Erasmus University Rotterdam

Erasmus School of Economics

Master's Thesis Econometrics and Management Science

# $ng$-Memory Based Capacity Cuts

A new Family of Robust Valid Inequalities for the Capacitated Vehicle Routing Problem with Time Windows using $ng$-Route Relaxation

*Ymro Nils Hoogendoorn (413127)*

Supervisor: MSc Kevin Dalmeijer
Second Assessor: dr. Remy Spliet

August 17, 2018

**Abstract**

This thesis introduces a new family of valid inequalities for the capacitated vehicle routing problem with time windows, the $ng$-memory based capacity cuts. These inequalities are a lifting of the ordinary capacity cuts and can be included robustly into a branch-price-and-cut algorithm if one uses the $ng$-route relaxation. This thesis evaluates the performance of these inequalities by solving the root node relaxation and by solving the full instance using branch-price-and-cut of several Augerat benchmark instances. Furthermore, four separate separation methods are devised to separate these new inequalities, which are all based on the CVRPSEP package for separating ordinary capacity cuts. Two of these methods guarantee that all violated $ng$-capacity cuts are found, provided all ordinary capacity cuts can be separated. In the root node relaxation, $ng$-memory based capacity cuts lifted the lower bound more than ordinary capacity cuts. For some instances, we even obtained an integer solution with the $ng$-cuts, whereas we did not with the capacity cuts. The fourth separation algorithm, which reduces arc flow with a factor of $\kappa$ to find more violated $ng$-capacity cuts, achieves the highest lower bound in almost all of the instances. Furthermore, no additional violated cuts were found for $\kappa > 4$. In the branch-price-and-cut setting, we found that including $ng$-capacity cuts reduces the number of nodes needed in the branching tree drastically. However, no real time savings were achieved due to the longer solving time per node for the $ng$-capacity cuts. All in all, including $ng$-capacity cuts gives us higher lower bounds and reduces the number of nodes in a branching tree.

# Contents

# 1 Introduction

The vehicle routing problem (VRP) is a class of problems that concerns itself with finding the least-cost way to serve multiple locations with one or more vehicles. There may be side constraints the vehicles have to obey. This problem was first introduced in Dantzig and Ramser (1959) as an extension of the travelling salesman problem (TSP), in which we can only use one vehicle to serve all customers and there are no side constraints. The VRP and its variants are encountered frequently in logistics. A straightforward example is the distribution of goods, for instance grocery distributions (Carter, Farvolden, Laporte, & Xu, 1996) and raw materials such as oil or gas (Bell et al., 1983). Due to the growing volume of these industries, it is now more important than ever to solve such problems to optimality.

The capacitated vehicle routing problem (CVRP) is one of the most popular members of the VRP. In the CVRP, the fleet of vehicles is identical and stationed at a central depot. The customers all demand a different amount of produce. However, as the vehicles have a capacity limit, one vehicle cannot serve all customers. Furthermore, deliveries cannot be split up, so that a customer is always served by exactly one vehicle. Another popular member is the vehicle routing problem with time windows (VRPTW), which is a generalization of the CVRP. In this variant, the customers have specific time windows in which they must be served. Both the CVRP and VRPTW are $\mathcal{NP}$-hard (Lenstra & Kan, 1981).

Prevalent algorithms used to solve CVRP and VRPTW instances can be divided into two main categories: inexact and exact methods. The exact methods guarantee that the found solution is optimal, but usually take much longer to find that solution. Inexact methods are usually faster but do not guarantee an optimal solution. In this thesis, we focus on exact methods.

One of the first exact methods to solve the VRP is given in Balinski and Quandt (1964), in which a set partitioning formulation is used with integer programming. This method actually enumerates all feasible routes and can thus only be used if the number of customers is small. Indeed, Balinski and Quandt (1964) only solved instances with up to 15 customers. This approach can also be used to solve the CVRP and VRPTW, as one can

model the extra route restrictions in the set of feasible routes. Rao and Zionts (1968) were one of the first to consider column generation for the VRP, in which one does not enumerate the routes, but rather generates them dynamically. They use this method to solve larger VRP instances to optimality.

Fisher and Jaikumar (1981) developed the three-index flow formulation for the CVRP. This mathematical program is completely different from the set partitioning formulation in that it models the arcs over which the vehicles travel directly, but it contains an exponential number of constraints. Fisher and Jaikumar (1981) do not solve this formulation exactly. Laporte, Nobert, and Desrochers (1985) use a variant of the three-index flow formulation: the two-index flow formulation, in which one does not have separate flow variables for vehicles. They use this formulation, together with branch-and-cut procedure, to obtain optimal solutions for the symmetric CVRP. In the branch-and-cut procedure, one solves the linear program with just a limited number of constraints. If the solution is fractional or violates some constraint, one branches on the fractional variable or adds the violated constraint dynamically. With this technique, CVRP instances with up to 50 customers are solved to optimality.

Desrochers, Desrosiers, and Solomon (1992) expanded on the set partitioning formulation combined with column generation to solve the VRPTW to optimality. In this approach, new routes are generated by solving a shortest path problem with resource constraints with 2-cycle elimination. This means that routes are allowed to visit customers multiple times, but cycles of length 2 are forbidden. Integer solutions are obtained by branching on arc flows, which results in a branch-and-price algorithm. With this approach, VRPTW instances with up to 100 customers are solved to optimality.

Augerat (1995) and Kohl (1995) both introduced the concepts of *valid inequalities* for the VRPTW and CVRP. Valid inequalities are inequalities that cut off a region of the linear program space, but leave the integer portion intact. These inequalities were used with a branch-and-cut procedure to reduce the number of branches needed to find an optimal solution. Further elaborating on valid inequalities, Augerat et al. (1998) also used a branch-and-cut procedure on the 2-index flow formulation to solve CVRP instances

to optimality. With this technique, an instance with 134 customers was solved to optimality, which was the largest CVRP instance solved to optimality to this date. Bard, Kontoravdis, and Yu (2002) also employed a similar brach-and-cut procedure with valid inequalities, but used it to solve VRPTW instances. They also executed heuristics to find upper bounds for the optimal objective, to cut off part of the solution space. This allowed them to solve VRPTW instances with up to 100 customers to optimality.

