2017b). The authors introduce the limited-memory Subset-Row Cuts (lmSRCs), based on the subset-row cuts presented by Jepsen et al. (2008). Each lmSRC is associated with a *memory*, which is a subset of the nodes in the graph used for the SPPRC. Varying the size of the cut memory allows for balancing the strength of the cut and the impact on the pricing problem. In Pecin et al. (2017a), the cut memory of the lmSRCs is defined for arcs instead of nodes.

In relation to the work by Pecin et al., the main idea of this paper is to define cut memory based on the existing resources in the SPPRC. Valid inequalities for which this is possible do not require additional resources to keep track of the cut memory. As such, the structure of the pricing problem is not affected.

This paper is structured as follows. In Section 2, we present the preliminaries that we need for the remainder of the paper. In Section 3, we formally introduce resource-robust valid inequalities and we discuss their theoretical properties. Our application to the CVRP is detailed in Section 4, and computational results are presented in Section 5. Finally, we present our conclusions and some directions for further research.

2 Preliminaries

In this section, we briefly summarize the models, methods, and enhancements that are needed for the remainder of this paper. First, we formally introduce set covering and set partitioning models. Next, we describe a basic BPC algorithm, and we explain column generation and labeling. Finally, we discuss state-space relaxations, and in particular, the ng-route relaxation.

2.1 Set covering and set partitioning models

Let $G = (V, A)$ be a directed graph with vertices $V = \{0, 1, \ldots, n+1\}$ and arc set $A \subseteq V \times V$. Each vertex i in the set $V' = \{1, 2, \ldots, n\}$ corresponds to a *task* that has to be performed.

Tasks are performed by *routes*. A route is a path in G from 0 to $n+1$, possibly adhering to many restrictions. We say that a task is covered by a route if the route visits the corresponding vertex. In vehicle routing, the task can be to deliver a package, and the route can be the route of a vehicle. In aviation, a task may correspond to an incoming aircraft and a route may correspond to a runway schedule.

Let $\mathcal{R}$ be the set of all feasible routes. The cost of traversing an arc $(i, j) \in A$ is given by $c_{ij} \in \mathbb{R}$. Let c_r be the cost of using route $r \in \mathcal{R}$, which is the sum of the arc costs.

We consider the problem of selecting a set of feasible routes that cover all tasks at minimum cost. This problem is known as the Set Covering Problem (SCP) if tasks may be repeated, and as the Set Partitioning Problem (SPP) is every task must be performed exactly once (Garfinkel and Nemhauser, 1969). In the remainder, we restrict ourselves

to the SPP, as extending our results to the SCP is trivial.

For every route $r \in \mathcal{R}$, let the binary variable $x_r \in \{0, 1\}$ indicate whether route r is contained in the solution. Let the parameter a_i^r be equal to one if task $i \in V'$ is covered by route $r \in \mathcal{R}$, and zero otherwise. The SPP can now be expressed as follows.

$$\min \quad \sum_{r \in \mathcal{R}} c_r x_r, \tag{1a}$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}} a_i^r x_r = 1 \qquad \forall i \in V', \tag{1b}$$

$$x_r \in \{0, 1\} \quad \forall r \in \mathcal{R}. \tag{1c}$$

The Objective (1a) is to minimize the cost of the selected routes. The Constraints (1b) ensure that all tasks are performed exactly once. Note that for the SCP, these constraints are inequalities. Equations (1c) enforce that route selections are binary.

In this paper, we are interested in route sets $\mathcal{R}$ that satisfy the following two conditions. First, all routes $r \in \mathcal{R}$ are elementary, i.e., no vertices are repeated. Second, the feasibility of the routes can be modeled through *resource constraints* (Irnich and Desaulniers, 2005).

Following Irnich and Desaulniers (2005), a *resource vector* is a vector $T \in \mathbb{R}^R$ that is defined for all vertices on a route. Each of the components of a resource vectors is called a *resource*. When following the route, this vector changes according to *resource extension functions* (REFs) associated with the arcs, denoted with $f_{ij} : \mathbb{R}^R \to \mathbb{R}^R$. Feasibility of routes with respect to resources is defined with *resource windows* $[e_i, l_i]$ associated with the vertices, with $e_i, l_i \in \mathbb{R}^R$. A resource vector, and thus per extension a route, is feasible if its associated resource vector lies within the resource window (defined component-wise) for each of the visited vertices. Note that the definition of REFs allows the individual resources to influence each other, which makes resource constraints a very flexible modeling tool.

In many applications, resources do not influence each other. In that case, we refer

to the REFs of a specific resource $f_{ij} : \mathbb{R} \to \mathbb{R}$, by abuse of notation. For example, a resource may correspond to time, and the REFs ensure that time progresses. Its resource window can correspond to the hard time window in which the specific task needs to be executed.

2.2 Branch-price-and-cut algorithm

Barnhart et al. (1998b) introduce the term *brance-and-price* for branch-and-bound algorithms that rely on column generation to solve the LP relaxations. BPC algorithms additionally add valid inequalities to strengthen the LP relaxation.

In this section, we discuss column generation and labeling algorithms. Adding valid inequalities is discussed extensively in Section 3. For details on the other components, we refer to the survey by Costa et al. (2019).

2.2.1 Column generation

We refer to (1a)-(1c) as the *integer master problem*, and to its LP relaxation as the *master problem*. To effectively deal with the potentially large amount of route variables, column generation is used to solve the master problem.

In each iteration, column generation algorithms solve a *restricted master problem* (RMP), which is obtained by restricting the master problem to a subset of the variables. After solving the RMP, a *pricing problem* is solved to find variables that have a negative reduced cost and can potentially improve the current solution. A selection of these variables is added to the RMP, and the process is repeated. Once all variables have non-negative reduced costs, the RMP solution is optimal for the master problem.

The reduced cost of x_r, denoted by $\bar{c}_r$, can be expressed as follows. For all $i \in V'$, let $\lambda_i \in \mathbb{R}$ be the optimal dual values corresponding to Constraints (1b). For notational convenience, we define $\lambda_0 = \lambda_{n+1} = 0$. Let the parameter b_{ij}^r be equal to one if arc $(i, j) \in A$ is on route $r \in \mathcal{R}$, and zero otherwise. Define the *modified arc cost* of arc

$(i,j) \in A$ as $\bar{c}_{ij} = c_{ij} - \lambda_i$. It follows from LP duality that

$$\bar{c}_r = c_r - \sum_{i \in V'} a_i^r \lambda^i = \sum_{(i,j) \in A} c_{ij} b_{ij}^r - \sum_{i \in V'} a_i^r \lambda^i = \sum_{(i,j) \in A} \bar{c}_{ij} b_{ij}^r. \tag{2}$$

The pricing problem is to find a feasible route $r \in \mathcal{R}$ for which the sum of the modified arc costs is negative. Alternatively, one can find the route that minimizes $\bar{c}_r$. If the minimum is non-negative, no additional variables need to be added to the RMP.

The pricing problem can be solved as an Elementary Shortest Path Problem with Resource Constraints (ESPPRC), using the modified arc costs (Irnich and Desaulniers, 2005). This follows immediately from our assumptions on the route set $\mathcal{R}$: All routes are elementary, and feasibility of the routes can be modeled through resources constraints (Section 2.1).

2.2.2 Labeling algorithm for the ESPPRC

The ESPPRC is typically solved with a *labeling algorithm*. A labeling algorithm maintains a set of *labels* that correspond to paths that start at vertex 0. Labels are *extended* over all arcs to generate new labels, as defined by the REFs. Infeasible labels are immediately removed, and *dominance rules* are used to eliminate labels that are not Pareto-optimal. When no more labels can be generated, we have obtained a set of feasible paths from vertex 0 to vertex $n+1$. By construction, the elementary shortest path is among these paths.

A label is defined by the tuple $L = (i, \bar{c}, T, E, p)$. For label L, we refer to these components as $i(L)$, $\bar{c}(L)$, $T(L)$, $E(L)$, and $p(L)$, respectively. The vertex $i(L) \in V$ is the current endpoint of the path that belongs to L, $\bar{c}(L)$ is the reduced cost of this path up to $i(L)$, $T(L) \in \mathbb{R}^R$ is the vector of current resource values, $E(L) \subseteq V'$ is the set of visited vertices, and $p(L)$ is the predecessor label of label L. The visited vertices are used to enforce elementarity of the routes, which is why we also refer to $E(L)$ as the *elementarity resources*.

We point out that in the literature, the set of visited vertices $E(L)$ is often replaced

by the set of *unreachable vertices* (Feillet et al., 2004). That is, vertices that cannot be visited due to resource constraints are added to $E(L)$. As this modification affects the resource-robust valid inequalities presented in this paper, we discuss unreachable vertices in more detail in Section 3.4.1.

Let $L' = L \oplus j$ be the label that is generated by extending label L over the arc $(i(L), j) \in A$. To enforce elementarity, we require that $j \notin E(L)$. The label L' is given by $i(L') = j$, $\bar{c}(L') = \bar{c}(L) + \bar{c}_{i(L)j}$, $E(L') = E(L) \cup \{j\}$, $p(L') = L$ and $T(L') = f_{ij}(T(L))$. If $r(L')$ is not within the resource bounds $[e_j, l_j]$ defined at j, then the extension is infeasible, and L' is discarded.

For label dominance, we consider the common case that the REFs are non-decreasing and only upper resource windows are present (Irnich and Desaulniers, 2005). Informally, the REFs are non-decreasing if inequalities among cost and resources are preserved by arc extensions. When these assumptions are satisfied, we call the pricing problem of *non-decreasing type*.

Definition 1 (Pricing problem of non-decreasing type)**.** *The ESPPRC is of non-decreasing type if only upper resource windows are present and inequalities are preserved along label extensions. That is, for given labels L and L' with the same current vertex $i(L) = i(L')$, and for a given extension over any arc $(i(L), j) \in A$, we have that $\bar{c}(L) \leq \bar{c}(L')$, $E(L) \subseteq E(L')$ and $T(L) \leq T(L')$ imply $\bar{c}(L \oplus j) \leq \bar{c}(L' \oplus j)$ and $T(L \oplus j) \leq T(L' \oplus j)$. For the SPPRC, the elementarity resources are not considered.*

When the pricing problem is of non-decreasing type, Irnich and Desaulniers (2005) show that the *non-decreasing dominance rule* can be used.

Definition 2 (Non-decreasing dominance rule)**.** *If the pricing problem is of non-decreasing type, label L dominates label L' if all of the following are true:*

- $i(L) = i(L')$,
- $\bar{c}(L) \leq \bar{c}(L')$,
- $T(L) \leq T(L')$,

- $E(L) \subseteq E(L')$.

and at least one of the inequalities is strictly satisfied. This dominance rule is referred to as the non-decreasing dominance rule.

If label L' is dominated by some label L, then label L' cannot be part of a cost-optimal path, and may be eliminated. Dominance checks can be performed every time a new label is created, but this is not required. More details are given in Section 5.1.

2.3 State-space relaxations

A common method to speed up the labeling algorithm is by using a *state-space relaxation*. State-space relaxations substitute the elementarity requirement of the routes by a weaker condition. As there may be 2^n different sets of visited vertices $E(L)$, relaxing elementarity may significantly reduce the number of non-dominated labels.

When a state-space relaxation is used, the set of feasible routes $\mathcal{R}$ is expanded. To deal with non-elementary routes, we redefine a_i^r to be the number of times that vertex $i \in V'$ is visited, and b_{ij}^r to be the number of times that arc $(i, j) \in A$ is used. Note that the solution to the integer master problem (1) does not change, as selecting non-elementary routes would violate Constraints (1b). The additional variables do affect the LP relaxation, which becomes weaker. State-space relaxations thus allow for balancing the strength of the bound and the speed of the labeling algorithm.

Different state-space relaxations have been proposed in the literature. Houck et al. (1980) introduce *2-cycle elimination*, which relaxes elementarity by allowing cycles of length three or more. Irnich and Villeneuve (2006) discuss the more general *k-cycle elimination*, which allows cycles of length $k+1$ or more. Desaulniers et al. (2008) propose *partial elementarity*, which requires elementarity for only a subset of the vertices. Baldacci et al. (2011) introduce the *ng-route relaxation*, which only allows cycles that leave a certain neighborhood. A generalization is presented by Bulhões et al. (2018), which is discussed in Section 3.4.4.

For the *ng*-route relaxation, every vertex $i \in V'$ is assigned a *neighborhood* $N_i \subseteq V'$ such that $i \in N_i$. The set of visited vertices $E(L)$ is replaced by the set $\Pi(L)$, which is

updated according to the rule

$$\Pi(L \oplus j) = (\Pi(L) \cap N_j) \cup \{j\}. \tag{3}$$

The feasibility and dominance rules (see Definition 2) $j \notin E(L)$ and $E(L) \subseteq E(L')$ are replaced by $j \notin \Pi(L)$ and $\Pi(L) \subseteq \Pi(L')$, respectively.

It will be instructive to interpret the sets $E(L)$ and $\Pi(L)$ as different ways to retain information about the visited vertices. The set $E(L)$ contains all visited vertices, and with a slight abuse of language, we may say that the label L has a *perfect memory* if it remembers all previous visits through $E(L)$.

With the set $\Pi(L)$ we associate the *ng-memory*, which is typically imperfect. This follows from the update rule (3): When label L is extended to $L \oplus j$, the intersection $\Pi(L) \cap N_j$ tells us that L forgets all previous visits, except those in the neighborhood of j. Next, the visit to j is added to the memory.

When using the ng-route relaxation, a label can be extended to a previously visited vertex j, and thus create a cycle, if the previous visit to j is forgotten first. This requires visiting a vertex k for which $j \notin N_k$, i.e., visiting a neighborhood that does not contain j. The advantage of using ng-memory $\Pi(L)$ instead of perfect memory $E(L)$ is that the labels can be compared on their memory of the visits within the neighborhood only, which allows more labels to be dominated. In the case that $N_i = V'$ for all $i \in V'$, then the label never forgets a visit, and $\Pi(L) = E(L)$.

3 Resource-robust valid inequalities

In practice, valid inequalities are added to the SPP (Problem (1)) to strengthen the LP relaxation. The main difficulty in using valid inequalities is incorporating their duals into the pricing problem, which is modeled as an SPPRC. In the literature, the distinction is made between *robust* and *non-robust* valid inequalities (Fukasawa et al., 2006). The duals of robust valid inequalities can be incorporated without changing the structure of the SPPRC. Including non-robust valid inequalities, on the other hand, complicates the

pricing problem.

In this section, we first make the notion of "complicating the pricing problem" more concrete. With this notion, we formally introduce a new category of valid inequalities that lie somewhere between robust and non-robust: *resource-robust valid inequalities*. Informally, resource-robust valid inequalities are robust when the SPPRC includes specific resources.

3.1 Impact of robustness on the pricing problem

In the literature, robust and non-robust valid inequalities are distinguished by their impact on the pricing problem. As mentioned in the introduction, robust cuts do not change the structure of the pricing problem, i.e. their duals can be projected onto the arcs (Fukasawa et al., 2006). This property is in contrast with non-robust cuts, where one needs additional resources or adjusted dominance rules in order to incorporate them into the pricing problem.

To formalize the above contrast, we use Definition 3. Our definition focuses on maintaining the non-decreasing type (see Definition 1) of the pricing problem, so that the non-decreasing dominance rule can still be employed without modification.

Definition 3 (Robust impact on the pricing problem)**.** *Given a pricing problem of non-decreasing type, a valid inequality has a robust impact if the duals can be included such that the pricing problem is still of non-decreasing type, without adding any additional resources.*

It is immediately clear that the non-decreasing dominance rule is still valid when adding a valid inequality with a robust impact. In that sense, the "structure" of the pricing problem is preserved. It is also clear that non-robust cuts do not fall into this definition; their inclusion requires modification of the dominance rule or additional resources. In the remainder of this paper, whenever we refer to (not) "changing the structure of the pricing problem", we mean it in the sense of Definition 3.

3.2 Definition of resource-robustness

Whenever a valid inequality gets added to the RMP, its duals need to be included into the SPPRC to preserve the validity the labeling algorithm. More specifically, this amounts to altering the cost update $\bar{c}(L)$ and potentially adding additional resources. For a single valid inequality with dual σ, the cost update when extending label L over arc (i, j) becomes

$$\bar{c}(L \oplus j) = \bar{c}(L) + \bar{c}_{ij} - \sigma \Upsilon_{ij}(C(L)), \tag{4}$$

where $C(L)$ are cut-specific resources and Υ_{ij} gives the factor with which σ is subtracted from the reduced cost of the label. We call $C(L)$ the *cut memory* of the valid inequality and Υ_{ij} the *dual application function*. Note that the definition of the cut memory, its REFs and the dual application function uniquely define a cut in the RMP.