Back to the column generation side, it was already known that a branch-and-price algorithm could be combined with the dynamic generation of valid inequalities to yield a branch-price-and-cut algorithm. The first branch-price-and-cut algorithm was introduced in Nemhauser and Park (1991) and was used to solve the graph colouring problem. For branch-price-and-cut algorithms, de Aragao and Uchoa (2003) introduced the distinction between *robust* and *non-robust* valid inequalities. Valid inequalities are called robust if they do not complicate the generation of new routes (the pricing problem). Non-robust valid inequalities do complicate the pricing problem, which means one must be very careful about how many non-robust valid inequalities to include. Branch-price-and-cut's potential was quickly noticed and used for VRPTW problems. For instance, Kohl, Desrosiers, Madsen, Solomon, and Soumis (1999) used column generation on the set partitioning formulation, together with 2-cycle elimination and 2-path cuts (which are robust), to solve VRPTW instances. Several unsolved 50- and 100-customer instances could be solved to optimality. One of the first instances branch-price-and-cut was used on the CVRP, was done by Fukasawa et al. (2006). In this paper, they solely use robust valid inequalities, combined with heuristic pricing in early iterations to solve CVRP instances to optimality with up to 120 customers. While this paper is a breakthrough for solving CVRP instances with the branch-price-and-cut technique, this technique is still not as effective as using branch-and-cut for the CVRP. This is why branch-price and cut was not used very often on CVRP instances.

Feillet et al. (2004) used a column generation algorithm to solve the VRPTW. They actually solved an elementary shortest path problem with resource constraints instead of the more common relaxed version with 2-cycle elimination. They found that the linear programming gap can be reduced using this technique. As an extension of the 2-cycle

elimination for the shortest path problem with resource constraints, Irnich and Villeneuve (2006) developed the $k$-cycle elimination, with $k \geq 2$. They found that using $k = 3$ or 4 really benefits some VRPTW instances, and solved some unsolved 50- and 100-customer instances to optimality. However, a different and more promising relaxation to the elementary shortest path problem with resource constraints is the $ng$-route relaxation, introduced in Baldacci, Mingozzi, and Roberti (2011). This relaxation is completely different from the established $k$-cycle elimination, but it proved very successfully for solving large CVRP and VRPTW instances.

Røpke (2012) also uses a robust branch-price-and-cut algorithm, but with the $ng$-route relaxation instead of $k$-cycle elimination. With this, they solved a 150-customer CVRP instance in about 5 days. Finally, Pecin, Pessoa, Poggi, and Uchoa (2017) used branch-price-and-cut with $ng$-route relaxations and both robust and non-robust valid inequalities to solve CVRP instances to optimality faster than any other algorithm to date. They limit the number of non-robust cuts in order to keep the pricing problem tractable.

As one can see from the above overview, branch-price-and-cut algorithms are recently among the best-performing algorithms for solving CVRP and VRPTW instances. Robust valid inequalities play an important role in such algorithms. The main contribution of this thesis is the introduction of a new type of robust valid inequalities for the VRPTW and CVRP using $ng$-route relaxation, the $ng$-capacity cuts (NGCCs). The main questions of this thesis are whether NGCCs increase the LP bound and/or save runtime on the CVRP and VRPTW compared to using no cuts and CCs, and which separation technique is most useful for separating NGCCs. Furthermore, we test whether NGCCs speed up finding an integer solution to the CVRP and VRPTW compared to using CCs in a branch-price-and-cut setting.

The outline of this thesis is as follows. First, in Section 2 we introduce the mathematical notation for the CVRP and VRPTW. We also introduce the set partitioning formulation, which forms the basis of the branch-price-and-cut algorithm. Then, Section 3 discusses the branch-price-and-cut algorithm in detail. Section 4 introduces the NGCCs and describes several separation algorithms. Next, Section 5 describes the experimental setup

and instances used, while Section 6 discusses the results of the experiments. Finally, Section 7 presents the final conclusions of this thesis.

# 2 Problem Definition

## 2.1 Problem Description

We model the CVRP and VRPTW by setting up a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. We denote $\mathcal{V} = \{1, \ldots, n\}$ as the set of customers, 0 as the starting depot and $n+1$ as the ending depot, giving us $\mathcal{N} = \mathcal{V} \cup \{0, n+1\}$. Note that the starting and ending depot can also refer to one, central, depot. The distinction is made for mathematical convenience. We denote the set of arcs with $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$. Travelling to the starting depot and from the ending depot is prohibited. With each arc $(i, j) \in \mathcal{A}$, we associate a travel cost $c_{ij} \geq 0$ and travel time $t_{ij} \geq 0$. Furthermore, each customer $i \in \mathcal{V}$ has a demand $q_i > 0$ and time window $[e_i, l_i]$. Here, $e_i$ and $l_i \geq e_i$ are the earliest and latest service times of customer $i$, respectively. Without loss of generality we set $e_0 = e_{n+1} = 0 = q_0 = q_{n+1}$ and assume that $e_i \geq 0$ for all $i \in \mathcal{N}$. $l_0 = l_{n+1}$ can be used to limit the overall length of routes. We denote the total capacity of a vehicle by $Q > 0$. We assume that $q_i \leq Q$ for all $i \in \mathcal{N}$.

For both the VRPTW and CVRP, a feasible solution consists of a collection of feasible routes, that together serve each customer. Every route $r$ is executed by one vehicle and can be defined as a path $r = (0, i_1, i_2, \ldots, i_{|r|}, n+1)$ through $\mathcal{G}$, with $i_1, i_2, \ldots, i_{|r|} \in \mathcal{V}$ and $|r| \geq 1$. We say $|r|$ is, by abuse of notation, equal to the number of customers $r$ visits. Here, the vehicle starts at the depot, 0, visits customers $(i_1, i_2, \ldots, i_{|r|})$ in the specified order and ends at the depot $n+1$. A route is feasible for the CVRP if the total demand on the route, $\sum_{k=1}^{|r|} q_{i_k}$, does not exceed the capacity of a vehicle, $Q$. Furthermore, the route must be elementary, which means no customer is visited more than once. For the VRPTW, we have the additional constraint that every customer can be served within their respective time window. Note that vehicles are allowed to wait.

## 2.2 The Set Partitioning Formulation

If we assume the total set of feasible routes, $\mathcal{R}$, is available, we can model the CVRP or the VRPTW as a set partitioning problem, as originally proposed by Balinski and Quandt (1964). The binary variable $x_r$ denotes whether we choose route $r \in \mathcal{R}$ in our solution. Parameter $c_r$ gives the costs of using route $r \in \mathcal{R}$. These costs can be calculated by summing $c_{ij}$ for all arcs $(i, j)$ visited by $r$. To track which customer is served in some route $r \in \mathcal{R}$, we introduce $a_r^i$, which equals the number of times route $r$ visits customer $i \in \mathcal{V}$. Note that, for now, this parameter can be regarded as binary. The set partitioning problem, is given by SPP.

$$\text{(SPP)} \qquad \min \quad \sum_{r \in \mathcal{R}} c_r x_r, \tag{1a}$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}} a_r^i x_r = 1 \qquad \forall i \in \mathcal{V}, \tag{1b}$$

$$x_r \in \mathbb{B} \qquad \forall r \in \mathcal{R}. \tag{1c}$$

As the size of $\mathcal{R}$ is exponential in the number of customers $n$, directly solving SPP is intractable, even for moderate $n$. Instead, we will use a branch-price-and-cut algorithm, which will be discussed in the next section.

# 3 Branch-Price-and-Cut

This section will explain the basics of a branch-price-and-cut algorithm, which has been used to successfully solve CVRP (Fukasawa et al., 2006; Pecin et al., 2017; Røpke, 2012) and VRPTW (Kohl et al., 1999) instances. Section 3.1 describes the technique of column generation, while Section 3.2 elaborates on the pricing problem of SPP used to generate new columns. The mathematical definition of the *ng*-route relaxation is given in Section 3.3. We discuss capacity cuts and their strengthened form in Section 3.4, the algorithm used to solve the LP relaxation of SPP to optimality in Section 3.5 and techniques used to obtain the final integer solution in Section 3.6.

## 3.1 Column Generation

With column generation, we can solve the LP relaxation of SPP to optimality, without enumerating the entire set $\mathcal{R}$ explicitly. Instead of the full set $\mathcal{R}$, one uses a tractable subset $\mathcal{R}' \subseteq \mathcal{R}$ of routes, which gives us the restricted master problem (RMP) (Desrosiers & Lübbecke, 2005).

Normally, in the simplex algorithm, one searches for a route $r \in \mathcal{R}$ whose reduced costs

$$\rho_r = c_r - \sum_{i \in \mathcal{V}} \lambda_i a_r^i,$$

are negative. Here $\lambda_i$, $i \in \mathcal{V}$ are the dual variables of (1b). If we then let this variable enter the basis, the objective decreases. Whenever there is no such variable with negative reduced costs, the current LP solution is optimal.

However, with column generation the entire set $\mathcal{R}$ is not explicitly available. Instead, searching for a route $r \in \mathcal{R}$ with negative reduced costs is formulated as an optimization problem (Desrosiers & Lübbecke, 2005):

$$\rho^* = \min_{r \in \mathcal{R}} \{\rho_r\}. \tag{2}$$

If $\rho^* < 0$, then we add such a route or several routes $r$ with $\rho_r < 0$ to $\mathcal{R}'$. However, if $\rho^* \geq 0$, then the solution to the RMP also solves the SPP to LP optimality (Desrosiers & Lübbecke, 2005). The optimization problem (2) is known as the pricing problem. We will discuss the pricing problem and how to solve it in detail in the next section.

## 3.2 The Pricing Problem

This section describes the pricing problem of the CVRP and VRPTW in detail. Firstly, we will note that the pricing problem (2) is an elementary shortest path problem with resource constraints (ESPPRC) (Kallehauge, Larsen, & Madsen, 2006). The costs of travelling over an arc $(i, j) \in \mathcal{A}$ equal $\bar{c}_{ij} = c_{ij} - \lambda_j$, where we define $\lambda_0 = \lambda_{n+1} = 0$. Then, $\rho_r$ of some route $r \in \mathcal{R}$ equals the sum of $\bar{c}_{ij}$ for every arc $(i, j)$ the route visits. The resources model the restrictions the route has to follow and keep track of the costs of the

route. For instance, we need a resource for the demand satisfied by the route, so we can prevent it from exceeding $Q$. The VRPTW also needs a time resource for the customer's time windows. Furthermore, as routes must be elementary, we can use $n$ resources to track which customer has already been visited.

Before we give an algorithm to solve the (E)SPPRC, we will first give a general description of the problem. After that, we will apply the general description to the VRPTW and CVRP.

### 3.2.1 General Description of the Shortest Path Problem with Resource Constraints

The SPPRC is defined on a digraph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, with $\mathcal{N}$ and $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ the nodes and arcs of the graph respectively. Note that this graph $\mathcal{G}$ is the same as the graph of the CVRP or the VRPTW. Just as the name of the problem suggests, not any path through $\mathcal{G}$ is deemed feasible. It has to adhere to the resource constraints, which are usually modelled with resource consumptions and resource intervals.

As the SPPRC has an arbitrary number of resources $R \geq 1$ (Irnich & Desaulniers, 2005), we first introduce some concepts concerning resource vectors. A vector $\mathbf{u} = [u^1, u^2, \ldots, u^R]^\top \in \mathbb{R}^R$ is called a resource vector, where $\mathbf{x}^\top$ denotes the transpose of the vector $\mathbf{x}$. We say for some resource vectors $\mathbf{u}$ and $\mathbf{w}$ that $\mathbf{u} \leq \mathbf{w}$ if and only if $u^s \leq w^s$, for all $s \in \{1, \ldots, R\}$. Furthermore, we define the resource interval $[\mathbf{a}, \mathbf{b}]$ as $\{\mathbf{u} \in \mathbb{R}^R : \mathbf{a} \leq \mathbf{u} \leq \mathbf{b}\}$. Note that $\mathbf{a}$ and $\mathbf{b}$ must also be resource vectors by construction.

In the SPPRC, only certain levels of resources are allowed in a node $i \in \mathcal{N}$. We model this by introducing a specific resource interval $[\mathbf{a}_i, \mathbf{b}_i]$ for every node. Whenever we travel over arcs, our resources change. Irnich and Desaulniers (2005) capture this via the resource extension functions (REFs). A REF $\mathbf{f}_{ij} : \mathbb{R}^R \to \mathbb{R}^R$ is a vector function that models how a resource vector changes over an arbitrary arc $(i, j) \in \mathcal{A}$. Note that, using these REFs, resource consumptions can be dependent on other resources and can be non-linear. We denote the $s$th component of $\mathbf{f}_{ij}$ as $f_{ij}^s : \mathbb{R}^R \to \mathbb{R}$, with $s \in \{1, \ldots, R\}$, such that $\mathbf{f}_{ij} = [f_{ij}^1, \ldots, f_{ij}^R]^\top$. As the CVRP and VRPTW can be modelled with non-decreasing

REFs, we assume that the REFs are non-decreasing. This makes the SPPRC easier to solve (Irnich & Desaulniers, 2005). That is, $\mathbf{u} \leq \mathbf{w}$ implies that $\mathbf{f}_{ij}(\mathbf{u}) \leq \mathbf{f}_{ij}(\mathbf{w})$ for all $(i,j) \in \mathcal{A}$.