Our terminology concerning cut memory is inspired by the presentation of the limited-memory Subset-Row Cuts (lmSRC) by Pecin et al. (2017b). To incorporate the duals of the lmSRCs in the SPPRC, the authors introduce an additional resource for every cut, which can be seen as its memory. By limiting the information that is stored in the memory, the impact of these non-robust valid inequalities on the pricing problem is reduced.

Robust valid inequalities do not need additional resources in the SPPRC. As such, we say that the cut memory is *empty*, or that no cut memory is necessary. This implies that the dual application function is a constant for every arc (i, j). In other words, one can redefine the reduced arc costs $\bar{c}_{ij}$ as $\bar{c}_{ij} - \sigma \Upsilon_{ij}$. The duals are thus projected onto the arcs, and the structure of the pricing problem is not affected.

It follows immediately from linear programming duality that dual application functions define the coefficients of the associated valid inequalities. This is formalized as Proposition 4.

Proposition 4. *Consider a valid inequality for which the dual application function Υ_{ij} is known for every arc $(i, j) \in A$. Let route $r \in \mathcal{R}$ consist of the arcs $(i_1, j_1), \ldots, (i_{K_r}, j_{K_r}) \in A$, and let $C_1, \ldots, C_{K_r}$ be the associated cut memories. Then the coefficient d_r of route*

variable x_r in the valid inequality is given by

$$d_r = \sum_{k=1}^{K_r} \Upsilon_{i_k j_k}(C_k). \tag{5}$$

We are now ready to define *resource-robust valid inequalities*. The main idea is that resource-robust valid inequalities have a cut memory that can be derived from existing resources, so no additional resources are necessary. Definition 5 also includes conditions on non-decreasingness. It is proven in Theorem 7 that these conditions ensure that resource-robust valid inequalities do not complicate the pricing problem.

Definition 5 (Resource-robust valid inequality)**.** *A valid inequality with cut memory C and dual application function $\Upsilon_{ij}(C)$ is* resource-robust *for a vector of resources X if and only if*

1. *$-\sigma\Upsilon_{ij}(C)$ is non-decreasing in C for all $(i,j) \in A$ and for all feasible values of σ,*
2. *$C = F(X)$ for some non-decreasing function F.*

By Proposition 4, the first condition can equivalently be stated in primal form.

Corollary 6. *Condition 1 of Definition 5 holds if and only if any of the following are true for d_r as defined in Proposition 4 and for some constant $d \in \mathbb{R}$:*

- *$\Upsilon_{ij}(C)$ is non-decreasing in C for all $(i,j) \in A$ and the valid inequality is of the form $\sum_{r\in\mathcal{R}} d_r x_r \leq d$,*
- *$\Upsilon_{ij}(C)$ is non-increasing in C for all $(i,j) \in A$ and the valid inequality is of the form $\sum_{r\in\mathcal{R}} d_r x_r \geq d$,*
- *$\Upsilon_{ij}(C)$ is constant in C for all $(i,j) \in A$ and the valid inequality is of the form $\sum_{r\in\mathcal{R}} d_r x_r = d$.*

Proof. Follows immediately from Proposition 4, Definition 5, and the fact that the three cases have dual $\sigma \leq 0$, $\sigma \geq 0$, and $\sigma \in \mathbb{R}$, respectively. □

Note that Corollary 6 implies that a resource-robust *equality* is always robust. After all, $\Upsilon_{ij}(C)$ is constant for cut memory C. This implies $\Upsilon_{ij}(C) = \Upsilon_{ij}$, which allows the dual to be projected on the arcs.

The main benefit of the newly defined resource-robust valid inequalities is stated in the following theorem.

Theorem 7. *If the pricing problem is of non-decreasing type, and if resources X are included in the SPPRC, then valid inequalities that are resource-robust for X have a robust impact on the pricing problem, i.e., non-decreasingness is preserved and no additional resources are required.*

Proof. For ease of exposition, we give the proof for a single valid inequality. The argument can be repeated when multiple valid inequalities are involved.

Let C be the cut memory of label L. By definition, $C = F(X)$, so no additional resources are necessary to determine C. By combining the two conditions of Definition 5, we have that $-\sigma\Upsilon_{ij}(C) = -\sigma\Upsilon_{ij}(F(X))$ is non-decreasing in X, which is a non-decreasing resource by assumption. The original REF for the reduced cost $\bar{c}(L)$ is also non-decreasing by assumption, and adding the non-decreasing term $-\sigma\Upsilon_{ij}(C)$ preserves this property.

We conclude that no additional resources are necessary, and the REFs remain non-decreasing. Thus, by Defintion 3, the cuts have a robust impact on the pricing problem. This also implies that the non-decreasing dominance rules of Definition 2 remain valid after updating $\bar{c}(L)$. □

Theorem 7 shows that the dual application function of a resource-robust valid inequality can be written in terms of the existing resources X, while maintaining non-decreasingness. This is similar to how the dual application function of a robust valid inequality can be projected onto the existing arc costs. In both cases, existing label information is used, and no additional resources are necessary.

Due to our particular interest in resource-robust valid inequalities that relate to elementarity and to state-space relaxations, we define the following two terms. If the resources X equal the elementarity resources E, we call the valid inequality *elementarity-robust*. If X equals the *ng*-resources Π, then the valid inequality is *ng-robust*.

3.3 Example: SDCs and their ng-equivalent

The Strong Degree Cuts (SDCs), introduced by Contardo et al. (2014), are given by

$$\sum_{r \in \mathcal{R}} \min\{1, a_i^r\} x_r \geq 1 \qquad \forall i \in V', \tag{6}$$

where $\min\{1, a_i^r\}$ is a binary parameter that indicates whether route r has visited task i at least once. Contardo et al. (2014) introduced these cuts as an alternative to enforcing elementarity in the pricing problem. That is, not including the elementarity resources and using these cuts instead gives the same master problem lower bound and better algorithmic performance (Contardo et al., 2014).

We will now show, as an example, that the SDCs are elementarity-robust. One way to correctly apply the dual $\sigma_i \geq 0$ is to subtract σ the first time that task i is visited (and only the first time). To model this behavior, we introduce the cut memory C_i, which consists of a single resource that is one if i is visited at least once, and zero otherwise. The dual application function can then be stated as

$$\Upsilon_{jk}(C_i) = \begin{cases} 1 & \text{if } C_i = 0,\ k = i \\ 0 & \text{else.} \end{cases} \tag{7}$$

This function is non-increasing in C_i, as increasing C_i from zero to one cannot increase the value of the dual application function. Hence, by Corollary 6, Condition 1 of Definition 5 is satisfied. We mention that Contardo et al. (2014) define the same cut memory and dual application function, just not explicitly.

To show that Condition 2 of Definition 5 is satisfied for the elementarity resources, we construct a non-decreasing function F_i such that $C_i = F_i(E)$. By definition, C_i is one if and only if i has been visited, which results in $C_i = |E \cap \{i\}| \equiv F_i(E)$. It is immediately clear that $E \subseteq E'$ implies $F_i(E) \leq F_i(E')$, and thus F_i is non-decreasing.

As both conditions in Definition 5 are satisfied, we conclude that the SDCs are elementarity-robust. It follows by Theorem 7 that SDCs can be added without changing the structure of the pricing problem, given that the elementarity resources are available

(i.e., no state-space relaxation is applied). The new dominance rules are the same as those in Definition 2, except that the new dual application functions need to be added to the reduced cost function $\bar{c}(L)$.

The example shows how resource-robustness can be used to obtain simple dominance rules. Beside this illustration, the fact that SDCs are elementarity-robust has little practical value. It can be seen that $\min\{1, a_i^r\} = a_i^r$ for all elementary routes r and vertices $i \in V'$. Therefore, when elementarity is enforced in the subproblem, the Constraints (6) are redundant and may simply be removed.

This derivation does lead to another interesting observation: by replacing the elementarity resources E in the cut memory $C_i \equiv |E \cap \{i\}|$ by ng-resources Π, we obtain new cuts. These cuts, which we call the ngSDCs, trade the elementarity-robustness of the SDCs for ng-robustness, at the expense of being weaker than the SDCs.

The ngSDCs state that tasks that are in ng-memory cannot be revisited. Therefore, when the ng-route relaxation is used, the ngSDCs are redundant. However, as mentioned at the start of this section, the SDCs are used to enforce elementarity in the subproblem with the same master problem lower bounds. Following the exact same argument as by Contardo et al. (2014), it can be shown that the ngSDCs can serve as an alternative to enforcing *ng-routes* in the subproblem. Similarly, this leads to the same lower bound as if the ng-route relaxation were used. Given the computational succes of the SDCs, studying the performance of using ngSDCs is an interesting topic for future research.

Finally, we point out the importance of the sign of the inequality in Equation (6). If the sign would be the other way around, then the dual $\sigma_i \leq 0$ would be non-positive. As a result, the dual application function should be non-decreasing (instead of non-increasing) by Corollary 6, and the valid inequality would not be elementarity-robust.

3.4 Practical aspects of ng-robust valid inequalities

In this section, we consider some practical aspects of using ng-robust or elementarity-robust valid inequalities that are important when implementing a BPC algorithm. As elementarity-robust is a special case of ng-robust (see Section 2.3), this section focuses

on the latter. Recall that ng-robust valid inequalities depend on the ng-memory, denoted by $\Pi(L)$.

3.4.1 Unreachable vertices

A common acceleration technique for BPC algorithms, introduced by Feillet et al. (2004), is to consider *unreachable vertices* instead of visited vertices. We now show that ng-robust valid inequalities may be incompatible with this acceleration technique, and we propose a way of mitigating this disadvantage.

Applied to the ng-route state-space relaxation, using unreachable vertices amounts to the following. Let $\Pi(L)$ be the ng-memory of label L. Next, determine a set of unreachable vertices $U(L) \subseteq V' \backslash \Pi(L)$ that cannot be reached by any extension of L due to the resource constraints. In vehicle routing, for example, the set $U(L)$ may correspond to the set of orders that cannot be added to the current vehicle without violating the capacity constraint. The resources $\Pi(L)$ are then replaced by $\Pi^*(L) \equiv \Pi(L) \cup U(L)$.

The fact that $\Pi(L)$ is no longer a resource in the SPPRC affects the ng-robust valid inequalities. By Definition 5, the cut memory of an ng-robust valid inequality is defined as $C = F(\Pi(L))$ for some non-decreasing function F. As $\Pi(L)$ is no longer available, one may try to define $C = F^*(\Pi^*(L))$ for some non-decreasing function F^*, but there is no guarantee that this is possible. The main complicating factor is that $\Pi^*(L) \subseteq \Pi^*(L')$ does not imply $\Pi(L) \subseteq \Pi(L')$, which ruins non-decreasingness.

To mitigate the disadvantage of not being able to use unreachable vertices, we propose a heuristic: use $\Pi(L)$ to determine the cut memory and to evaluate the dual application functions, and use $\Pi^*(L)$ in the dominance rules. This allows too many labels to be dominated, but if a route with negative reduced cost is found, this route is valid and can be added to the RMP. Only when the heuristic fails to find such a route, we need to weaken the dominance rules by using $\Pi(L)$ instead of $\Pi^*(L)$. Such *heuristic pricing* is discussed in more detail in Section 5.1.1.

3.4.2 Dynamic ng-route relaxation

Roberti and Mingozzi (2014) introduce the *dynamic ng-route relaxation*. In this case, the initial neighborhoods $N_i \subseteq V'$ for all $i \in V'$ are dynamically expanded to make non-elementary routes infeasible after they are generated. By excluding more non-elementary routes, the lower bound is improved. For more details, we refer to Roberti and Mingozzi (2014).

Expanding the neighborhoods in the dynamic ng-route relaxation results in some computational overhead when ng-robust valid inequalities are used. First, the reduced costs of columns in the column pool (columns that are currently not part of the RMP, but may be added later) will have to be recalculated. This follows from the fact that expanding the neighborhoods changes the ng-memory, which changes the cut memory, and therefore the evaluations of the dual application functions. Second, if the coefficients of the ng-robust valid inequalities depend on the cut memory, which is case in Sections 4.1, these coefficients have to be updated as well.

On the other hand, ng-robust valid inequalities have the benefit that they increase in strength when the neighborhoods are expanded, thereby improving the lower bound of the relaxation. This result is stated formally as Proposition 8. Note that "increase in strength" is formally an abuse of language, as technically the SPP formulation becomes stronger instead of the cuts. However, as we only consider one formulation (the SPP), this should be clear from context.

Proposition 8. *Valid inequalities that are ng-robust weakly increase in strength when the ng-route relaxation neighborhoods are expanded from N_i to $N_i^+ \supseteq N_i$ for all $i \in V'$.*

Proof. For a given label L, let $\Pi(L)$ and $\Pi(L)^+$ respectively denote the ng-memory before and after expanding the neighborhoods. It follows immediately from the definition of ng-memory that $\Pi(L) \subseteq \Pi^+(L)$.

Corollary 6 gives three possible descriptions for the valid inequality. In the first case, $\Upsilon_{ij}(C)$ is non-decreasing in C, and the valid inequality is of the form $\sum_{r \in \mathcal{R}} d_r x_r \leq d$, with $d_r = \sum_{k=1}^{K_r} \Upsilon_{i_k j_k}(C_k)$ as in Proposition 4. By non-decreasingness, $\Upsilon_{ij}(\Pi(L)) \leq$

$\Upsilon_{ij}(\Pi^+(L))$. Hence, the left-hand side coefficients weakly increase when the neighborhoods are expanded, which makes the valid inequality weakly stronger.

The argument for the second case is analogous. The third case is trivial, as the dual application function is constant, and the coefficients do not change when the neighborhoods are expanded. This completes the proof. □

3.4.3 Bidirectional labeling

Righini and Salani (2006) propose *bidirectional labeling* to speed up labeling algorithms such as the one presented in Section 2.2.2. Bidirectional labeling algorithms generate both *forward paths* starting in vertex 0 and *backward paths* starting in vertex $n+1$. By combining forward and backward paths, feasible routes are generated.

Bidirectional labeling requires that the REFs can be inverted (redefined for the backward path) in such a way that the inverted functions are also non-decreasing. Irnich (2008) defines inverse REFs that are applicable to a wide range of problems. Baldacci et al. (2011) define inverse ng-paths and the associated inverse ng-memory $\Pi^{-1}(L)$.

In the presence of ng-robust valid inequalities, inverting the REF of the reduced cost function $\bar{c}(L)$ is impossible in general. The reduced cost function depends on the dual application functions, which depend on the ng-memory. For the backward path, only the inverse ng-memory is available, which does not contain any information about the forward path. It follows that the duals cannot be applied correctly.

To achieve compatibility with bidirectional labeling, we propose to relax the ng-robust valid inequalities as follows. For the forward path, apply the duals in the same way as in the monodirectional labeling algorithm. For the backward path, evaluate the dual application functions as if $N_i = \{i\}$ for all $i \in V'$, i.e., no visits are remembered, and therefore $\Pi^{-1}(L) = \{i(L)\}$. The relaxed valid inequality is robust on the backward path, which means that the reduced cost function can be inverted. It follows from Proposition 8 that the relaxed valid inequality is indeed not stronger than the original, and is therefore valid.

Another relaxation can be obtained when the ng-robust valid inequality is symmetric

in the sense that the cut remains valid if $\Pi(L)$ is replaced by $\Pi^{-1}(L)$ everywhere. The ngSDCs and the valid inequalities presented in Section 4.1 have this property, for example. In this case, the opposite relaxation is possible: the valid inequality is made robust on the forward path by assuming that $\Pi(L) = \{i(L)\}$ when duals are applied, while the reduced cost on the backward path is calculated using $\Pi^{-1}(L)$.

In a bidirectional labeling algorithm, either or both relaxed valid inequalities can be used instead of the original ng-robust valid inequality. Additionally, we note that any positive linear combination of the two relaxations results in an ng-robust valid inequality that is compatible with bidirectional labeling.

3.4.4 Arc-based ng-memory

Bulhões et al. (2018) present a generalization of the ng-route relaxation in which the ng-memory is updated according to neighborhoods associated with the arcs. That is, the vertex neighborhoods N_j for $j \in V'$ are replaced by arc neighborhoods N_{ij} for $(i, j) \in A$. The update rule (3) is replaced by

$$\Pi(L \oplus j) = (\Pi(L) \cap N_{ij}) \cup \{j\}. \tag{8}$$

Note that the ng-memory $\Pi(L)$ is still a set of vertices and remains non-decreasing in L. It is straightforward to verify that changing the update rule does not affect any of our results. That is, ng-robust valid inequalities are compatible with the generalized ng-route relaxation.

4 Application to the CVRP

In this section, we apply the notion of resource-robustness to the Capacitated Vehicle Routing Problem (CVRP). In the CVRP, every route corresponds to a delivery route, and every task corresponds to a delivery to a client. A positive *demand* $q_i > 0$ is associated with every task $i \in V'$, and the total demand of a route cannot exceed the *vehicle capacity* $Q > 0$. To ensure feasibility, we assume that $q_i \leq Q$ for all $i \in V'$. For more information

on the CVRP and other vehicle routing problems, we refer to Toth and Vigo (2014).