The objective of the SPPRC is to find a feasible path that minimizes the first resource, which we will call the "cost" resource. We will now define paths and their feasibility. A path $P$ through $\mathcal{G}$ is defined as a $(p+1)$-tuple $(i_0 = 0, i_1, \ldots, i_p)$, with $i_k \in \mathcal{N}$ for all $k \in \{0, \ldots, p\}$ and $p \geq 0$. We say $p$ is the length of $P$. We implicitly assume that $(i_k, i_{k+1}) \in \mathcal{A}$, for all $k \in \{0, \ldots, p-1\}$. Path $P$ is resource-feasible if and only if there exists resource vectors $\mathbf{u}_k$ for all $k \in \{0, \ldots, p\}$ such that $\mathbf{u}_k \in [\mathbf{a}_{i_k}, \mathbf{b}_{i_k}]$ for all $k \in \{0, \ldots, p\}$ and $\mathbf{f}_{i_k, i_{k+1}}(\mathbf{u}_k) \leq \mathbf{u}_{k+1}$ for all $k \in \{0, \ldots, p-1\}$ (Irnich & Desaulniers, 2005). This definition of feasibility implies that there are many different possible choices for these resource vectors $\mathbf{u}_0, \ldots, \mathbf{u}_p$ if a path $P$ is feasible. Fortunately, if all the REFs are non-decreasing, we associate a unique "minimal" collection of resource vectors $(\mathbf{u}_0(P), \ldots, \mathbf{u}_p(P))$ with path $P$ which are all smaller than or equal to all other feasible resource vectors. As costs are also a resource, other feasible collections of resource vectors for $P$ are not relevant, as their costs could be higher. Irnich and Desaulniers (2005) state the formulae for these minimal resource vectors recursively as

$$\mathbf{u}_k(P) = \begin{cases} \mathbf{a}_{i_0} & \text{, if } k = 0 \\ \max\left\{\mathbf{a}_{i_k}, \mathbf{f}_{i_{k-1}, i_k}(\mathbf{u}_{k-1}(P))\right\} & \text{, if } k \in \{1, \ldots, p\} \end{cases}. \tag{3}$$

Here, the max function is defined component-wise. Note that path $P$ is resource-feasible if and only if $\mathbf{u}_k(P) \leq \mathbf{b}_{i_k}$ for all $k \in \{0, \ldots, p\}$.

### 3.2.2 The SPPRC for the CVRP and VRPTW

The CVRP can be modelled with $R = n + 2$ resources. The first is the cost resource $R_C$. Its resource windows are positive unbounded, that is, $a_i^{R_C} = 0$ and $b_i^{R_C} = +\infty$, for all $i \in \mathcal{N}$. The REFs are defined as $f_{ij}^{R_C}(\mathbf{u}) = u^{R_C} + \bar{c}_{ij}$ for all $(i,j) \in \mathcal{A}$. The second resource is the capacity resource $R_Q$. Its resource windows are bounded by $Q$, the capacity of one vehicle. That is, $a_i^{R_Q} = 0$ and $b_i^{R_Q} = Q$, for all $i \in \mathcal{N}$. The REFs simply add the demand of the newly visited customer to the capacity, that is $f_{ij}^{R_Q}(\mathbf{u}) = u^{R_Q} + q_j$ for all $(i,j) \in \mathcal{A}$. Finally, as it is forbidden for a vehicle to visit the same customer twice in a

route, we introduce $n$ additional resources $R_E^1, \ldots, R_E^n$ to track which customers we have already visited in the route. This was introduced in Beasley and Christofides (1989) and is called the elementary route restriction. The windows are $a_i^{R_E^j} = 0$ and $b_i^{R_E^j} = 1$, for all $i \in \mathcal{N},\ j \in \mathcal{V}$. The REFs are

$$f_{ij}^{R_E^v}(\mathbf{u}) = \begin{cases} u^{R_E^v} & \text{, if } j \neq v \\ u^{R_E^v} + 1 & \text{, if } j = v \end{cases} \qquad \forall (i,j) \in \mathcal{A},\ v \in \mathcal{V}.$$

Feillet et al. (2004) actually alter the above resources, by also setting $R_E^i$ to 1 if the partial path cannot reach that customer anymore. For instance, when visiting a customer is impossible time-wise or when its demand exceeds the remaining capacity of the vehicle. This makes solving the SPPRC easier. As this technique is incompatible with the duals of the *ng*-memory capacity cuts (see Section 4.4), we do not include this technique.

For the VRPTW, we also model the time resource $R_T$. Its resource windows are exactly the time windows defined earlier: $a_i^{R_T} = e_i$ and $b_i^{R_T} = l_i$, for all $i \in \mathcal{N}$. The REFs are, similar to the cost resource, $f_{ij}^{R_T}(\mathbf{u}) = u^{R_T} + t_{ij}$, for all $(i,j) \in \mathcal{A}$.

### 3.2.3 Solving the SPPRC

Usually, the SPPRC is solved by means of a labelling algorithm (Irnich & Desaulniers, 2005). In such an algorithm, labels are used to represent feasible (partial) paths through the graph, which are extended by investigating all possible feasible extensions of them. All partial paths start at node 0, the starting depot. Paths are regarded as completed as soon as they have reached node $n+1$, the ending depot.

We can define a label $L$ as a tuple $L = (i, \mathbf{u}, L_{prev})$. That is, a label is represented by a current node $i \in \mathcal{N}$, a resource vector $\mathbf{u}$ and a predecessor label $L_{prev}$. One could trace back the predecessor labels of a label to construct the path $P = (i_0 = 0, i_1, \ldots, i_p = i)$ implied by the label $L$. Note that the resource vector stored in the label should equal the minimal resource vector $\mathbf{u}_p(P)$ of path $P$. The initial label $L_0$ is the label corresponding to the empty path $P = (0)$ and equals $L_0 = (0,\ \mathbf{a}_0,\ NO)$. Here, $\mathbf{a}_0$ is a resource vector with its components equal to the resource lower bounds of the depot 0. The symbol $NO$ indicates the lack of a predecessor label. We denote with $i(L)$ and $\mathbf{u}(L)$ the current node

and resource vector of label $L$, respectively.

The way we extend a label is rather straightforward. Given a label $L = (i, \mathbf{u}, L_{prev})$, we can extend it via arc $(i, j) \in \mathcal{A}$ into label $L'$, which equals

$$L' = (j, \max\{\mathbf{a}_j, \mathbf{f}_{ij}(\mathbf{u})\}, L). \tag{4}$$

The updating of the resource vector is akin to recursion scheme (3). Consequently, $L'$ is resource-feasible if and only if $\max\{\mathbf{a}_j, \mathbf{f}_{ij}(\mathbf{u})\} \leq \mathbf{b}_j$.