We introduce the *ng-Capacity Cuts* (*ng*CCs) as a stronger alternative to the well-known rounded capacity cuts for the CVRP. The *ng*CCs are analyzed theoretically, and we present a separation algorithm to find violated inequalities. We also discuss other *ng*-robust valid inequalities that can be derived in the same way. In Section 5, the *ng*CCs are tested empirically with computational experiments.

In terms of constraints, the CVRP is one of the simplest problems that fit in our framework. It can be stated as a set partitioning problem in which all routes are elementary, with the single additional constraint that routes respect the vehicle capacity. Many vehicle routing problems include the CVRP as a special case, and our results for the CVRP are immediately applicable to these problems.

4.1 *ng*-capacity cuts

Capacity cuts for the CVRP are based on the following observation. Let $S \subseteq V'$ be a subset of the tasks, with total demand $\sum_{i \in S} q_i$. As each route respects the vehicle capacity Q, at least $\left\lceil \frac{1}{Q} \sum_{i \in S} q_i \right\rceil$ routes are needed to execute the tasks in S. Rounding up the fraction is allowed, as the number of routes is integral, which follows from the integrality of the route variables.

4.1.1 Capacity cuts and strengthened capacity cuts

Two known classes of capacity cuts are the rounded Capacity Cuts (CCs, Augerat et al. (1998)) and the Strengthened Capacity Cuts (SCCs, Baldacci et al. (2004)). Before stating the *ng*CC, we discuss these valid inequalities in reverse order.

The SCCs are given by

$$\sum_{r \in \mathcal{R}} \zeta_r^{SCC}(S) x_r \geq \left\lceil \frac{1}{Q} \sum_{i \in S} q_i \right\rceil, \tag{9}$$

with $\zeta_r^{SCC}(S)$ a binary parameter equal to one if route r contains any task in S, and zero otherwise. Note that the left-hand side is the number of routes that visit S, and the

right-hand side is the lower bound discussed before.

The SCCs are known not to be robust, but can be shown to be elementarity-robust in the same way as in the example in Section 3.3. Let the cut memory C_S consist of a single resource that is one if the set S is entered at least once, and zero otherwise. The dual application function is given by

$$\Upsilon_{ij}^{SCC}(C_S) = \begin{cases} 1 & \text{if } i \notin S,\ j \in S,\ C_S = 0 \\ 0 & \text{else.} \end{cases} \tag{10}$$

It is straightforward to prove elementarity-robustness by verifying the conditions in Definition 5.

The CCs are a robust but weaker alternative to the SCCs, given by

$$\sum_{r \in \mathcal{R}} \zeta_r^{CC}(S) x_r \geq \left\lceil \frac{1}{Q} \sum_{i \in S} q_i \right\rceil, \tag{11}$$

with $\zeta_r^{CC}(S)$ the number of times that route r enters the set S. The associated dual application function is given by

$$\Upsilon_{ij}^{CC} = \begin{cases} 1 & \text{if } i \notin S,\ j \in S \\ 0 & \text{else.} \end{cases} \tag{12}$$

Note that the CCs are robust, and therefore the cut memory $C_S = \varnothing$ is empty. As mentioned in Section 3.2, the dual can be projected onto the arcs.

It is important to note that the CCs over-estimate the left-hand side of (9) by *double-counting* the entries into the set S. For the SCCs, only the first entry into the set S increases the value of ζ_r^{SCC}. For the CCs, every time an arc $(i, j) \in A$ is used with $i \notin S$ and $j \in S$, the value of ζ_r^{CC} is increased. As a result, routes that enter and leave S more than once are counted double. This double-counting results in a valid inequality that is weaker, but does not require any cut memory, because it is not necessary to remember whether S has been visited before.

Naturally, as the CCs and SCCs are different kinds of cuts, separation also differs

between them. We discuss this in more detail in Section 4.3.

4.1.2 *ng*-capacity cuts

We are now ready to introduce the *ng*CCs. The *ng*CCs have a similar structure to the CCs and the SCCs, and fall in between the two in strength and in robustness. Where the CCs are robust and the SCCs are elementarity-robust, the *ng*CCs are *ng*-robust.

The *ng*CCs are defined as

$$\sum_{r \in \mathcal{R}} \zeta_r^{ng}(S) x_r \geq \left\lceil \frac{1}{Q} \sum_{i \in S} q_i \right\rceil, \tag{13}$$

where $\zeta_r^{ng}(S)$ equals the number of times route r travels into set S for the first time, *according to the ng-memory*. That is, whenever route r travels into S, it is counted if and only if all nodes in set S are not contained in the current *ng*-memory $\Pi(L)$.

Figure 1 shows an example for a route $r = (0, 1, 2, 3, 4, 5, n+1)$ and a subset $S = \{1, 3, 5\}$. Assume that the neighborhoods are given by $N_1 = N_3 = N_4 = N_5 = \{1, 3, 4, 5\}$ and $N_2 = \{2, 3\}$. In this example, route r travels into S three times in total, which implies $\zeta_r^{CC}(S) = 3$. As r travels into S at least once, we have $\zeta_r^{SCC}(S) = 1$. According to *ng*-memory, the route enters S *two times*: first when traveling from 0 to 1 (as the *ng*-memory for this extension is empty), and second when traveling from 2 to 3 (the visit to 1 is not remembered because $1 \notin N_2$). Traveling from 4 to 5, the *ng*-memory equals $\{3, 4\}$, so this entry is not counted. It follows that $\zeta_r^{ng}(S) = 2$.

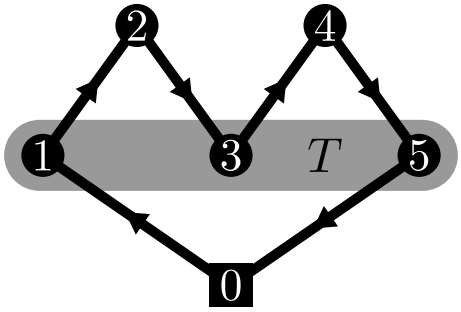


Figure 1: Example instance for the *ng*CCs.

For the *ng*CCs, whether the entries into the set S are double-counted depends on the current state of the *ng*-memory. It follows that $\zeta_r^{SCC}(S) \leq \zeta_r^{ng}(S) \leq \zeta_r^{CC}(S)$, which

implies that the ng-capacity cuts are valid for the CVRP. Note that if the ng-memory is perfect ($N_i = V'$ for all $i \in V'$), then double-counting is eliminated and the ngCCs are equivalent to the SCCs. On the other hand, if $N_i = \{i\}$ for all $i \in V'$, then every entry is double-counted, which makes the ngCCs equivalent to the CCs.

To incorporate the ngCCs into the pricing problem, we use the cut memory C_S^{ng}, which equals one if S has been visited according to ng-memory and zero otherwise. That is, $C_S^{ng} \equiv \min\{1, |S \cap \Pi(L)|\}$. The dual application function is given by

$$\Upsilon_{ij}^{ng}(C_S^{ng}) = \begin{cases} 1 & \text{if } i \notin S,\ j \in S,\ C_S^{ng} = 0 \\ 0 & \text{else.} \end{cases} \tag{14}$$

We verify ng-robustness by checking the two conditions in Definition 5. First, we note that $\Upsilon_{ij}^{ng}(C_S^{ng})$ is non-increasing in the cut memory C_S^{ng}. Second, the cut memory $C_S^{ng} = \min\{1, |S \cap \Pi(L)|\}$ is non-decreasing in the ng-memory $\Pi(L)$. It follows that the ngCCs are ng-robust.

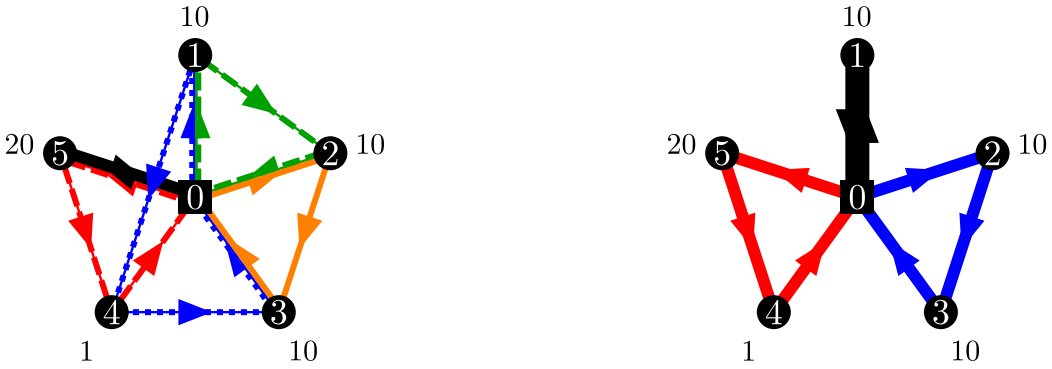
By Theorem 7 we have that the duals of the ngCCs can be incorporated without changing the structure of the pricing problem. After adding the dual application functions to $\bar{c}(L)$, the non-decreasing dominance rules can be used.

4.2 Effect on the lower bound

Next, we present an example to demonstrate the size of the effect that ngCCs can have on the lower bound. In this example, using only CCs results in an optimality gap of 50%. By including ngCCs, the optimality gap is reduced to 0% and the optimal solution is found at the root node.

Our five-task CVRP instance on a complete graph is presented in Figure 2a, with vertices 0 and $n+1$ at the same location. The cost for traversing arcs $(0,2)$, $(0,3)$, and $(0,4)$ is equal to one, and all other arcs have cost zero. The demand for each task is displayed next to the task vertex, and vehicle capacity Q is set to 29. The neighborhoods are defined as $N_1 = N_2 = N_3 = N_4 = \{1,2,3,4\}$ and $N_5 = \{5\}$. Finally, the colored arrows show an optimal solution to the master problem when all CCs are included. Each

colored tour corresponds to a route variable with value $x_r = 0.5$. It can be seen that the corresponding objective value is equal to 0.5.



(a) RMP solution that satisfies all CCs.

(b) RMP solution that satisfies all ngCCs.

Figure 2: Example instance where ngCCs increase the lower bound.

In this fractional solution, the ngCC corresponding to $S = \{1, 2, 3\}$ is violated: The number of vehicles needed is $\lceil \frac{30}{29} \rceil = 2$, whereas only 1.5 vehicles currently serve these tasks. The CC corresponding to the same set counts both entries of the blue route $(0 \to 1 \to 4 \to 3 \to n + 1)$ into S. For the ngCC, the second entry is not counted, as $1 \in N_4$. Adding the violated ngCC to the master problem results in an integer optimal solution with objective value 1, shown in Figure 2b.

We hypothesize that ngCCs are most effective for instances with high demand clusters of tasks, as is the case for $\{1, 2, 3, 4\}$ in the example. If capacity is restrictive, this may lead to fractional routes that enter subsets of the cluster multiple times. As tasks in clusters are typically in each others neighborhoods, the ngCCs can prevent double counting in this case, which leads to stronger cuts and better bounds.

4.3 Separation algorithm

We present a *separation algorithm* to find violated ngCCs for the current solution to the master problem. That is, for current (fractional) values of the x_r variables, the separation algorithm returns a set of tasks $S \subseteq V'$ for which Equation (13) does not hold.

Instead of designing a specialized separation scheme, we propose to construct a *separation network* such that violated ngCCs in the original network correspond to violated

CCs in the separation network. Existing separation algorithms for CCs, such as those presented by Lysgaard et al. (2004), can then be leveraged to separate *ng*CCs.

Definition 9 (κ-scaled separation network)**.** *For a given $\kappa \in \mathbb{N}$, the* κ-scaled separation network *is constructed from the original network as follows.*

1. *Scale down all non-zero route variables $x_r > 0$ to obtain*

$$\bar{x}_r = \frac{x_r}{\min\left\{\kappa, \left\lceil \frac{\#(r)}{2} \right\rceil\right\}},$$

   with $\#(r)$ the number of tasks that route r visits (double counting multiple visits).

2. *Project the $\bar{x}_r$ variables onto the arcs to obtain arc flows.*

3. *Introduce zero-demand dummy tasks following Algorithm 1 in Appendix A to restore the in- and outflow of each task to one.*

Proposition 10. *For a given $\kappa \in \mathbb{N}$, if a violated ngCC in the original network corresponds to a subset $S \subseteq V'$ that is entered at most κ times by every non-zero route r, then the CC corresponding to S in the separation network is violated as well.*

Proof. See Appendix B. □

Corollary 11. *For $\kappa = \max_{r \in \mathcal{R} | x_r > 0} \left\lceil \frac{\#(r)}{2} \right\rceil$, all violated ngCCs in the original network correspond to violated CCs in the separation network.*

Proof. Follows immediately from Proposition 10 and the fact that every route can enter subset S at most $\left\lceil \frac{\#(r)}{2} \right\rceil$ times. □

By Corollary 11, all violated *ng*CCs can be found by separating CCs on a separation network with sufficiently high κ. In the reverse direction, violated CCs on the separation network do not necessarily correspond to violated *ng*CCs. For performance reasons, we therefore propose to start with a separation network for $\kappa = 1$, and only increment the parameter when no more violated *ng*CCs can be found.

4.4 Application to other valid inequalities

The derivation in Section 4.1 and the separation algorithm in Section 4.3 are not specific to the *ng*CCs, and can be used to derive *ng*-robust variants of other cuts in the literature as well.

For example, consider the *k-path cuts*, which are given by

$$\sum_{r \in R} \sum_{(i,j) \in A} b_{ij}^r x_r \geq k(S), \tag{15}$$

with $S \subseteq V'$ a subset of tasks and with $k(S) \geq 1$ a lower bound on the number of routes that enter S in any feasible solution. Recall that b_{ij}^r is the number of times that arc (i, j) is used in route r. Note that CCs are specific k-path cuts with $k(S) = \left\lceil \frac{1}{Q} \sum_{i \in S} q_i \right\rceil$.

In the same way that we derive *ng*CCs from the CCs in Section 4.1, one can immediately derive *ng*-robust k-path cuts from the k-path cuts. As special cases, this includes *ng*-robust *subtour elimination constraints* (Dantzig et al., 1954) and *ng*-robust *2-path cuts* (Kohl et al., 1999). It is straightforward to see that the same is true for the generalized k-path cuts presented by Desaulniers et al. (2008). For all these *ng*-robust valid inequalities, κ-scaled separation networks as introduced in Section 4.3 allow existing separation algorithms to be used.

The *ng*SDCs presented in Section 3.3 provide another example. These cuts can serve as an alternative to enforcing *ng*-routes in the subproblem, similarly to how the SDCs can enforce elementarity in the subproblem.

5 Computational experiments

In this section, we empirically compare the effects of adding CCs or *ng*CCs in a BPC algorithm for the CVRP. The goal of these experiments is to study the performance of using *ng*-robust cuts –and, more generally, resource-robust cuts– in a relatively simple setting. A straightforward BPC algorithm is implemented in C++, following the descriptions in Sections 2.2 and 2.3, details of which are discussed in Section 5.1.

All experiments are run on an Intel Xeon W-2123 3.6GHz processor and 16GB of RAM. The algorithm is run on a single thread, and a time limit of three hours per instance is imposed. CPLEX version 12.7.1 is used to solve the RMP.

5.1 Implementation BPC algorithm

The basic steps of the BPC algorithm are implemented as described in Sections 2.2 and 2.3. To model the CVRP, we use current load $q(L)$ as an additional resource. Its REF is defined as $q(L \oplus j) = q(L) + q_j$ with upper resource window Q. That is, if $q(L \oplus j) > Q$, the extension of L to j is infeasible. For notational convenience, we denote $q_0 = q_{n+1} = 0$. Following Pecin (2014), dominance checks are only performed between labels with the same current load.

The ng-route relaxation is implemented as described in Section 2.3. Neighborhoods of size 10 are constructed by selecting the nearest clients. For valid inequalities, either CCs or ngCCs are added, with dual application functions defined in (12) and (14), respectively. The duals of the CCs are projected onto the arcs, as discussed in Section 3.2. The ngCCs are included by adding the dual application functions to the reduced cost of the label, as discussed in Section 4.1.2.

5.1.1 Heuristic pricing

To quickly discover negative reduced cost columns, we use heuristic pricing as described by Desaulniers et al. (2008), among others. See Costa et al. (2019) for a survey on this topic. The original pricing problem is simplified through *aggressive dominance* and *arc elimination*. The former ignores some of the resources in the dominance check, and the latter ignores unpromising arcs based on their reduced costs.

If a route with negative reduced costs is identified, it can be added to the RMP. When no more routes can be found, the labeling algorithm is applied again with more resources and more arcs are taken into account. Eventually, the full problem —with all arcs and resources— is solved, which guarantees that all negative reduced cost columns are found.

As discussed in Section 3.4.1, the unreachable vertices acceleration technique is in-

compatible with ng-robust valid inequalities when the pricing problem is solved exactly, but may be included heuristically. Therefore, we employ this technique when either no ngCCs are present in the model, or heuristic pricing is used as discussed above. The set of unreachable vertices is taken to be the set of tasks that cannot be added to the current route without violating the capacity constraint.

5.1.2 Separation of valid inequalities

As discussed in Section 4.3, we separate ngCCs by separating CCs on κ-scaled separation networks. The CCs are separated with the CVRPSEP package by Lysgaard (2003), which is described by Lysgaard et al. (2004). We modify the code to prevent CVRPSEP from terminating early if one of the four heuristics is successful. This is done because not all violated CCs in the separation network correspond to violated ngCCs in the original network. In preliminary experiments, no cuts have been found for $\kappa > 4$, so $\kappa = 4$ is set as the maximum value.

For each potential cut defined by subset $S \subseteq V'$, it is determined whether the cut is added as an ngCC or as a weaker CC. Even though ngCCs do not change the structure of the pricing problem, we find that it is not worth the effort to evaluate the more complicated dual application function if the strength of the ngCC and the CC is similar.

Let $v^{ng}(S)$ and $v^{CC}(S)$ be the violation of the ngCC and the CC corresponding to subset $S \subseteq V'$, respectively. If $v^{ng}(S) \geq \max\{0.2,\ v^{CC}(S) + 0.1\}$, then the cut is added as an ngCC. Else if $v^{CC}(S) \geq 0.1$, then the cut is added as a CC. Otherwise, the violated valid inequality is ignored.

Next to the above heuristic separation procedure, we also experiment with exact separation of CCs and ngCC. To this end, the separation problems are formulated as MIPs and solved by CPLEX. These MIPs (see Appendix C for details) are able to find the single *most violated* CC or ngCC in the current solution. Therefore, it is impractical to separate all cuts using this exact separation. Instead, we only execute the exact separation whenever the heuristic one fails, in order to find any additional cuts the heuristic separation might have missed.

5.1.3 Branching rule

As is common in the literature (e.g., see Desrochers et al. (1992)), we branch on the arc flows $z_{ij} \in \{0, 1\}$, which are defined as $z_{ij} = \sum_{r \in \mathcal{R}} b_{ij}^r x_r$ for all $(i, j) \in A$. Recall that b_{ij}^r is the number of times that arc (i, j) is used in route r.

We use approximate strong branching, similarly to Ropke (2012) and Pecin et al. (2017b). For a given arc $(i, j) \in A$, the score of a potential branch is given by $\Delta = 0.75 \min\{\Delta^+, \Delta^-\} + 0.25 \max\{\Delta^+, \Delta^-\}$, where Δ^+ and Δ^- are the objective values after forcing the arc flow z_{ij} to one or zero, respectively. The arc to branch on is chosen as follows:

1. Select the 30 arcs for which $|z_{ij} - \frac{1}{2}|$ is the smallest.

2. For these arcs, calculate the Δ-score without generating any additional routes. Select the five arcs with the highest score.

3. For these arcs, calculate the Δ-score while generating additional routes with heuristic pricing only. Select the arc with the highest score to branch on.

5.2 Test instances

We test the BPC algorithm on the A, B and P benchmark instance classes by Augerat (1995). The instance classes mainly differ in where tasks are located in the Euclidean plane: for the A instances the placement is uniform, for the B instances it is clustered and for the P instances the placements are derived from real-life instances.

As hypothesized in Section 4.2, we suspect the *ng*CCs to work best in clustered instances where capacity is restrictive. Therefore, we find it interesting to also test the performance of the algorithm on the Augerat B instances (for which the task locations are clustered) with the demand doubled (or set to the vehicle capacity, if exceeding). We call these the B2 instances. The restriction on the number of vehicles is ignored for all instances.

5.3 Separation strategies

To compare the performance of using CCs to using ngCCs, the BPC algorithm is tested for six different separation strategies. Three strategies only use CCs, while the others allow for ngCCs. An overview is presented in Table 1.

	Cut type	Max κ	Separation type
`cc1`	CC	1	Heuristic
`cc4`	CC	4	Heuristic
`ccX`	CC	4	Exact
`ng1`	ngCC	1	Heuristic
`ng4`	ngCC	4	Heuristic
`ngX`	ngCC	4	Exact

Table 1: Overview of separation strategies.

The *cut type* in Table 1 indicates whether CCs or ngCCs are used. The next column states the maximum value of κ that is used for the κ-scaled separation networks, as discussed in Section 4.3. If $\max \kappa = 1$, then only CCs are separated. Note that for the ngCC separation methods, these CCs may be added as ngCCs. If $\max \kappa = 4$, then the value of κ may be increased up to four to allow more ngCCs to be found. The final column indicates whether valid inequalities are separated with the heuristic or with the exact implementation, as discussed in Section 5.1.2. Note that the exact separation is executed only when the heuristic separation (with Max $\kappa = 4$) fails to find sufficiently violated cuts.

5.4 Computational results

In this section, we analyze the performance of the BPC algorithm on the instances discussed in Section 5.2 for the six strategies introduced in Section 5.3. We first compare CCs and ngCCs on how well they contribute to closing the integrality gap for the considered instances. Next, we analyze the time spent in different parts of the algorithm. For

detailed tables with numerical results, we refer to Appendix D.

5.4.1 Relevant gap closing

To evaluate how the different strategies contribute to closing the optimality gap, we introduce the *relevant gap* metric. For a given instance, the relevant gap of a given separation strategy s at time t is given by

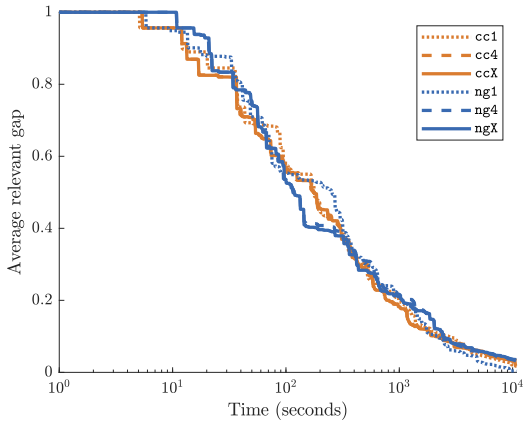
$$relgap_{s,t} = \min\left\{\frac{z^* - LB_{s,t}}{z^* - LB_{root}},\ 1\right\},$$

where $LB_{s,t}$ is the lower bound of strategy s at time t, LB_{root} is the lowest lower bound obtained by any of the strategies at the root node (immediately before branching), and z^* is the optimal objective value of the instance (or the best-known lower bound rounded up to the nearest integer, if no strategy reached optimality).

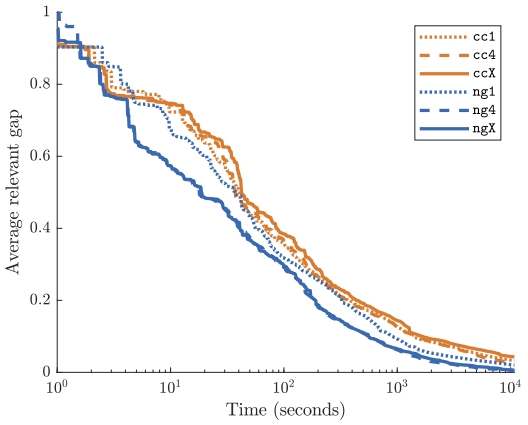
When a branch-and-bound process starts, the lower bound $LB_{s,t}$ is set to 0 and stays that way until the root node is solved. During that period, $LB_{s,t}$ is smaller than LB_{root} which means the relevant gap is fixed at one. Only when the root node with cuts is solved does $LB_{s,t} \geq LB_{root}$ hold, which means the relevant gap can become lower than one. Given enough time, the gap will decrease to zero, at which point the relevant gap is closed and the problem is solved to optimality. As the relevant gap ranges between zero and one, we can average over all instances and compare their performance with each other. By defining LB_{root} as-is, we focus only on closing the gap *after* solving the root node.

Figure 3 presents the average relative gaps over time for all tested instances classes, and for all six separation strategies, based on the data in Appendix D. Note that the graph stops at 10800 seconds, which corresponds to the time limit of 3 hours.

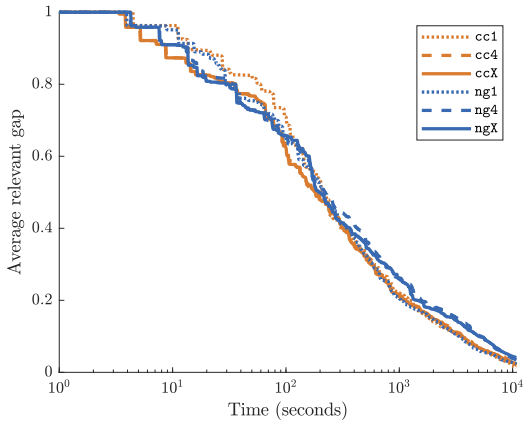
Our first observation is that for the A, B, and P instances, the average relevant gap over time progresses very similarly for the six strategies. This indicates that the additional strength of the *ng*CCs compensates for the overhead of evaluating more complicated dual application functions, and for the fact that the unreachable vertices acceleration technique cannot always be used (see Section 3.4.1). Although the *ng*CCs do not seem to improve



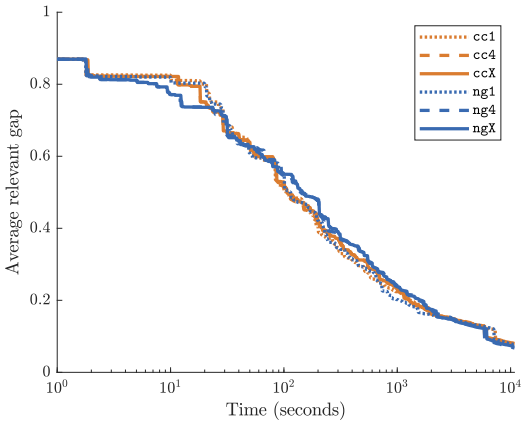
(a) Augerat B instances.



(b) Augerat B2 instances.



(c) Augerat A instances.



(d) Augerat P instances.

Figure 3: Average relevant gap over time (logarithmic scale).

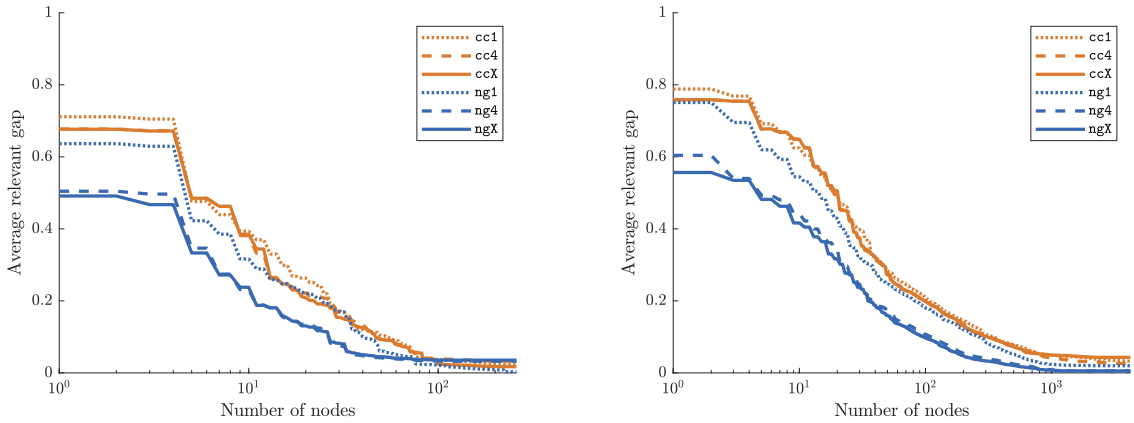
performance, they also do not hinder the algorithm for these instances.

For the B2 instances, on the other hand, the strategies that use *ng*CCs clearly demonstrate better performance. On average, `ng1` outperforms the strategies `cc1`, `cc4`, and `ccX` for most of the runtime. Recall that for `ng1` only CCs are separated, which are potentially added as *ng*CCs. The strategies `ng4` and `ngX` allow for more *ng*CCs to be found, which improves performance further. The average relevant gap is closed up to 19 percentage points further at the same time, or looking at it differently, the same average relevant gap is obtained up to 9309 seconds earlier.

The difference in average relative gap for the strategies in Figure 3b decreases over time. This is a consequence of the typical behavior where the last part of the gap is the most difficult to close. For the relevant gaps close to zero, the `ng` strategies still obtain

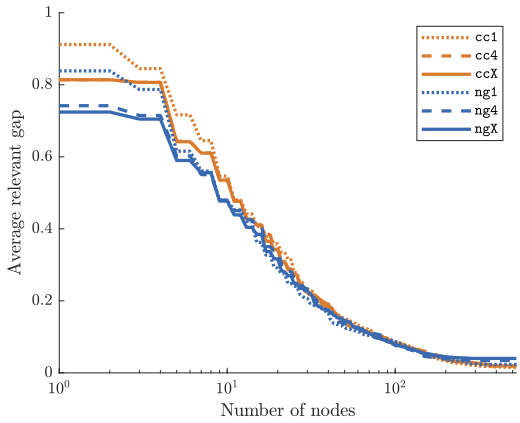
the same value significantly faster than the `cc` strategies. If route enumeration is used to close the remainder of the gap (e.g., see Pessoa et al. (2009) and Baldacci et al. (2011)), this process can be started significantly earlier.

It is also of interest to see how the relevant gap decreases per node of the branching tree, rather than per time unit. This more clearly demonstrates the effect of the valid inequalities, rather than how time-efficient the current implementation is. Figure 4 shows the average relevant gap per solved node for each of the four instance classes.

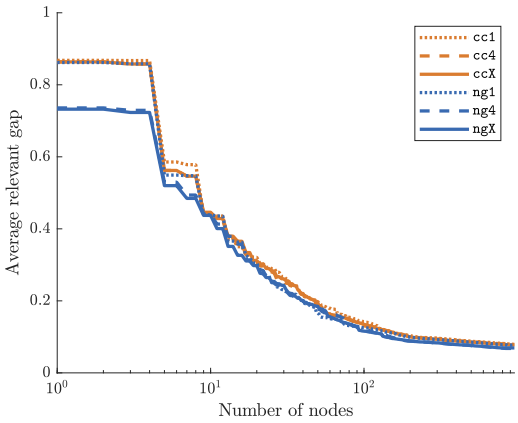


(a) Augerat B instances.

(b) Augerat B2 instances.



(c) Augerat A instances.



(d) Augerat P instances.

Figure 4: Average relevant gap per node

We observe from Figure 4 that using *ng*CCs results in smaller relevant gaps for a given number of nodes in the search tree. This effect is especially pronounced at the root node, where `ng4` and `ngX` close the relevant gap up to 23 percentage points further than the other strategies. Again, the clearest advantage is observed for the B2 instances.

It is noteworthy that `ng1` performs similarly to the `cc` separation strategies, showing that naive separation of ngCCs simply does not yield enough cuts. On the other hand, there is barely any difference between `ngX` and `ng4`, showing that the κ-scaled separation networks are able to capture the most relevant valid inequalities, and that exact separation is unnecessary.

Comparing Figures 3 and 4, we see that the performance of the `ng` strategies is better when measured per node of the search tree than per unit of time. The main reason is that solving a single node with the `ng` strategies is more expensive than with the `cc` strategies, even though the structure of the pricing problem is the same. This is an important observation that is explored further in Section 5.4.2.

In all four instance classes –especially the A and P instances–, the differences between the CCs and ngCCs become less prevalent as the number of nodes increases. In part, this is again due to the last part of the gap being the most difficult to close. Furthermore, the `ng` strategies typically solve less nodes within the three hour time limit, and therefore the bound stops improving after a certain number of nodes, earlier than is the case for the `cc` strategies.

5.4.2 Time spent per node

Table 2 reports the amount of time spent on the different components of the BPC algorithm, on average per node. These numbers are shown for each separation strategy and for each instance class. The columns LP, PP, and SEP represent the average time per node spent on solving LPs, solving pricing problems (heuristically and/or exactly), and executing the cut separation, respectively. The BRANCH column presents the average time needed for approximate strong branching (see Section 5.1.3), and TOTAL gives the average total time spent per node.

The PP column is further detailed by columns PPi for $i \in \{0, 1, 2, 3, 4\}$. PP0 up to PP3 represent the time spent solving different levels of heuristic pricing, each level taking more resources or arcs back into account (see Section 5.1.1). PP4 corresponds to time spent solving the exact pricing problem.

As both LPs and pricing problems are solved during branching, the columns LP, PP, SEP and BRANCH do not add up to TOTAL. In fact, minus some computational overhead, LP, PP and SEP add up to TOTAL approximately.