If we start with $L_0$ and keep extending labels until all of them are completed, we just enumerate all feasible partial paths. We therefore use dominance rules to exclude some undesired labels. We say a label $L_1$ dominates label $L_2$ if the following conditions are met (Irnich & Desaulniers, 2005):

1. $L_1$ and $L_2$ have the same current node.
2. Every feasible extension of label $L_2$ is also feasible for $L_1$.
3. For every feasible extension of $L_2$, possibly of multiple arcs, the costs of the extended $L_1$ are lower than or equal to the costs of the extended $L_2$.

These dominance rules are not practical to verify, as one needs to consider all feasible extensions of a label, of which there are an exponential number. Fortunately, because our REFs are non-decreasing, these conditions can be restated neatly (Irnich & Desaulniers, 2005). We say $L_1$ dominates $L_2$, denoted by $L_1 \preceq L_2$, if and only if equation (5) holds.

$$L_1 \preceq L_2 \iff i(L_1) = i(L_2),\ \mathbf{u}(L_1) \leq \mathbf{u}(L_2) \tag{5}$$

We also enforce that at least one inequality is strict, to prevent equal labels from getting eliminated. Note that, because costs are included in the resource vector $\mathbf{u}$, $L_1 \preceq L_2$ implies that the costs of $L_1$ are lower than or equal to the costs of $L_2$. As dominated paths are never part of an optimal solution, we can remove those during the labelling process. The resulting labelling algorithm can be found in Algorithm 1, which is adapted from Irnich and Desaulniers (2005).

**Algorithm 1:** Labelling Algorithm for SPPRC.

1 $\mathcal{L}_U \leftarrow \{L_0\}$, the set of unprocessed labels;
2 $\mathcal{L}_P \leftarrow \emptyset$, the set of processed labels;
3 **while** $\mathcal{L}_U \neq \emptyset$ **do**
4  $L \leftarrow$ some label in $\mathcal{L}_U$;
5  **foreach** $j : (i(L), j) \in \mathcal{A}$ **do**
6  Extend $L$ into $L'$ over arc $(i(L), j)$ via equation (4);
7  **if** $L'$ resource-feasible **and** $\nexists L'' \in \mathcal{L}_U \cup \mathcal{L}_P : L'' \preceq L'$ **then**
8  $\mathcal{L}_U \leftarrow \mathcal{L}_U \cup \{L'\}$;
9  **foreach** $L'' \in \mathcal{L}_U : L' \preceq L''$ **do**
10  $\mathcal{L}_U \leftarrow \mathcal{L}_U \setminus \{L''\}$;
11  **end**
12  **end**
13  **end**
14  $\mathcal{L}_U \leftarrow \mathcal{L}_U \setminus \{L\}$;
15  $\mathcal{L}_P \leftarrow \mathcal{L}_P \cup \{L\}$;
16 **end**
17 Return one or several labels $L \in \mathcal{L}_P$ with $i(L) = n + 1$ and $u^{R_C}(L) < 0$;

There are many techniques for choosing an unprocessed label (line 4). As preliminary experiments showed that extending the shortest labels usually dominate more labels, we choose the first created unprocessed label. For line 17, we return only one label with the most negative reduced costs. Note that lines 7 and 9 suggest that every created label is compared to every other label. This can be time-consuming however. Instead, as is suggested in Martinelli, Pecin, and Poggi (2014), one can limit the number of dominance checks to a certain subset of labels. For instance, one could only check for dominance in labels with the same capacity resource $R_Q$. We do not employ this technique in this thesis, though.

### 3.2.4 Pricing States

While label domination can greatly speed up the labelling process, it may still take a long time to generate enough columns so that the CVRP or the VRPTW is solved to LP-

optimality. There exist other techniques to speed up this process. A powerful strategy is to accelerate the labelling process early in the column generation. The idea is based on the fact that the duals obtained by solving the RMP are not representative for the optimal duals of the SPP, as indicated by the unstable behaviour of column generation by Kallehauge et al. (2006), when the number of generated columns is small. In other words, the optimal routes generated in early pricing problems are probably not used in any final solution.

So, it seems obvious we can save time by not trying to find every path with negative reduced costs, as Algorithm 1 does, but just any path with negative reduced costs. We accomplish this by employing heuristic dynamic programming, as discussed in Desaulniers, Lessard, and Hadjar (2008). These authors use two methods to accomplish this: aggressive dominance and arc elimination.

With aggressive dominance, one adjusts the label domination rule (5) so that more dominance takes place. For instance, one could ignore all $n$ different elementarity (or $ng$-resources, see Section 3.3) in dominance. Desaulniers et al. (2008) adjust the dominance rule as follows: they check dominance on the cost ($R_C$), capacity ($R_Q$), time ($R_T$) and $R_{max}$ of the "customer" resources ($R^i_E$ or $R^i_{NG}$, $i \in \mathcal{V}$). When $R_{max} = n$, the dominance rule is exact and no heuristic dynamic programming takes place. When $R_{max} = 0$, all elementarity or $ng$-resources are ignored in the dominance rule.

Given the number of customer resources, $R_{max}$, to use in the dominance check, Desaulniers et al. (2008) determine which $R_{max}$ customers to use according to the following rule. They compute the difference of the customer duals $\lambda_i$ between the current and previous iteration, for all $i \in \mathcal{V}$, and select the $R_{max}$ customers for which this difference is the highest.

Arc elimination reduces the number of arcs in the graph $\mathcal{G}$. Fewer arcs mean fewer partial routes can be constructed, thus saving time in the labelling algorithm. Desaulniers et al. (2008) limit the number of outgoing and incoming arcs of every customer $i \in \mathcal{V}$ by $A_{out}$ and $A_{in}$, respectively. To select which arcs should be included, they choose arcs with least travel costs $\bar{c}_{ij} = c_{ij} - \lambda_j$. Note that arcs going to and from the depot are always included.

In this thesis, we only limit the number of outgoing arcs, as this limits the number of arcs in the graph enough. That is, we always set $A_{in} = n$. Whenever we include capacity cuts (see Section 3.4), we also incorporate those duals into the reduced arc costs $\bar{c}_{ij}$. If we include $ng$-capacity cuts (see Section 4), we treat those duals as CC duals. This thus gives us an approximation of the arc costs.