We see from Table 2 that the total time per node for the `ng1` strategy is similar to that of the `cc` strategies, while the `ng4` and `ngX` strategies require more time. This agrees with the earlier observation in Section 5.4.1 that only separating CCs and adding them as ngCCs does not yield much benefit.

The additional time spent on pricing for `ng4` and `ngX` can in part be explained by the fact that more cuts are found, and therefore the pricing problems are called more often. To isolate this effect, Table 3 presents the average time spent per pricing problem, instead of per node.

It can be seen from Table 3 that the time taken by heuristic pricing is not significantly different between separation strategies, while exact pricing is more expensive for the `ng` strategies. This may be because the unreachable vertices acceleration technique cannot be used when solving the exact pricing problem for the `ng` strategies, as discussed in Section 3.4.1.

Moving back to Table 2, we see that for the A and B instances, separation strategies `ng4` and `ngX` require more time to determine which arc to branch on. Recall that our approximate strong branching rule only solves heuristic pricing problems, which have a similar solution time over all strategies. As such, we attribute this difference to the increase in LP solution time due to the larger number of valid inequalities that are added by `ng4` and `ngX`.

The SEP column demonstrates that the total time spent on separating valid inequalities is small compared to the total time per node. It is clear that increasing κ up to four (`cc4` and `ng4`) but the overall time increase is very small. This supports the claim that κ-scaled separation networks provide a practically efficient way of separating valid inequalities. It is also clear that exact separation (`ccX` and `ngX`) is much more expensive than heuristic separation, especially for the B2 instances.

We would like to point out that while the `ng` strategies require more time per node,

				PP details							
Class	Sep.	LP	PP	PP0	PP1	PP2	PP3	PP4	SEP	BRANCH	TOTAL
A	cc1	8.572	20.786	0.143	1.285	4.269	11.357	3.733	0.008	26.787	34.351
A	cc4	9.890	19.651	0.194	1.252	3.852	10.506	3.848	0.232	26.610	34.536
A	ccX	10.082	19.745	0.182	1.278	3.920	10.540	3.825	0.780	26.858	35.287
A	ng1	8.349	23.948	0.161	1.385	4.496	11.888	6.018	0.009	27.536	37.939
A	ng4	19.766	24.982	0.218	1.482	4.564	11.652	7.066	0.238	37.504	50.816
A	ngX	20.984	24.829	0.210	1.539	4.739	11.573	6.769	1.018	38.757	52.334
B	cc1	8.168	30.488	0.129	1.199	5.452	18.001	5.708	0.008	34.830	46.235
B	cc4	12.611	32.540	0.175	1.265	5.572	19.257	6.271	0.207	40.902	54.025
B	ccX	13.015	30.775	0.158	1.262	5.515	17.766	6.074	0.560	39.051	52.017
B	ng1	10.739	34.777	0.132	1.199	5.470	17.101	10.874	0.010	34.859	53.293
B	ng4	42.356	56.031	0.246	1.957	8.550	27.638	17.638	0.226	79.214	111.671
B	ngX	43.216	54.035	0.234	1.986	8.715	26.150	16.951	0.657	78.157	109.482
B2	cc1	3.009	1.393	0.006	0.087	0.350	0.820	0.130	0.004	4.709	5.274
B2	cc4	3.603	1.427	0.009	0.102	0.363	0.817	0.137	0.156	5.277	6.091
B2	ccX	2.911	1.551	0.010	0.110	0.397	0.884	0.149	4.515	4.705	9.816
B2	ng1	3.241	1.848	0.007	0.112	0.440	1.017	0.272	0.005	5.204	6.024
B2	ng4	3.468	1.999	0.009	0.130	0.471	1.039	0.350	0.204	5.348	6.548
B2	ngX	3.322	2.019	0.010	0.132	0.477	1.049	0.351	3.064	5.201	9.258
P	cc1	2.627	17.459	0.210	0.920	3.051	10.106	3.170	0.006	18.626	25.409
P	cc4	3.139	17.960	0.255	0.962	3.168	10.149	3.428	0.181	19.495	26.985
P	ccX	3.076	18.072	0.248	0.972	3.205	10.236	3.411	0.616	19.500	27.407
P	ng1	2.823	19.456	0.225	0.967	3.239	10.683	4.342	0.007	19.682	28.198
P	ng4	3.954	19.777	0.272	1.038	3.327	10.300	4.839	0.193	20.461	29.857
P	ngX	3.567	20.496	0.278	1.093	3.602	10.789	4.734	1.228	20.922	31.484

Table 2: Average time spent on parts of the BPC algorithm (in seconds per node).

Class	Sep.	PP0	PP1	PP2	PP3	PP4
A	cc1	0.006	0.055	0.256	1.009	2.768
A	cc4	0.008	0.055	0.234	0.936	2.910
A	ccX	0.007	0.055	0.236	0.938	2.860
A	ng1	0.006	0.056	0.257	1.017	4.189
A	ng4	0.007	0.056	0.239	0.948	4.492
A	ngX	0.007	0.055	0.234	0.887	4.155
B	cc1	0.004	0.044	0.279	1.416	4.065
B	cc4	0.005	0.041	0.249	1.354	4.346
B	ccX	0.005	0.042	0.252	1.274	4.185
B	ng1	0.004	0.040	0.247	1.194	6.299
B	ng4	0.005	0.043	0.261	1.422	7.292
B	ngX	0.005	0.044	0.262	1.328	6.994
B2	cc1	0.001	0.008	0.034	0.088	0.109
B2	cc4	0.001	0.009	0.035	0.089	0.113
B2	ccX	0.001	0.009	0.035	0.087	0.120
B2	ng1	0.001	0.009	0.039	0.102	0.214
B2	ng4	0.001	0.011	0.045	0.115	0.254
B2	ngX	0.001	0.011	0.045	0.116	0.249
P	cc1	0.011	0.057	0.247	0.878	2.518
P	cc4	0.013	0.059	0.258	0.890	2.699
P	ccX	0.013	0.060	0.261	0.902	2.656
P	ng1	0.011	0.059	0.261	0.935	3.325
P	ng4	0.013	0.061	0.259	0.883	3.623
P	ngX	0.014	0.065	0.283	0.938	3.479

Table 3: Average time per pricing problem (in seconds).

Figure 3 shows that the additional work is worth the effort. Due to the stronger bounds, the total time for the BPC algorithm does not increase for the A, B, and P instances, and even decreases for the B2 instances. We also note that the CVRP pricing problem is relatively simple, and for problems with a more complicated pricing problem, the advantage of having stronger bounds may be more pronounced.

The findings in this section also provide opportunities to improve performance further. To reduce the amount of time spent on exact pricing, more sophisticated heuristics pricing techniques may be used. For example, tabu search (Desaulniers et al., 2008) may be adapted to take the new dual application functions into account. By having better heuristics, the exact pricing problem, which is relatively slow for the ng4 and ngX

strategies, can be avoided more often.

6 Conclusion

In this paper, we consider valid inequalities for set covering type problems that are solved by branch-price-and-cut, and for which the pricing problem is a SPPRC. This is a broad class of problems with routing and scheduling applications in aviation, shipping, rail, and road transportation.

We introduce *resource-robust* as a new category of valid inequalities between robust and non-robust. Theoretical properties are considered, and it is proven that resource-robust valid inequalities can be incorporated in the BPC algorithm without changing the structure of the pricing problem, under the condition that the resources that the cut is robust for are already included in the SPPRC.

Elementarity-robust and ng-robust are identified as two widely applicable special cases of resource-robust valid inequalities. These valid inequlaities are based on the resources that enforce routes to be elementary or ng-routes, respectively, which are common in practice.

We discuss important practical aspects of using elementarity-robust and ng-robust valid inequalities in combination with common acceleration techniques such as unreachable vertices, dynamic ng-route relaxation, bidirectional labeling, and arc-based ng-memory. The latter three are discussed theoretically, while the first is included in our computational experiments. This choice was made to isolate the effect of the ngCCs without any additional improvements.

An application to the CVRP is provided, for which we derive an ng-robust variant of the well-known rounded capacity cuts. We show with an example that adding these ngCCs instead of the CCs can reduce the optimality gap from 50% to 0%. A separation algorithm for the ngCCs is presented based on the newly introduced κ-scaled separation networks, which allows leveraging separation algorithms for the CCs to separate the ngCCs.

We point out that ng-robust variants of other cuts may be derived in the same way as well. This results in ng-robust (generalized) k-path cuts, subtour elimination constraints, and 2-path cuts. We also mention how to obtain ng-robust strong degree cuts, and how these can be used to enforce ng-routes in the master problem.

To explore the potential benefit of using resource-robust valid inequalities, we perform computational experiments for a relatively simple set partitioning problem with a relatively straightforward BPC algorithm. In particular, we compare CCs and ngCCs on how they affect the performance of the BPC algorithm for solving the CVRP.

In the computational experiments, we observe that ngCCs increase the solution time per node of the search tree, but this is compensated by the stronger bounds that are obtained. For the A, B, and P, instances, the performance for CCs and ngCCs is similar. For the clustered high-demand B2 instances, there is a clear advantage in using ng-robust valid inequalities. We find that especially exact pricing is expensive for ngCCs. Therefore, we note that performance may be improved further by using more sophisticated heuristic pricing techniques that incorporate the new dual application functions, which could be a direction for further research.

We conclude that resource-robust valid inequalities are widely applicable and computationally promising, and there are several interesting avenues for further research. To isolate the effect of the valid inequalities, the computational experiments in this paper consider a relatively simple problem and a straightforward implementation. However, it will be very interesting to see the interaction of resource-robust valid inequalities with other valid inequalities and acceleration techniques in state-of-the-art implementations for different set covering type problems. Another direction would be to derive resource-robust analogues of known valid inequalities, or even find completely new ones.

References

Philippe Augerat. *Approche polyèdrale du problème de tournées de véhicules*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 1995.

Philippe Augerat, José M. Belenguer, Enrique Benavent, Angel Corberán, and Denis Naddef. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, 106(2-3):546–557, 1998.

Lukas Bach, Michel Gendreau, and Sanne Wøhlk. Freight railway operator timetabling and engine scheduling. *European Journal of Operational Research*, 241(2):309–319, 2015.

Lukas Bach, Twan Dollevoet, and Dennis Huisman. Integrating Timetabling and Crew Scheduling at a Freight Railway Operator. *Transportation Science*, 50(3):878–891, 2016.

Roberto Baldacci, Eleni Hadjiconstantinou, and Aristide Mingozzi. An Exact Algorithm for the Capacitated Vehicle Routing Problem Based on a Two-Commodity Network Flow Formulation. *Operations Research*, 52(5):723–738, 2004.

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

Cynthia Barnhart, Natashia L. Boland, Lloyd W. Clarke, Ellis L. Johnson, George L. Nemhauser, and Rajesh G. Shenoi. Flight String Models for Aircraft Fleeting and Routing. *Transportation Science*, 32(3):208–220, 1998a.

Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, 46(3):316–329, 1998b.

Teobaldo Bulhões, Ruslan Sadykov, and Eduardo Uchoa. A branch-and-price algorithm for the Minimum Latency Problem. *Computers & Operations Research*, 93:66–78, 2018.

Claudio Contardo, Jean-François Cordeau, and Bernard Gendron. An Exact Algorithm Based on Cut-and-Column Generation for the Capacitated Location-Routing Problem. *INFORMS Journal on Computing*, 26(1):88–102, 2014.

Luciano Costa, Claudio Contardo, and Guy Desaulniers. Exact Branch-Price-and-Cut Algorithms for Vehicle Routing. *Transportation Science*, page forthcoming, 2019.

George B. Dantzig, Delbert R. Fulkerson, and Selmer M. Johnson. Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors. *Column Generation*. Springer, 2005.

Guy Desaulniers, François Lessard, and Ahmed Hadjar. Tabu Search, Partial Elementarity, and Generalized k-path Inequalities for the Vehicle Routing Problem with Time Windows. *Transportation Science*, 42(3):387–404, 2008.

Martin Desrochers, Jacques Desrosiers, and Marius M. Solomon. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Operations Research*, 40(2):342–354, 1992.

Niklaus Eggenberg, Matteo Salani, and Michel Bierlaire. Constraint-specific recovery network for solving airline recovery problems. *Computers & Operations Research*, 37 (6):1014–1026, 2010.

Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F. Werneck. Robust Branch-and-Cut-and-Price for the Capacitated VehicleRouting Problem. *Mathematical Programming*, 106(3):491–511, 2006.

R. S. Garfinkel and George L. Nemhauser. The Set-Partitioning Problem: Set Covering with Equality Constraints. *Operations Research*, 17(5):848–856, 1969.

Ahmed Ghoniem, Farbod Farhadi, and Mohammad Reihaneh. An accelerated branch-and-price algorithm for multiple-runway aircraft sequencing problems. *European Journal of Operational Research*, 246(1):34–43, 2015.

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

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

Stefan Irnich and Guy Desaulniers. Shortest Path Problems with Resource Constraints. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *Column Generation*, chapter 2, pages 33–65. Springer, 2005.

Stefan Irnich and Daniel Villeneuve. The Shortest-Path Problem with Resource Constraints andk-Cycle Elimination for k $\geq$ 3. *INFORMS Journal on Computing*, 18(3): 391–406, 2006.

Mads Jepsen, Bjørn Petersen, Simon Spoorendonk, and David Pisinger. Subset-Row Inequalities Applied to the Vehicle-Routing Problem with Time Windows. *Operations Research*, 56(2):497–511, 2008.

Niklas Kohl, Jacques Desrosiers, Oli B. G. Madsen, Marius M. Solomon, and François Soumis. 2-Path Cuts for the Vehicle Routing Problem with Time Windows. *Transportation Science*, 33(1):101–116, feb 1999.

Jens Lysgaard. CVRPSEP: A package of separation routines for the Capacitated Vehicle Routing Problem. Working paper, Aarhus School of Business, 2003.

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

Diego Pecin. *Exact Algorithms for the Capacitated VehicleRouting Problem*. PhD thesis, Pontifícia Universidade Católica do Rio de Janiero, 2014.

Diego Pecin, Claudio Contardo, Guy Desaulniers, and Eduardo Uchoa. New Enhancements for the Exact Solution of the Vehicle Routing Problem with Time Windows. *INFORMS Journal on Computing*, 29(3):489–502, 2017a.

Diego Pecin, Artur Pessoa, Marcus Poggi de Aragão, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, 2017b.

Artur Pessoa, Eduardo Uchoa, and Marcus Poggi de Aragão. A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. *Networks: An International Journal*, 54(4):167–177, 2009.

Christian E.M. Plum, David Pisinger, Juan-José Salazar-González, and Mikkel M. Sigurd. Single liner shipping service design. *Computers & Operations Research*, 45:1–6, 2014.

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

Roberto Roberti and Aristide Mingozzi. Dynamic ng-Path Relaxation for the Delivery Man Problem. *Transportation Science*, 48(3):413–424, 2014.

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

Paolo Toth and Daniele Vigo, editors. *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2nd edition, 2014.

Ilaria Vacca, Matteo Salani, and Michel Bierlaire. An Exact Algorithm for the Integrated Planning of Berth Allocation and Quay Crane Assignment. *Transportation Science*, 47 (2):148–161, 2013.

A Restoring in- and outflow of a task

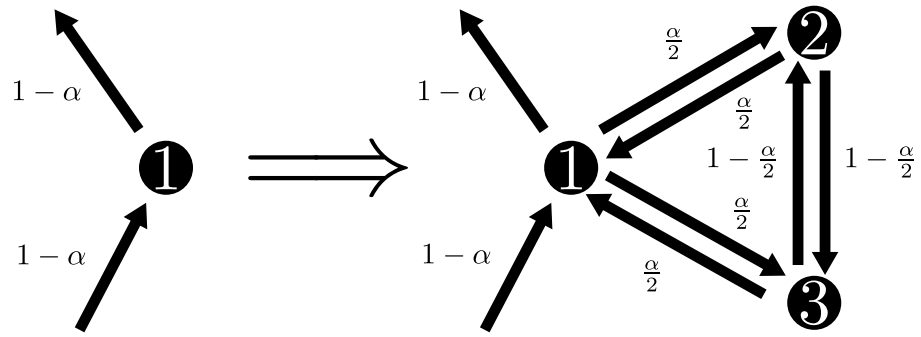


Figure 5: A method to restore the in- and outflow of a task to one, for given $\alpha \in (0, 1)$.

Algorithm 1 Algorithm to restore the in- and outflow all tasks to one

1: **for all** tasks $i \in V'$ **do**
2: Calculate reduced missing in- and outflow $\alpha \equiv 1 - \sum_{r \in \mathcal{R}} a_i^r \bar{x}_r$
3: **if** $\alpha > 0$ **then**
4: Introduce dummy nodes i' and i'' with $q_{i'} = q_{i''} = 0$, with flow on arcs (i, i'), (i, i''), (i', i) and (i'', i) equal to $\frac{\alpha}{2}$ and flow on arcs (i', i'') and (i'', i') equal to $1 - \frac{\alpha}{2}$ (see Figure 5).
5: **end if**
6: **end for**

B Proof Proposition 10

Proposition 10. *For a given $\kappa \in \mathbb{N}$, if a violated ngCC in the original network corresponds to a subset $S \subseteq V'$ that is entered at most κ times by every non-zero route r, then the CC corresponding to S in the separation network is violated as well.*

Proof. Let S be a subset of tasks corresponding to a violated ngCC in the original network. Furthermore, it is entered at most κ times by every non-zero route r. This gives us that $\zeta_r^{CC}(S) \leq \kappa$ for all r with $x_r > 0$. As every route can enter a subset at most $\left\lceil \frac{\#(r)}{2} \right\rceil$ times, we have

$$\zeta_r^{CC}(S) \leq \min \left\{ \kappa, \left\lceil \frac{\#(r)}{2} \right\rceil \right\} \equiv \kappa_r$$

for all r such that $x_r > 0$. Multiplying $\zeta_r^{ng}(S) \geq \zeta_r^{SCC}(S) = \min\{\zeta_r^{CC}(S), 1\}$ with κ_r, we

get the following string of inequalities:

$$\kappa_r \zeta_r^{ng}(S) \geq \min\{\kappa_r \zeta_r^{CC}(S), \kappa_r\} \geq \zeta_r^{CC}(S),$$

where the last inequality follows from the fact that $\kappa_r \geq 1$ and $\zeta_r^{CC}(S) \geq 0$ (so $\zeta_r^{CC}(S) \leq \kappa_r \zeta_r^{CC}(S)$).

As the ngCC in the original network for subset S is violated, we have

$$\sum_{r:x_r>0} \zeta_r^{ng}(S) x_r < \left\lceil \frac{1}{Q} \sum_{i \in S} q_i \right\rceil.$$

Combining this with the previously proven inequality, we get

$$\begin{aligned} \left\lceil \frac{1}{Q} \sum_{i \in S} q_i \right\rceil &> \sum_{r:x_r>0} \zeta_r^{ng}(S) x_r \\ &= \sum_{r:x_r>0} \kappa_r \zeta_r^{ng}(S) \bar{x}_r \\ &\geq \sum_{r:x_r>0} \zeta_r^{CC}(S) \bar{x}_r. \end{aligned}$$

Note that the right-hand-side corresponds to a CC in the κ-scaled network (when augmenting S with its dummy nodes). So, this CC corresponding to S is violated in the κ-scaled network as well. □

C Exact separation of ngCCs and CCs

For the exact separation of CCs and ngCCs, we use the integer programs (16) and (17) respectively. The programs find the most violated CC or ngCC by minimizing the left-hand-side of the valid inequality (13) and (11), while keeping the right-hand-side constant.

To separate a CC or ngCC exactly, we execute the program a number of times, each for a different value of θ, which represents $\left\lceil \frac{1}{Q} \sum_{i \in T} q_i \right\rceil$, the right-hand-side of the cut. As θ is integer, we execute the program for $\theta \in \left\{1, 2, \ldots, \left\lceil \frac{1}{Q} \sum_{i \in V} q_i \right\rceil \right\}$. If in any of the stages a violated cut is found, we return that cut and terminate the separation.

Otherwise, no violated cuts are present in the current solution.

$$\min \quad \sum_{(i,j)\in A} x_{ij} z_{ij}, \tag{16a}$$

$$\text{s.t.} \quad z_{ij} \geq y_j - y_i \quad \forall (i,j) \in A, \tag{16b}$$

$$(\theta - 1)Q + 1 \leq \sum_{i \in V} q_i y_i \quad , \tag{16c}$$

$$\theta Q \geq \sum_{i \in V} q_i y_i \quad , \tag{16d}$$

$$y_0 = y_{n+1} = 0 \quad , \tag{16e}$$

$$z_{ij}, y_i \in \{0,1\} \quad \forall (i,j) \in A. \tag{16f}$$

In (16), x_{ij}, $(i,j) \in A$ represents the arc flows of the current (not necessarily integer) solution to the RMP $(x_r)_{r\in\mathcal{R}}$, with $x_{ij} = \sum_{r\in\mathcal{R}} b_{ij}^r x_r$. The binary decision variables y_i equals 1 if task i is in the subset S of the CC and equals 0 otherwise. The binary decision variable z_{ij} equals 1 if arc (i,j) enters this set S and 0 otherwise.

Equation (16a) is the objective: it counts the number of vehicles entering set S. Equation (16b) makes sure that z_{ij} equals 1 if $y_j = 1$ and $y_i = 0$. The constraint is inactive otherwise. Next, constraints (16c) and (16d) make sure that θ equals $\left\lceil \frac{1}{Q} \sum_{i\in S} q_i \right\rceil$, where we add 1 to the left-hand-side of (16c) as we assume all q_i to be integer. Constraint (16e) ensures that the depots 0 and $n+1$ cannot be in set S. Finally, constraint (16f) restricts the variables to be binary.

$$\min \quad \sum_{r\in\mathcal{R}}\sum_{k=1}^{\#(r)} x_r z_{rk}, \tag{17a}$$

$$\text{s.t.} \quad z_{rk} + \sum_{i\in\Pi_r(k-1)} y_i \geq y_{r_k} \qquad \forall r \in \mathcal{R}, k \in \{1,\ldots,\#(r)\} \tag{17b}$$

$$(\theta-1)Q + 1 \leq \sum_{i\in V} q_i y_i \quad , \tag{17c}$$

$$\theta Q \geq \sum_{i\in V} q_i y_i \quad , \tag{17d}$$

$$y_0 = y_{n+1} = 0 \quad , \tag{17e}$$

$$z_{rk}, y_i \in \{0,1\} \qquad \forall i \in V, r \in \mathcal{R}, k \in \{1,\ldots,\#(r)\} \tag{17f}$$

In (17), x_r represents the current (not necessarily integer) solution to the RMP. Again, the binary decision variable y_i equals 1 if task i is in the subset S of the ngCC and 0 otherwise. The binary variable z_{rk} equals 1 if route r travels into S as the kth task on the route for the first time, according to its ng-memory. Here, $k \in \{1,\ldots,\#(r)\}$, where $\#(r)$ is the number of tasks on the route. Furthermore, let r_k be the kth task on the route, where we set $r_0 = 0$ and $r_{\#(r)+1} = n+1$ the starting and ending depot respectively. We also define $\Pi_r(k)$ is the ng-memory of route r at its kth task.

Equation (17a) is the objective: it counts the number of vehicles entering subset S for the first time, according to ng-memory. Constraint (17b) makes sure that z_{rk} is forced to 1 if its kth task is in S and all tasks in its current ng-memory are not in S. Next, constraints (17c)-(17f) are identical to their counterparts (16c)-(16f).

D List of computational results

Instance	Method	LB	UB	Gap	Time	Nodes	Routes	CCs	ngCCs
B-n31-k5	cc1	672.00	672	0.000	5.12	1	3488	30	0
	cc4	672.00	672	0.000	5.43	1	3583	39	0
	ccX	672.00	672	0.000	5.48	1	3583	39	0
	ng1	672.00	672	0.000	5.84	1	3601	20	10
	ng4	672.00	672	0.000	10.90	1	4666	28	33
	ngX	672.00	672	0.000	10.96	1	4666	28	33
B-n34-k5	cc1	788.00	788	0.000	353.83	75	10575	84	0
	cc4	788.00	788	0.000	739.50	83	17285	143	0
	ccX	788.00	788	0.000	743.29	83	17285	143	0
	ng1	788.00	788	0.000	644.77	75	13701	77	15
	ng4	788.00	788	0.000	1810.69	33	19618	109	369
	ngX	788.00	788	0.000	1857.84	33	19258	104	360
B-n35-k5	cc1	955.00	955	0.000	12.03	1	4428	73	0
	cc4	955.00	955	0.000	12.07	1	4143	111	0
	ccX	955.00	955	0.000	12.05	1	4143	111	0
	ng1	955.00	955	0.000	13.61	1	4377	51	31
	ng4	955.00	955	0.000	22.19	1	5734	73	166
	ngX	955.00	955	0.000	22.37	1	5734	73	166
B-n38-k6	cc1	805.00	805	0.000	199.47	31	6663	48	0
	cc4	805.00	805	0.000	96.13	13	6053	53	0
	ccX	805.00	805	0.000	98.35	13	6053	53	0
	ng1	805.00	805	0.000	80.37	11	5813	41	6
	ng4	805.00	805	0.000	112.97	11	6441	38	35
	ngX	805.00	805	0.000	115.48	11	6441	38	35
B-n39-k5	cc1	549.00	549	0.000	43.53	1	5657	41	0
	cc4	549.00	549	0.000	36.70	1	6105	95	0
	ccX	549.00	549	0.000	36.65	1	6105	95	0
	ng1	549.00	549	0.000	71.22	1	5653	39	2
	ng4	549.00	549	0.000	144.52	1	7732	82	262
	ngX	549.00	549	0.000	144.03	1	7732	82	262
B-n41-k6	cc1	829.00	829	0.000	115.38	13	7648	57	0
	cc4	829.00	829	0.000	122.04	13	8810	72	0
	ccX	829.00	829	0.000	124.71	13	8810	72	0
	ng1	829.00	829	0.000	239.57	19	9387	45	12
	ng4	829.00	829	0.000	198.60	21	7062	57	26
	ngX	829.00	829	0.000	219.73	23	6973	57	26
B-n43-k6	cc1	742.00	742	0.000	533.33	44	9843	153	0
	cc4	742.00	742	0.000	584.71	41	11849	127	0
	ccX	742.00	742	0.000	379.38	22	10430	128	0
	ng1	742.00	742	0.000	381.04	27	8099	130	26
	ng4	742.00	742	0.000	413.38	15	10698	119	118
	ngX	742.00	742	0.000	419.05	15	10698	119	118
B-n44-k7	cc1	909.00	909	0.000	28.52	2	4408	81	0
	cc4	909.00	909	0.000	13.35	1	4953	138	0
	ccX	909.00	909	0.000	13.35	1	4953	138	0
	ng1	909.00	909	0.000	33.16	1	4703	49	38
	ng4	909.00	909	0.000	20.89	1	5112	86	230
	ngX	909.00	909	0.000	21.09	1	5112	86	230
B-n45-k5	cc1	751.00	751	0.000	131.23	7	7927	36	0
	cc4	751.00	751	0.000	423.38	15	12017	57	0
	ccX	751.00	751	0.000	428.44	15	12017	57	0
	ng1	751.00	751	0.000	374.76	9	11886	42	2
	ng4	751.00	751	0.000	56.23	1	6403	49	52
	ngX	751.00	751	0.000	56.10	1	6403	49	52
B-n45-k6	cc1	678.00	678	0.000	90.79	5	10984	33	0
	cc4	678.00	678	0.000	74.63	5	10649	31	0
	ccX	678.00	678	0.000	75.31	5	10649	31	0
	ng1	678.00	678	0.000	75.39	5	7677	24	11
	ng4	678.00	678	0.000	94.67	5	10898	22	24
	ngX	678.00	678	0.000	96.40	5	10898	22	24
B-n50-k7	cc1	741.00	741	0.000	50.09	2	6844	62	0
	cc4	741.00	741	0.000	36.91	1	6630	190	0
	ccX	741.00	741	0.000	36.91	1	6630	190	0
	ng1	741.00	741	0.000	59.55	2	7112	50	11
	ng4	741.00	741	0.000	67.41	1	9247	42	72
	ngX	741.00	741	0.000	67.16	1	9247	42	72
B-n50-k8	cc1	1312.00	1312	0.000	6479.26	196	30242	410	0
	cc4	1312.00	1312	0.000	7399.97	200	30163	477	0
	ccX	1312.00	1312	0.000	8030.39	210	31259	496	0
	ng1	1312.00	1312	0.000	10468.20	205	37394	285	128
	ng4	1311.79	Inf	Inf	10843.90	115	30196	267	460
	ngX	1311.77	Inf	Inf	10883.80	118	30847	287	475
B-n51-k7	cc1	1016.00	1016	0.000	37.29	1	5927	25	0
	cc4	1016.00	1016	0.000	16.92	1	5762	65	0
	ccX	1016.00	1016	0.000	17.04	1	5762	65	0
	ng1	1016.00	1016	0.000	48.64	1	5849	23	4
	ng4	1016.00	1016	0.000	38.48	1	6899	55	48
	ngX	1016.00	1016	0.000	38.39	1	6899	55	48
B-n52-k7	cc1	747.00	747	0.000	1389.19	37	23102	61	0
	cc4	747.00	747	0.000	2049.97	52	25595	101	0
	ccX	747.00	747	0.000	2045.04	52	25595	101	0
	ng1	747.00	747	0.000	2496.77	48	21897	53	8
	ng4	747.00	747	0.000	2391.45	34	24593	119	47
	ngX	747.00	747	0.000	2923.65	46	24342	119	50
B-n56-k7	cc1	707.00	707	0.000	533.14	15	14892	52	0
	cc4	707.00	707	0.000	968.36	27	13042	81	0
	ccX	707.00	707	0.000	973.51	27	13042	81	0
	ng1	707.00	707	0.000	662.60	13	14670	42	17
	ng4	707.00	707	0.000	679.82	9	13462	48	124
	ngX	707.00	707	0.000	689.48	9	13462	48	124
B-n57-k7	cc1	1140.00	1140	0.000	6260.07	157	27777	83	0
	cc4	1140.00	1140	0.000	1226.55	31	14313	96	0
	ccX	1140.00	1140	0.000	1213.56	31	14313	96	0
	ng1	1140.00	1140	0.000	8327.71	157	31822	56	38
	ng4	1140.00	1140	0.000	5283.59	49	26902	112	88
	ngX	1140.00	1140	0.000	4655.95	41	26515	112	87
B-n57-k9	cc1	1598.00	1598	0.000	646.57	27	12986	266	0
	cc4	1598.00	1598	0.000	570.54	17	13436	337	0
	ccX	1598.00	1598	0.000	568.15	17	13436	337	0
	ng1	1598.00	1598	0.000	645.49	23	12401	254	34
	ng4	1598.00	1598	0.000	766.56	21	11876	295	123
	ngX	1598.00	1598	0.000	768.28	21	11876	295	123
B-n63-k10	cc1	1496.00	1496	0.000	741.03	27	11066	248	0
	cc4	1496.00	1496	0.000	383.68	15	8191	232	0
	ccX	1496.00	1496	0.000	459.57	19	8285	239	0
	ng1	1496.00	1496	0.000	1923.47	41	12480	179	44
	ng4	1496.00	1496	0.000	1950.86	27	11963	186	175
	ngX	1496.00	1496	0.000	588.09	5	9310	173	177
B-n64-k9	cc1	861.00	861	0.000	978.23	25	17792	95	0
	cc4	861.00	861	0.000	712.31	13	14337	235	0
	ccX	861.00	861	0.000	710.90	13	14337	235	0
	ng1	861.00	861	0.000	462.97	9	12594	112	13
	ng4	861.00	861	0.000	867.99	9	16944	226	58
	ngX	861.00	861	0.000	1013.61	11	18801	223	59
B-n66-k9	cc1	1316.00	1316	0.000	9739.79	259	29688	403	0
	cc4	1314.68	1317	0.176	10847.60	189	30831	471	0
	ccX	1314.68	Inf	Inf	10843.10	187	32803	463	0
	ng1	1316.00	1316	0.000	9309.55	223	26994	317	91
	ng4	1313.31	Inf	Inf	10814.60	86	23642	317	612
	ngX	1313.39	Inf	Inf	10886.30	89	23978	317	613
B-n67-k10	cc1	1032.00	1032	0.000	4345.03	71	19608	208	0
	cc4	1032.00	1032	0.000	3594.02	71	14897	202	0
	ccX	1032.00	1032	0.000	3046.15	71	14766	202	0
	ng1	1032.00	1032	0.000	2688.42	39	13866	136	51
	ng4	1032.00	1032	0.000	3561.24	40	13301	98	96
	ngX	1032.00	1032	0.000	2854.72	36	12326	105	99
B-n68-k9	cc1	1266.81	Inf	Inf	10948.70	96	30599	286	0
	cc4	1266.81	Inf	Inf	10813.40	90	27190	262	0
	ccX	1266.94	Inf	Inf	10833.90	103	28619	263	0
	ng1	1267.69	Inf	Inf	10839.10	69	26289	210	102
	ng4	1266.92	Inf	Inf	10896.60	34	26383	166	482
	ngX	1266.94	Inf	Inf	10961.20	36	26951	179	482
B-n78-k10	cc1	1220.28	Inf	Inf	10896.20	87	22659	226	0
	cc4	1219.55	1221	0.119	10865.40	74	20548	340	0
	ccX	1221.00	1221	0.000	10593.80	99	20599	343	0
	ng1	1221.00	1221	0.000	4879.71	47	15201	184	36
	ng4	1220.31	Inf	Inf	10818.60	38	19270	270	491
	ngX	1219.94	Inf	Inf	10921.50	44	20479	265	493