In early iterations of the column generation algorithm, when a small number of columns have been added, we want our pricing problem to be fast and inexact. Thus, we want low $R_{max}$ and $A_{out}$. In later iterations, we desire to price exactly, thus with $R_{max} = A_{out} = n$. To achieve this, we use *pricing states*. A pricing state dictates the values of $R_{max}$ and $A_{out}$ and thus the inexactness of the labelling algorithm. When the labelling algorithm fails to find a route with negative reduced costs, the pricing state is increased and the algorithm is run again. If the pricing state equals the highest (exact) state, we know for sure no routes with negative reduced costs exist and the LP is solved to optimality. If a route is found with negative reduced costs, the pricing state is reset to the lowest (most inexact) state. An overview of the used pricing states, determined by preliminary experiments, is given in Table 1.

Table 1: List of pricing states used.

| Pricing State | $R_{max}$ | $A_{out}$ |
|---|---|---|
| 0 | 0 | 5 |
| 1 | 0 | $\lfloor \frac{n}{3} \rfloor$ |
| 2 | $\lfloor \frac{n}{3} \rfloor$ | $\lfloor \frac{2n}{3} \rfloor$ |
| 3 | $\lfloor \frac{2n}{3} \rfloor$ | $n$ |
| 4 | $n$ | $n$ |

## 3.3 $ng$-Route Relaxation

The previous section already mentioned that the PP of the CVRP and VRPTW is the ESPPRC. The ESPPRC is $\mathcal{NP}$-hard in the strong sense (Dror, 1994), so it may be worthwhile to relax the problem. Several successful branch-and-price or branch-price-and-cut applications from the past indeed relax the elementarity of the pricing problem. Notable examples include the 2-cycle elimination (Desrochers et al., 1992), the $k$-cycle elimination (Irnich & Villeneuve, 2006) and the $ng$-route relaxation (Baldacci et al., 2011). Of these,

$ng$-route relaxation performs the most promising. This relaxation allows for certain types of non-elementary routes, namely $ng$-feasible paths.

Looking at elementary paths, if we have visited customer $i \in \mathcal{V}$, we cannot visit it again. In other words, customer $i$ is in the path's *memory*. An elementary path cannot visit customers that are already in its memory, as it has already visited this customer. The $ng$-route relaxation adjusts the concept of the paths memory, the $ng$-memory, making it "forget" some customers under certain conditions.

We define separate "neighbourhoods" $\mathcal{V}_i \subseteq \mathcal{V}$ per node $i \in \mathcal{N}$. We require that $i \in \mathcal{V}_i$ for all $i \in \mathcal{V}$ and define $\mathcal{V}_0 = \mathcal{V}_{n+1} = \emptyset$. Whenever we travel over an arc $(i, j) \in \mathcal{A}$, we "remember" having visited customer $i$ if and only if $i \in \mathcal{V}_j$. This also holds for every other customer we remember. To make this idea more precise, let us consider some (partial) path $P = (i_0 = 0, i_1, \ldots, i_p)$ through $\mathcal{G}$. We denote our $ng$-memory of previously visited customers, while currently visiting $i_k$, as $\Pi_k(P) \subseteq \{i_1, \ldots, i_k\}$, with $k \in \{0, \ldots, p\}$. Note that we do not remember the starting depot 0, as it is trivial that we visit that node once. The same holds for the ending depot $n + 1$. We can determine our memory using recursive scheme (6).

$$\Pi_k(P) = \begin{cases} \emptyset & \text{, if } k = 0 \\ \big(\Pi_{k-1}(P) \cap \mathcal{V}_{i_k}\big) \cup \{i_k\} & \text{, if } k \in \{1, \ldots, p\} \end{cases} \tag{6}$$

In other words, travelling to customer $i_k$ makes us remember that customer, while simultaneously forgetting all customers not in $\mathcal{V}_{i_k}$. Note that $\Pi_p(P)$ in the recursion scheme (6) is equivalent to the definition of $\Pi(P)$ in Baldacci et al. (2011). The proof is shown in Lemma 1 of Appendix A.

We say a path $P = (i_0, \ldots, i_p)$ is $ng$-feasible if and only if it holds that $i_k \notin \Pi_{k-1}(P)$, for all $k \in \{1, \ldots, p\}$. That is, we are not allowed to visit any nodes that are currently in our memory. The concept of $ng$-memory and $ng$-paths nicely generalizes elementary paths. If $\mathcal{V}_i = \mathcal{V}$, for all $i \in \mathcal{V}$, then $ng$-paths are elementary paths, and the $ng$-memory is a perfect memory, as assumed in the previous sections. On the other hand, if $\mathcal{V}_i = \{i\}$ for all $i \in \mathcal{V}$, then the $ng$-memory corresponds to having no memory at all, which gives a

shortest path problem without elementarity.

To incorporate $ng$-memory into our SPPRC pricing problem, we only need to replace the $n$ elementarity resources $R_E^1, \ldots, R_E^n$ with $n$ $ng$-memory resources $R_{NG}^1, \ldots, R_{NG}^n$. These $n$ binary resources together will represent which nodes are currently in the $ng$-memory of some path. The windows are, identical to the elementarity resources, $a_i^{R_{NG}^j} = 0$ and $b_i^{R_{NG}^j} = 1$, for all $i \in \mathcal{N}$ and $j \in \mathcal{V}$. The REFs are defined as

$$f_{ij}^{R_{NG}^v}(\mathbf{u}) = \begin{cases} 0 & \text{, if } v \notin \mathcal{V}_j \\ u^{R_{NG}^v} & \text{, if } v \in \mathcal{V}_j,\ j \neq v \\ u^{R_{NG}^v} + 1 & \text{, if } j = v \end{cases} \qquad \forall (i,j) \in \mathcal{A},\ v \in \mathcal{V}. \tag{7}$$

These new REFs are still non-decreasing, which is shown in Lemma 2 in Appendix A.

Of course, $ng$-paths are not necessarily elementary. Fortunately, the SPP and RMP can handle non-elementary paths through $a_r^i$. Note that, in an integer solution, any path selected needs to be elementary, because restriction (1b) is forced with equality. This need not to be the case with the LP-relaxed (RMP). So, adding non-elementary paths to the master problem does not affect the final integer solution, but can affect the LP lower bound. Keep in mind that $\mathcal{V}_i$ should be chosen carefully. The smaller $|\mathcal{V}_i|$, the lower and less tight the LP bound is, but a larger $|\mathcal{V}_i|$ gives a higher LP bound at the cost of a more expensive pricing problem.

## 3.4 Capacity Cuts and Strengthened Capacity Cuts

When solving the LP relaxation of the RMP (and thus the SPP), we receive a lower bound on the optimal integer objective value. It is preferable that this lower bound is as high as possible, as close to the integer objective as possible. One way to raise this bound is by introducing valid inequalities to the RMP, as was first done by Kohl (1995) for the VRPTW. Valid inequalities should be satisfied by all integer solutions, but not necessarily by non-integer solutions. In other words, those inequalities cut off a part of the non-integer feasible space, while leaving the integer solution space intact.

Robust valid inequalities for the CVRP and the VRPTW are the capacity cuts (CCs), and have been used successfully in previous literature to raise the LP bound (Augerat et al., 1998; Baldacci, Christofides, & Mingozzi, 2008; Fukasawa et al., 2006). If we let $b_r^{ij}$ be the number of times route $r$ travels over arc $(i,j) \in \mathcal{A}$, we can define new variables $z_{ij}$, for all $(i,j) \in \mathcal{A}$ as

$$z_{ij} = \sum_{r \in \mathcal{R}} b_r^{ij} x_r, \tag{8}$$

so that $z_{ij}$ equals the number of vehicles that travel over arc $(i,j)$. In an integer solution, $z_{ij}$ is necessarily binary, but need not be in a non-integer solution.

The capacity cuts state that the number of times vehicles travel into some set of customers $\mathcal{S} \subseteq \mathcal{V}$, with $|\mathcal{S}| \geq 2$, should be at least some lower bound on the number of vehicles needed to serve the customers in $\mathcal{S}$. Usually, this lower bound is set to the total demand in $\mathcal{S}$, divided by the capacity of one vehicle. Using the $z_{ij}$-variables, Naddef and Rinaldi (2002) define the capacity cuts (CCs) as

$$\sum_{(i,j) \in \mathcal{A}:\, i \notin \mathcal{S},\, j \in \mathcal{S}} z_{ij} \geq \left\lceil \frac{\sum_{j \in \mathcal{S}} q_j}{Q} \right\rceil \qquad \forall \mathcal{S} \subseteq \mathcal{V} :\ |\mathcal{S}| \geq 2. \tag{9}$$

We can rewrite these inequalities easily into the $x_r$ variables using definition (8):

$$\begin{aligned}
\sum_{r \in \mathcal{R}} \left( \sum_{(i,j) \in \mathcal{A}:\, i \notin \mathcal{S},\, j \in \mathcal{S}} b_r^{ij} \right) x_r &\geq \left\lceil \frac{\sum_{j \in \mathcal{S}} q_j}{Q} \right\rceil \qquad \forall \mathcal{S} \subseteq \mathcal{V} :\ |\mathcal{S}| \geq 2 \\
\sum_{r \in \mathcal{R}} \left( \sum_{(i,j) \in \mathcal{A}} \beta_{\mathcal{S}}^{ij} b_r^{ij} \right) x_r &\geq \left\lceil \frac{\sum_{j \in \mathcal{S}} q_j}{Q} \right\rceil \qquad \forall \mathcal{S} \subseteq \mathcal{V} :\ |\mathcal{S}| \geq 2 \\
\sum_{r \in \mathcal{R}} \zeta_r^{CC}(\mathcal{S}) x_r &\geq \left\lceil \frac{\sum_{j \in \mathcal{S}} q_j}{Q} \right\rceil \qquad \forall \mathcal{S} \subseteq \mathcal{V} :\ |\mathcal{S}| \geq 2.
\end{aligned} \tag{10}$$

Here, we defined parameter $\beta_{\mathcal{S}}^{ij}$, which is equal to 1 if $i \notin \mathcal{S}$ and $j \in \mathcal{S}$ and 0 otherwise. Furthermore, $\zeta_r^{CC}(\mathcal{S})$ is the number of times route $r \in \mathcal{R}$ travels into set $\mathcal{S}$.

Note that it is impractical to include CCs for all $\mathcal{S} \subseteq \mathcal{V} :\ |\mathcal{S}| \geq 2$, as there are an exponential number of them. In practice, one uses a small collection of sets $\mathcal{S}_1, \ldots, \mathcal{S}_K$

with $\mathcal{S}_k \subseteq \mathcal{V}$, $|\mathcal{S}_k| \geq 2$, for all $k \in \{1, \ldots, K\}$, $K \geq 1$ for which to include the capacity cuts in a cutting planes fashion. That is, one only adds such a set $\mathcal{S}$ if it cuts off the current LP solution. This is also what we will assume for the remainder of this thesis.

Let us consider the example instance in Figure 1, Instance A. In the figure, square nodes denote the depots and round nodes the customers. The vehicle capacity $Q$ equals 3 and the customer demands are displayed next to the nodes. Instance A has an LP solution consisting of 4 different routes, each with a usage of $\frac{1}{2}$.

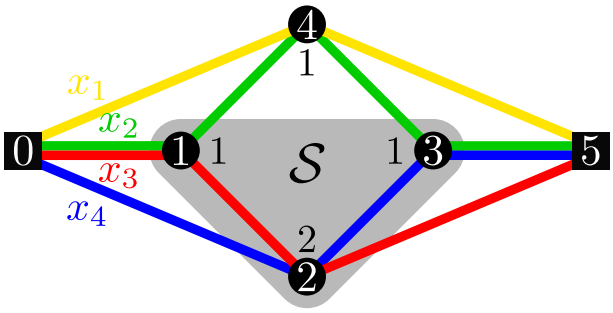



Figure 1: Example Instance A, in which a CC can be strengthened.

The LP solution of Instance A adheres to all CCs. Let us inspect $\mathcal{S} = \{1, 2, 3\}$. We need at least $\lceil \frac{4}{3} \rceil = 2$ vehicles to serve these customers. However, inspecting the figure, we only use $x_2 + x_3 + x_4 = \frac{3}{2}$ routes. The reason this does not violate the CC is because the CC corresponding to $\mathcal{S} = \{1, 2, 3\}$ equals

$$2x_2 + x_3 + x_4 \geq 2,$$

which holds for the presented solution. Because route 2 enters the set twice, it is counted multiple times in the CC. We can also observe this from equation (10). Thus, it seems we can strengthen the capacity cuts by counting every route at most once. If we do this, we obtain the strengthened capacity cuts (SCCs):

$$\begin{aligned} \sum_{r \in \mathcal{R}} \min\left\{1,\, \zeta_r^{CC}(\mathcal{S})\right\} x_r &\geq \left\lceil \frac{\sum_{j \in \mathcal{S}} q_j}{Q} \right\rceil \qquad && \forall \mathcal{S} \subseteq \mathcal{V} :\ |\mathcal{S}| \geq 2 \\ \sum_{r \in \mathcal{R}} \zeta_r^{SCC}(\mathcal{S}) x_r &\geq \left\lceil \frac{\sum_{j \in \mathcal{S}} q_j}{Q} \right\rceil \qquad && \forall \mathcal{S} \subseteq \mathcal{V} :\ |\mathcal{S}| \geq 2. \end{aligned} \tag{11}$$

We can interpret the binary parameter $\zeta_r^{SCC}(\mathcal{S})$ as an indicator whether route $r$ crosses

at least one arc going into $\mathcal{S}$ or not, or as the number of times route $r \in \mathcal{R}$ travels into $\mathcal{S}$ *for the first time*. Inequalities (11) were first introduced by Baldacci, Hadjiconstantinou, and Mingozzi (2004). As we can see from (11), the SCC corresponding to $\mathcal{S} = \{1, 2, 3\}$ is indeed violated in Instance A.