Instance	Method	LB	UB	Gap	Time	Nodes	Routes	CCs	*ng*CCs
B2-n31-k5	cc1	1100.00	1100	0.000	0.30	1	667	39	0
	cc4	1100.00	1100	0.000	0.52	1	750	51	0
	ccX	1100.00	1100	0.000	0.67	1	750	51	0
	ng1	1100.00	1100	0.000	0.34	1	670	34	7
	ng4	1100.00	1100	0.000	1.52	3	1081	70	17
	ngX	1100.00	1100	0.000	0.88	1	992	53	14
B2-n34-k5	cc1	1262.00	1262	0.000	0.65	1	1333	42	0
	cc4	1262.00	1262	0.000	0.58	1	1496	55	0
	ccX	1262.00	1262	0.000	0.62	1	1496	55	0
	ng1	1262.00	1262	0.000	0.84	1	1384	41	2
	ng4	1262.00	1262	0.000	1.60	1	1682	55	80
	ngX	1262.00	1262	0.000	1.73	1	1682	55	80
B2-n35-k5	cc1	1628.00	1628	0.000	948.04	667	8086	355	0
	cc4	1628.00	1628	0.000	960.02	573	8956	419	0
	ccX	1628.00	1628	0.000	1138.86	593	9540	419	0
	ng1	1628.00	1628	0.000	1109.39	645	8591	311	56
	ng4	1628.00	1628	0.000	322.96	259	4780	146	72
	ngX	1628.00	1628	0.000	300.64	203	4546	150	81
B2-n38-k6	cc1	1338.00	1338	0.000	30.37	37	1808	119	0
	cc4	1338.00	1338	0.000	35.21	49	1886	121	0
	ccX	1338.00	1338	0.000	42.19	49	1886	121	0
	ng1	1338.00	1338	0.000	22.64	25	1682	130	19
	ng4	1338.00	1338	0.000	34.56	35	1897	120	48
	ngX	1338.00	1338	0.000	40.67	35	1897	120	48
B2-n39-k5	cc1	952.00	952	0.000	2.13	1	1907	40	0
	cc4	952.00	952	0.000	1.90	1	1972	63	0
	ccX	952.00	952	0.000	1.98	1	1972	63	0
	ng1	952.00	952	0.000	2.51	1	2026	40	3
	ng4	952.00	952	0.000	2.42	1	2013	53	30
	ngX	952.00	952	0.000	2.49	1	2013	53	30
B2-n41-k6	cc1	1477.00	1477	0.000	38.66	40	2287	158	0
	cc4	1477.00	1477	0.000	36.78	37	2144	183	0
	ccX	1477.00	1477	0.000	45.09	37	2144	183	0
	ng1	1477.00	1477	0.000	30.75	34	2082	104	15
	ng4	1477.00	1477	0.000	13.90	17	1710	142	42
	ngX	1477.00	1477	0.000	22.29	21	1738	132	43
B2-n43-k6	cc1	1223.00	1223	0.000	48.81	27	2774	218	0
	cc4	1223.00	1223	0.000	74.21	29	3716	178	0
	ccX	1223.00	1223	0.000	75.14	29	3613	165	0
	ng1	1223.00	1223	0.000	111.45	45	3758	150	45
	ng4	1223.00	1223	0.000	196.17	51	4833	188	94
	ngX	1223.00	1223	0.000	94.57	39	2768	177	97
B2-n44-k7	cc1	1592.00	1592	0.000	56.53	61	2954	186	0
	cc4	1592.00	1592	0.000	76.71	71	3329	196	0
	ccX	1592.00	1592	0.000	99.00	71	3329	196	0
	ng1	1592.00	1592	0.000	78.13	61	3410	174	33
	ng4	1592.00	1592	0.000	92.42	41	3340	160	104
	ngX	1592.00	1592	0.000	67.19	27	2707	150	104
B2-n45-k5	cc1	1145.00	1145	0.000	49.20	25	2875	180	0
	cc4	1145.00	1145	0.000	36.58	19	2562	139	0
	ccX	1145.00	1145	0.000	38.98	19	2562	139	0
	ng1	1145.00	1145	0.000	52.65	21	2916	150	48
	ng4	1145.00	1145	0.000	31.02	11	2318	101	70
	ngX	1145.00	1145	0.000	32.47	11	2318	101	70
B2-n45-k6	cc1	1165.00	1165	0.000	913.57	259	5580	438	0
	cc4	1165.00	1165	0.000	1112.07	351	5275	633	0
	ccX	1165.00	1165	0.000	1237.90	351	5275	633	0
	ng1	1165.00	1165	0.000	667.29	183	4963	378	56
	ng4	1165.00	1165	0.000	832.81	199	4173	405	136
	ngX	1165.00	1165	0.000	811.60	199	4010	394	151
B2-n50-k7	cc1	1204.00	1204	0.000	3.02	1	1710	127	0
	cc4	1204.00	1204	0.000	2.73	1	1850	95	0
	ccX	1204.00	1204	0.000	2.84	1	1850	95	0
	ng1	1204.00	1204	0.000	3.62	1	1766	98	48
	ng4	1204.00	1204	0.000	4.20	1	2099	72	39
	ngX	1204.00	1204	0.000	4.33	1	2099	72	39
B2-n50-k8	cc1	2181.00	2181	0.000	1096.98	395	4595	613	0
	cc4	2181.00	2181	0.000	2028.96	447	5390	822	0
	ccX	2181.00	2181	0.000	1277.62	271	4447	731	0
	ng1	2181.00	2181	0.000	1529.76	383	4766	660	58
	ng4	2181.00	2181	0.000	607.68	175	3736	491	164
	ngX	2181.00	2181	0.000	563.23	153	3575	465	171
B2-n51-k7	cc1	1783.10	1793	0.555	10802.90	4211	11390	739	0
	cc4	1786.00	1786	0.000	10093.90	3515	11715	817	0
	ccX	1785.15	1789	0.216	10802.80	2279	11246	689	0
	ng1	1786.00	1786	0.000	1243.18	849	5294	257	21
	ng4	1786.00	1786	0.000	1951.44	1065	7193	302	175
	ngX	1786.00	1786	0.000	2823.52	1143	7185	305	196
B2-n52-k7	cc1	1242.00	1242	0.000	88.06	41	4447	176	0
	cc4	1242.00	1242	0.000	96.71	40	3761	190	0
	ccX	1242.00	1242	0.000	114.43	40	3761	190	0
	ng1	1242.00	1242	0.000	145.14	43	4860	181	13
	ng4	1242.00	1242	0.000	53.38	21	3026	142	103
	ngX	1242.00	1242	0.000	71.05	19	2969	141	105
B2-n56-k7	cc1	1202.00	1202	0.000	323.29	55	6728	251	0
	cc4	1202.00	1202	0.000	280.69	39	6118	329	0
	ccX	1202.00	1202	0.000	418.75	51	6999	344	0
	ng1	1202.00	1202	0.000	445.14	53	8195	242	31
	ng4	1202.00	1202	0.000	238.99	35	6002	267	86
	ngX	1202.00	1202	0.000	238.81	35	6034	223	76
B2-n57-k7	cc1	2065.00	2065	0.000	1476.66	284	10081	272	0
	cc4	2065.00	2065	0.000	1162.71	178	11567	305	0
	ccX	2065.00	2065	0.000	1277.27	205	10324	295	0
	ng1	2065.00	2065	0.000	702.36	134	7882	237	37
	ng4	2065.00	2065	0.000	629.99	101	8930	222	87
	ngX	2065.00	2065	0.000	371.79	52	6867	177	79
B2-n57-k9	cc1	2894.00	2894	0.000	431.90	163	3849	442	0
	cc4	2894.00	2894	0.000	559.39	235	3796	469	0
	ccX	2894.00	2894	0.000	672.53	239	3578	443	0
	ng1	2894.00	2894	0.000	678.01	227	3829	476	63
	ng4	2894.00	2894	0.000	163.56	55	2764	267	60
	ngX	2894.00	2894	0.000	150.80	55	2614	311	71
B2-n63-k10	cc1	2736.29	2739	0.099	10805.40	2075	7209	1283	0
	cc4	2735.58	2742	0.235	10814.70	1387	7160	1497	0
	ccX	2732.90	Inf	Inf	10827.50	471	5499	1004	0
	ng1	2738.00	2738	0.000	10091.30	2223	8417	1167	100
	ng4	2738.00	2738	0.000	3855.63	1029	4814	1094	274
	ngX	2736.50	2738	0.055	10820.40	575	4443	1009	253
B2-n64-k9	cc1	1539.00	1539	0.000	1386.91	379	7579	509	0
	cc4	1539.00	1539	0.000	5611.37	853	9411	871	0
	ccX	1537.79	1540	0.144	10964.90	672	8801	794	0
	ng1	1539.00	1539	0.000	2480.02	551	8211	609	54
	ng4	1539.00	1539	0.000	1301.28	199	6126	572	149
	ngX	1539.00	1539	0.000	1475.58	235	5938	512	123
B2-n66-k9	cc1	2312.61	Inf	Inf	10812.30	1011	11489	995	0
	cc4	2314.39	2318	0.156	10835.30	1183	12226	1001	0
	ccX	2311.32	2318	0.289	10824.90	692	8997	912	0
	ng1	2312.63	2320	0.319	10810.20	1255	16881	659	110
	ng4	2316.00	2316	0.000	4925.53	725	7836	765	200
	ngX	2316.00	2316	0.000	8246.75	781	9275	728	193
B2-n67-k10	cc1	1776.00	1776	0.000	3736.07	735	11494	592	0
	cc4	1776.00	1776	0.000	4685.64	691	10884	904	0
	ccX	1776.00	1776	0.000	7963.59	611	9939	860	0
	ng1	1776.00	1776	0.000	1268.18	291	7459	374	84
	ng4	1776.00	1776	0.000	1547.87	181	7060	429	191
	ngX	1776.00	1776	0.000	766.30	127	5332	326	196
B2-n68-k9	cc1	2296.64	Inf	Inf	10806.60	986	12580	548	0
	cc4	2296.35	Inf	Inf	10813.90	951	12629	599	0
	ccX	2296.08	Inf	Inf	10808.50	824	11759	566	0
	ng1	2297.43	Inf	Inf	10801.50	867	12933	478	90
	ng4	2297.94	Inf	Inf	10801.40	771	12663	455	162
	ngX	2298.01	Inf	Inf	10815.80	742	12360	414	169
B2-n78-k10	cc1	2108.92	Inf	Inf	10804.00	805	15429	715	0
	cc4	2109.31	Inf	Inf	10801.70	860	15192	733	0
	ccX	2108.65	Inf	Inf	10813.10	586	12555	600	0
	ng1	2110.43	Inf	Inf	10806.50	918	13419	541	94
	ng4	2112.30	2114	0.080	10804.30	891	11681	555	221
	ngX	2112.73	2114	0.060	10816.60	787	12316	539	212

Instance	Method	LB	UB	Gap	Time	Nodes	Routes	CCs	*ng*CCs
A-n32-k5	cc1	784.00	784	0.000	27.48	3	3909	58	0
	cc4	784.00	784	0.000	5.15	1	2795	70	0
	ccX	784.00	784	0.000	5.26	1	2795	70	0
	ng1	784.00	784	0.000	29.01	3	4449	53	2
	ng4	784.00	784	0.000	7.70	1	2910	67	21
	ngX	784.00	784	0.000	7.67	1	2910	67	21
A-n33-k5	cc1	661.00	661	0.000	4.50	1	1904	43	0
	cc4	661.00	661	0.000	3.93	1	1909	56	0
	ccX	661.00	661	0.000	3.88	1	1909	56	0
	ng1	661.00	661	0.000	3.96	1	1913	24	19
	ng4	661.00	661	0.000	4.27	1	1897	36	202
	ngX	661.00	661	0.000	4.37	1	1897	36	202
A-n33-k6	cc1	742.00	742	0.000	34.32	13	1961	42	0
	cc4	742.00	742	0.000	74.27	25	2238	90	0
	ccX	742.00	742	0.000	77.58	25	2238	90	0
	ng1	742.00	742	0.000	43.74	17	2344	36	4
	ng4	742.00	742	0.000	63.94	21	2302	70	118
	ngX	742.00	742	0.000	70.72	21	2302	70	118
A-n34-k5	cc1	778.00	778	0.000	169.76	27	6387	89	0
	cc4	778.00	778	0.000	84.47	13	4204	128	0
	ccX	778.00	778	0.000	86.22	13	4204	128	0
	ng1	778.00	778	0.000	100.48	15	4487	82	5
	ng4	778.00	778	0.000	262.38	21	5544	102	294
	ngX	778.00	778	0.000	182.99	23	5225	87	41
A-n36-k5	cc1	799.00	799	0.000	78.09	5	6037	83	0
	cc4	799.00	799	0.000	64.62	5	6282	112	0
	ccX	799.00	799	0.000	65.86	5	6282	112	0
	ng1	799.00	799	0.000	94.15	5	6119	48	26
	ng4	799.00	799	0.000	36.37	1	4621	76	221
	ngX	799.00	799	0.000	36.66	1	4621	76	221
A-n37-k5	cc1	669.00	669	0.000	141.89	11	5572	56	0
	cc4	669.00	669	0.000	132.75	9	5691	131	0
	ccX	669.00	669	0.000	133.78	9	5691	131	0
	ng1	669.00	669	0.000	288.79	23	7576	66	7
	ng4	669.00	669	0.000	269.72	17	6824	143	269
	ngX	669.00	669	0.000	278.15	17	6761	143	270
A-n37-k6	cc1	949.00	949	0.000	1014.94	219	10492	166	0
	cc4	949.00	949	0.000	1003.12	223	10623	155	0
	ccX	949.00	949	0.000	879.73	203	10586	155	0
	ng1	949.00	949	0.000	1018.93	221	11267	152	11
	ng4	949.00	949	0.000	758.94	144	9062	118	50
	ngX	949.00	949	0.000	986.18	161	10556	129	60
A-n38-k5	cc1	730.00	730	0.000	288.27	43	8743	81	0
	cc4	730.00	730	0.000	364.15	53	8424	98	0
	ccX	730.00	730	0.000	367.83	53	8424	98	0
	ng1	730.00	730	0.000	213.84	27	7942	82	9
	ng4	730.00	730	0.000	291.93	43	8126	89	22
	ngX	730.00	730	0.000	476.35	54	9956	86	21
A-n39-k5	cc1	822.00	822	0.000	489.85	33	12341	126	0
	cc4	822.00	822	0.000	476.29	27	11288	258	0
	ccX	822.00	822	0.000	486.95	27	11288	258	0
	ng1	822.00	822	0.000	597.22	27	12613	140	43
	ng4	822.00	822	0.000	759.76	25	10588	219	243
	ngX	822.00	822	0.000	1253.13	33	14389	240	243

Instance	Method	LB	UB	Gap	Time	Nodes	Routes	CCs	*ng*CCs
A-n39-k6	cc1	831.00	831	0.000	284.00	35	6128	152	0
	cc4	831.00	831	0.000	292.33	29	6400	155	0
	ccX	831.00	831	0.000	292.12	29	6402	152	0
	ng1	831.00	831	0.000	453.19	39	6642	122	29
	ng4	831.00	831	0.000	538.69	41	8001	141	242
	ngX	831.00	831	0.000	256.52	21	4128	127	237
A-n44-k6	cc1	937.00	937	0.000	109.24	9	5958	92	0
	cc4	937.00	937	0.000	92.15	11	4605	156	0
	ccX	937.00	937	0.000	93.92	11	4605	156	0
	ng1	937.00	937	0.000	107.36	9	4700	80	3
	ng4	937.00	937	0.000	104.06	7	5097	186	11
	ngX	937.00	937	0.000	163.22	11	5197	185	14
A-n45-k6	cc1	944.00	944	0.000	303.08	25	7701	101	0
	cc4	944.00	944	0.000	358.57	27	7690	126	0
	ccX	944.00	944	0.000	359.71	27	7690	126	0
	ng1	944.00	944	0.000	398.95	25	9350	94	24
	ng4	944.00	944	0.000	503.03	33	7860	132	25
	ngX	944.00	944	0.000	511.59	33	7860	132	25
A-n45-k7	cc1	1146.00	1146	0.000	666.00	43	8956	264	0
	cc4	1146.00	1146	0.000	590.60	35	9234	294	0
	ccX	1146.00	1146	0.000	508.79	43	7543	288	0
	ng1	1146.00	1146	0.000	572.59	41	8338	250	15
	ng4	1146.00	1146	0.000	1333.53	45	10035	282	241
	ngX	1146.00	1146	0.000	1319.19	55	9619	256	240
A-n46-k7	cc1	914.00	914	0.000	26.44	2	3688	119	0
	cc4	914.00	914	0.000	18.63	2	3645	87	0
	ccX	914.00	914	0.000	18.65	2	3645	87	0
	ng1	914.00	914	0.000	23.00	2	3963	115	22
	ng4	914.00	914	0.000	23.97	2	3765	64	37
	ngX	914.00	914	0.000	24.36	2	3765	64	37
A-n48-k7	cc1	1073.00	1073	0.000	316.16	15	7525	223	0
	cc4	1073.00	1073	0.000	240.86	11	7235	254	0
	ccX	1073.00	1073	0.000	242.21	11	7235	254	0
	ng1	1073.00	1073	0.000	215.01	9	7130	177	50
	ng4	1073.00	1073	0.000	402.16	11	8298	178	189
	ngX	1073.00	1073	0.000	324.27	11	6870	176	188
A-n53-k7	cc1	1010.00	1010	0.000	1345.66	73	12950	173	0
	cc4	1010.00	1010	0.000	800.61	47	9893	130	0
	ccX	1010.00	1010	0.000	796.88	47	9997	133	0
	ng1	1010.00	1010	0.000	898.58	19	9724	216	19
	ng4	1010.00	1010	0.000	911.49	34	8952	96	14
	ngX	1010.00	1010	0.000	1159.90	37	9311	113	17
A-n54-k7	cc1	1167.00	1167	0.000	6652.65	179	29292	299	0
	cc4	1167.00	1167	0.000	4602.11	161	20862	297	0
	ccX	1167.00	1167	0.000	4754.10	159	22252	284	0
	ng1	1167.00	1167	0.000	5007.19	115	24292	273	14
	ng4	1167.00	1167	0.000	8176.80	215	24400	349	52
	ngX	1167.00	1167	0.000	8657.26	221	26345	372	55
A-n55-k9	cc1	1073.00	1073	0.000	494.54	27	6752	193	0
	cc4	1073.00	1073	0.000	619.20	51	5835	197	0
	ccX	1073.00	1073	0.000	616.73	51	5835	197	0
	ng1	1073.00	1073	0.000	673.76	41	6247	147	29
	ng4	1073.00	1073	0.000	1464.82	37	9945	186	350
	ngX	1073.00	1073	0.000	961.45	31	7834	189	351

Instance	Method	LB	UB	Gap	Time	Nodes	Routes	CCs	*ng*CCs
A-n60-k9	cc1	1353.22	1363	0.723	10826.40	301	27794	287	0
	cc4	1354.00	1354	0.000	9956.14	357	23815	362	0
	ccX	1354.00	1354	0.000	10580.70	361	24196	362	0
	ng1	1354.00	1354	0.000	7909.73	265	20719	225	78
	ng4	1354.00	1354	0.000	6496.41	147	17144	234	355
	ngX	1353.69	Inf	Inf	10808.10	180	23630	224	354
A-n61-k9	cc1	1031.40	Inf	Inf	10839.30	349	21006	362	0
	cc4	1030.54	1035	0.433	10836.10	355	23497	337	0
	ccX	1030.33	Inf	Inf	10819.10	344	22519	288	0
	ng1	1030.48	Inf	Inf	10816.50	310	21156	326	27
	ng4	1030.85	Inf	Inf	10806.90	306	19924	276	77
	ngX	1030.65	Inf	Inf	10839.80	291	21121	311	68
A-n62-k8	cc1	1288.00	1288	0.000	10209.00	139	28972	329	0
	cc4	1288.00	1288	0.000	4448.79	91	17255	344	0
	ccX	1288.00	1288	0.000	4029.16	89	17019	335	0
	ng1	1288.00	1288	0.000	4467.13	61	19752	251	54
	ng4	1288.00	1288	0.000	9006.60	95	20385	308	341
	ngX	1288.00	1288	0.000	6879.25	73	19638	280	342
A-n63-k9	cc1	1616.00	1616	0.000	4817.65	141	19890	272	0
	cc4	1616.00	1616	0.000	9451.38	206	25622	353	0
	ccX	1616.00	1616	0.000	9278.26	202	25929	344	0
	ng1	1616.00	1616	0.000	4483.58	148	15714	208	59
	ng4	1615.20	1617	0.111	10874.60	141	20984	230	456
	ngX	1615.20	1617	0.111	10893.40	141	20984	230	456
A-n63-k10	cc1	1312.95	1314	0.080	10812.50	527	20952	325	0
	cc4	1311.67	Inf	Inf	10835.10	366	23586	285	0
	ccX	1311.67	Inf	Inf	10825.30	333	24982	311	0
	ng1	1311.33	Inf	Inf	10812.80	325	23570	208	31
	ng4	1311.93	1314	0.158	10804.30	261	20221	234	312
	ngX	1311.62	Inf	Inf	10833.40	224	21214	219	311
A-n64-k9	cc1	1397.25	Inf	Inf	10837.40	232	26664	283	0
	cc4	1397.55	Inf	Inf	10802.80	214	28570	378	0
	ccX	1397.53	Inf	Inf	10838.40	215	28474	380	0
	ng1	1397.35	Inf	Inf	10849.00	195	23826	301	36
	ng4	1396.52	Inf	Inf	10899.60	115	19867	281	461
	ngX	1396.63	Inf	Inf	10803.60	117	21337	273	461
A-n65-k9	cc1	1174.00	1174	0.000	1585.97	41	10660	280	0
	cc4	1174.00	1174	0.000	988.20	41	8104	156	0
	ccX	1174.00	1174	0.000	978.35	41	8104	156	0
	ng1	1174.00	1174	0.000	1304.92	27	9622	249	85
	ng4	1174.00	1174	0.000	1279.39	29	8934	151	41
	ngX	1174.00	1174	0.000	1020.51	25	9556	127	36
A-n69-k9	cc1	1155.54	Inf	Inf	10804.70	183	18757	311	0
	cc4	1154.42	Inf	Inf	10803.90	160	18972	345	0
	ccX	1154.52	Inf	Inf	10822.50	154	18424	375	0
	ng1	1154.86	Inf	Inf	10818.20	158	16298	211	74
	ng4	1153.73	Inf	Inf	10825.50	102	15521	219	670
	ngX	1153.60	Inf	Inf	10870.30	103	16325	227	674
A-n80-k10	cc1	1760.23	Inf	Inf	10942.40	64	20022	324	0
	cc4	1760.68	Inf	Inf	10949.00	53	19488	370	0
	ccX	1760.69	Inf	Inf	10821.30	60	19573	378	0
	ng1	1760.16	Inf	Inf	10810.00	60	19404	317	19
	ng4	1759.56	Inf	Inf	11015.50	32	18616	329	501
	ngX	1759.12	Inf	Inf	10859.30	32	20933	317	498

Instance	Method	LB	UB	Gap	Time	Nodes	Routes	CCs	*ng*CCs
P-n16-k8	cc1	450.00	450	0.000	0.13	9	113	16	0
	cc4	450.00	450	0.000	0.20	9	118	37	0
	ccX	450.00	450	0.000	0.77	9	118	37	0
	ng1	450.00	450	0.000	0.13	9	113	16	0
	ng4	450.00	450	0.000	0.04	1	91	32	1
	ngX	450.00	450	0.000	0.07	1	91	32	1
P-n19-k2	cc1	212.00	212	0.000	35.26	9	4557	7	0
	cc4	212.00	212	0.000	18.33	5	3327	6	0
	ccX	212.00	212	0.000	18.30	5	3327	6	0
	ng1	212.00	212	0.000	35.12	9	4557	7	0
	ng4	212.00	212	0.000	38.17	5	3297	8	5
	ngX	212.00	212	0.000	38.44	5	3297	8	5
P-n20-k2	cc1	216.00	216	0.000	58.26	17	8350	9	0
	cc4	216.00	216	0.000	60.30	17	8294	9	0
	ccX	216.00	216	0.000	60.65	17	8294	9	0
	ng1	216.00	216	0.000	58.82	17	8350	9	0
	ng4	216.00	216	0.000	107.15	21	9728	9	3
	ngX	216.00	216	0.000	100.00	19	9325	9	7
P-n21-k2	cc1	211.00	211	0.000	1.80	1	2777	0	0
	cc4	211.00	211	0.000	1.79	1	2777	0	0
	ccX	211.00	211	0.000	1.88	1	2777	0	0
	ng1	211.00	211	0.000	1.80	1	2777	0	0
	ng4	211.00	211	0.000	1.85	1	2777	0	0
	ngX	211.00	211	0.000	1.82	1	2777	0	0
P-n22-k2	cc1	216.00	216	0.000	28.95	5	5371	4	0
	cc4	216.00	216	0.000	29.16	5	5372	4	0
	ccX	216.00	216	0.000	29.12	5	5372	4	0
	ng1	216.00	216	0.000	29.17	5	5371	4	0
	ng4	216.00	216	0.000	31.36	5	5761	4	1
	ngX	216.00	216	0.000	31.59	5	5761	4	1
P-n22-k8	cc1	590.00	590	0.000	0.05	1	239	2	0
	cc4	590.00	590	0.000	0.05	1	239	46	0
	ccX	590.00	590	0.000	0.09	1	239	46	0
	ng1	590.00	590	0.000	0.06	1	239	2	0
	ng4	590.00	590	0.000	0.07	1	239	46	3
	ngX	590.00	590	0.000	0.18	1	239	46	3
P-n23-k8	cc1	529.00	529	0.000	0.06	1	243	33	0
	cc4	529.00	529	0.000	0.07	1	238	33	0
	ccX	529.00	529	0.000	0.07	1	238	33	0
	ng1	529.00	529	0.000	0.07	1	248	22	5
	ng4	529.00	529	0.000	0.05	1	237	27	10
	ngX	529.00	529	0.000	0.15	1	237	27	10
P-n40-k5	cc1	458.00	458	0.000	87.97	5	3601	65	0
	cc4	458.00	458	0.000	88.84	5	4366	84	0
	ccX	458.00	458	0.000	89.94	5	4366	84	0
	ng1	458.00	458	0.000	106.29	5	3518	61	16
	ng4	458.00	458	0.000	203.78	7	4109	58	299
	ngX	458.00	458	0.000	205.70	7	4109	58	299
P-n45-k5	cc1	510.00	510	0.000	1313.52	38	13118	115	0
	cc4	510.00	510	0.000	1230.04	40	14300	150	0
	ccX	510.00	510	0.000	1249.55	40	14447	150	0
	ng1	510.00	510	0.000	1485.28	37	12587	92	25
	ng4	510.00	510	0.000	1793.47	31	11031	118	477
	ngX	510.00	510	0.000	1801.73	31	11031	118	477
P-n50-k7	cc1	554.00	554	0.000	531.12	25	6295	104	0
	cc4	554.00	554	0.000	509.64	25	6975	136	0
	ccX	554.00	554	0.000	626.48	29	7546	140	0
	ng1	554.00	554	0.000	597.77	25	6744	81	29
	ng4	554.00	554	0.000	541.69	21	5756	93	244
	ngX	554.00	554	0.000	511.23	21	4975	89	245
P-n50-k8	cc1	626.02	Inf	Inf	10813.20	956	20171	436	0
	cc4	625.87	630	0.660	10811.30	885	18613	512	0
	ccX	625.94	Inf	Inf	10801.90	865	18610	535	0
	ng1	626.01	Inf	Inf	10800.10	876	19930	411	41
	ng4	626.16	Inf	Inf	10813.30	842	19204	399	116
	ngX	626.10	629	0.463	10813.20	813	18406	354	132
P-n50-k10	cc1	696.00	696	0.000	577.61	123	4210	99	0
	cc4	696.00	696	0.000	775.70	157	4667	134	0
	ccX	696.00	696	0.000	806.39	157	4667	134	0
	ng1	696.00	696	0.000	288.48	73	3092	97	8
	ng4	696.00	696	0.000	1089.45	158	4449	133	250
	ngX	696.00	696	0.000	953.60	131	4336	113	245
P-n51-k10	cc1	741.00	741	0.000	181.65	37	3163	109	0
	cc4	741.00	741	0.000	282.26	55	3238	158	0
	ccX	741.00	741	0.000	195.22	37	3183	128	0
	ng1	741.00	741	0.000	223.46	37	2949	105	8
	ng4	741.00	741	0.000	210.04	31	3230	129	27
	ngX	741.00	741	0.000	233.52	41	3248	128	27
P-n55-k7	cc1	565.29	Inf	Inf	10836.70	228	23661	207	0
	cc4	565.14	Inf	Inf	10821.10	211	22065	215	0
	ccX	565.23	Inf	Inf	10811.30	224	23023	226	0
	ng1	566.00	Inf	Inf	10876.30	217	23745	165	44
	ng4	565.56	Inf	Inf	10846.20	190	21146	179	82
	ngX	565.50	Inf	Inf	10857.40	193	21736	170	65
P-n55-k8	cc1	576.00	576	0.000	1261.99	31	9049	234	0
	cc4	576.00	576	0.000	1880.59	45	9180	273	0
	ccX	576.00	576	0.000	2248.38	57	9324	281	0
	ng1	576.00	576	0.000	729.57	17	6151	174	73
	ng4	576.00	576	0.000	1715.40	35	7451	174	495
	ngX	576.00	576	0.000	1726.61	35	7451	174	495
P-n55-k10	cc1	690.99	Inf	Inf	10805.10	948	14192	374	0
	cc4	690.22	Inf	Inf	10808.70	944	13728	419	0
	ccX	690.20	Inf	Inf	10813.70	907	13688	444	0
	ng1	690.41	Inf	Inf	10804.00	873	13897	358	29
	ng4	691.53	Inf	Inf	10800.20	837	14426	371	81
	ngX	691.33	Inf	Inf	10815.20	761	13486	365	98
P-n55-k15	cc1	941.00	941	0.000	59.43	37	1392	84	0
	cc4	941.00	941	0.000	39.18	21	1445	175	0
	ccX	941.00	941	0.000	44.37	21	1445	175	0
	ng1	941.00	941	0.000	51.52	29	1360	70	4
	ng4	941.00	941	0.000	33.04	15	1364	85	22
	ngX	941.00	941	0.000	37.92	15	1364	85	22
P-n60-k10	cc1	744.00	744	0.000	2253.97	145	6697	154	0
	cc4	744.00	744	0.000	1396.09	85	5742	134	0
	ccX	744.00	744	0.000	1743.00	129	6225	154	0
	ng1	744.00	744	0.000	650.65	51	4210	97	10
	ng4	744.00	744	0.000	1684.65	79	5718	135	380
	ngX	744.00	744	0.000	1721.64	91	5429	108	380
P-n60-k15	cc1	968.00	968	0.000	207.93	55	2575	117	0
	cc4	968.00	968	0.000	243.50	59	2668	139	0
	ccX	968.00	968	0.000	218.64	49	2502	137	0
	ng1	968.00	968	0.000	197.53	49	2544	118	3
	ng4	968.00	968	0.000	237.70	53	2551	133	17
	ngX	968.00	968	0.000	298.74	61	2764	144	26
P-n65-k10	cc1	792.00	792	0.000	1917.97	66	6600	145	0
	cc4	792.00	792	0.000	1183.91	53	6402	144	0
	ccX	792.00	792	0.000	1145.85	53	6398	145	0
	ng1	792.00	792	0.000	1530.51	48	5788	107	38
	ng4	792.00	792	0.000	4707.00	113	9130	157	365
	ngX	792.00	792	0.000	2212.93	57	7097	118	364
P-n70-k10	cc1	819.80	Inf	Inf	10825.00	226	12625	238	0
	cc4	819.61	Inf	Inf	10817.40	220	12522	290	0
	ccX	819.62	Inf	Inf	10828.50	209	11942	271	0
	ng1	819.81	Inf	Inf	10859.40	206	11373	214	9
	ng4	820.02	Inf	Inf	10805.20	197	11793	292	30
	ngX	819.69	Inf	Inf	10805.00	199	12462	263	29
P-n76-k4	cc1	586.32	Inf	Inf	11699.20	2	28220	4	0
	cc4	586.35	Inf	Inf	15208.30	3	28532	5	0
	ccX	586.35	Inf	Inf	15007.80	3	28532	5	0
	ng1	586.32	Inf	Inf	11847.50	2	28220	4	0
	ng4	586.32	Inf	Inf	10930.50	2	27459	5	1
	ngX	586.35	Inf	Inf	14667.70	3	28422	5	1
P-n76-k5	cc1	615.63	Inf	Inf	11993.20	6	23836	17	0
	cc4	615.74	Inf	Inf	10809.00	7	21963	15	0
	ccX	615.74	Inf	Inf	10846.30	7	21963	15	0
	ng1	615.63	Inf	Inf	11973.20	6	23946	17	2
	ng4	615.76	Inf	Inf	12650.90	8	24331	17	2
	ngX	615.76	Inf	Inf	10844.20	7	22799	15	2