**Incorporation into the Pricing Problem** If one wants to include valid inequalities such as the CCs or SCCs into a column generation framework, the duals of such inequalities must be incorporated into the pricing problem. This is straightforward for the CCs. If we denote $\gamma_{\mathcal{S}} \geq 0$ the duals associated with constraints (10), the reduced costs $\rho_r$ of a route $r \in \mathcal{R}$ become

$$\begin{aligned}
\rho_r &= \sum_{(i,j)\in\mathcal{A}} c_{ij} b_r^{ij} - \sum_{j\in\mathcal{V}} \lambda_j a_r^j - \sum_{k=1}^{K} \sum_{(i,j)\in\mathcal{A}} \gamma_{\mathcal{S}_k} \beta_{\mathcal{S}_k}^{ij} b_r^{ij} \\
&= \sum_{(i,j)\in\mathcal{A}} (c_{ij} - \lambda_j) b_r^{ij} - \sum_{k=1}^{K} \sum_{(i,j)\in\mathcal{A}} \gamma_{\mathcal{S}_k} \beta_{\mathcal{S}_k}^{ij} b_r^{ij} \\
&= \sum_{(i,j)\in\mathcal{A}} \left( c_{ij} - \lambda_j - \sum_{k=1}^{K} \gamma_{\mathcal{S}_k} \beta_{\mathcal{S}_k}^{ij} \right) b_r^{ij} \\
&= \sum_{(i,j)\in\mathcal{A}} \left( \bar{c}_{ij} - \sum_{k=1}^{K} \gamma_{\mathcal{S}_k} \beta_{\mathcal{S}_k}^{ij} \right) b_r^{ij}.
\end{aligned}$$

The CCs can thus easily be incorporated into the pricing problem, as one only needs to adjust the reduced arc costs using the new duals $\gamma_{\mathcal{S}}$, as seen from the derivation above. That is, we subtract the dual $\gamma_{\mathcal{S}}$ from the reduced arc costs of arc $(i, j) \in \mathcal{A}$ if $i \notin \mathcal{S}$ and $j \in \mathcal{S}$. In other words, the CCs are robust, as their inclusion do not change the pricing problem.

The duals of the SCCs, $\gamma_{\mathcal{S}}^{SCC} \geq 0$, are not as easy to incorporate in the pricing problem

as the ordinary capacity cuts. After all, the reduced costs $\rho_r$ of a route $r \in \mathcal{R}$ become

$$\begin{aligned}\rho_r &= \sum_{(i,j)\in\mathcal{A}} c_{ij} b_r^{ij} - \sum_{j\in\mathcal{V}} \lambda_j a_r^j - \sum_{k=1}^{K} \gamma_{\mathcal{S}_k}^{SCC} \zeta_r^{SCC}(\mathcal{S}_k) \\ &= \sum_{(i,j)\in\mathcal{A}} (c_{ij} - \lambda_j) b_r^{ij} - \sum_{k=1}^{K} \gamma_{\mathcal{S}_k}^{SCC} \zeta_r^{SCC}(\mathcal{S}_k) \\ &= \sum_{(i,j)\in\mathcal{A}} \bar{c}_{ij} b_r^{ij} - \sum_{k=1}^{K} \gamma_{\mathcal{S}_k}^{SCC} \zeta_r^{SCC}(\mathcal{S}_k).\end{aligned}$$

We cannot simply adjust the reduced arc costs to include these duals, as the value of $\gamma_{\mathcal{S}}^{SCC}$ is only subtracted at most once. We can adjust the REFs corresponding to the reduced cost resource $R_C$ to incorporate the SCC duals. If a partial path $P = (i_0 = 0, i_1, \ldots, i_p)$ visits any node in $\mathcal{S}$, then it has visited an arc going into $\mathcal{S}$. This holds, as $0 \notin \mathcal{S}$ for any $\mathcal{S} \subseteq \mathcal{V}$. So, we can adjust the REFs as follows:

$$f_{ij}^{R_C}(\mathbf{u}) = u^{R_C} + \bar{c}_{ij} - \sum_{\substack{k:\, i\notin\mathcal{S}_k,\, j\in\mathcal{S}_k, \\ u^{R_E^v}=0 \;\forall v\in\mathcal{S}_k, \\ k\in\{1,\ldots,K\}}} \gamma_{\mathcal{S}_k}^{SCC} \qquad \forall (i,j) \in \mathcal{A}. \tag{12}$$

In other words, the discount $\gamma_{\mathcal{S}}^{SCC}$ is applied on arc $(i,j) \in \mathcal{A}$ only if $(i,j)$ is indeed an arc going into $\mathcal{S}$ and the partial route has not yet visited any node in $\mathcal{S}$. The latter constraint is translated into $u^{R_E^v} = 0$, $\forall v \in \mathcal{S}$. Note that $f_{ij}^{R_C}$ is still non-decreasing in $\mathbf{u}$, as increasing $u^{R_C}$ increases $f_{ij}^{R_C}$ and increasing any $u^{R_E^v}$, for any $v \in \mathcal{V}$, either increases $f_{ij}^{R_C}$ by $\gamma_{\mathcal{S}}^{SCC} \geq 0$ for some $\mathcal{S} \in \{\mathcal{S}_1, \ldots, \mathcal{S}_K\}$ or does nothing.

Equation (12) shows that the SCCs are robust if we use the elementarity resources $R_E^1, \ldots, R_E^n$. However, these are not available when we use the *ng*-route relaxation. So, it is not obvious how to include the SCCs into the pricing problem without altering the structure of it. In other words, the SCCs are non-robust when using the *ng*-route relaxation. Section 4 introduces the *ng*-capacity cuts, which are robust even when using the *ng*-route relaxation.

## 3.5 Algorithm for Obtaining a Lower Bound

Now that we have discussed the labelling algorithm, the pricing states and valid inequalities, we can state the full algorithm used to determine a lower bound on SPP. This algorithm is best summarized in a flowchart and is shown in Figure 2.

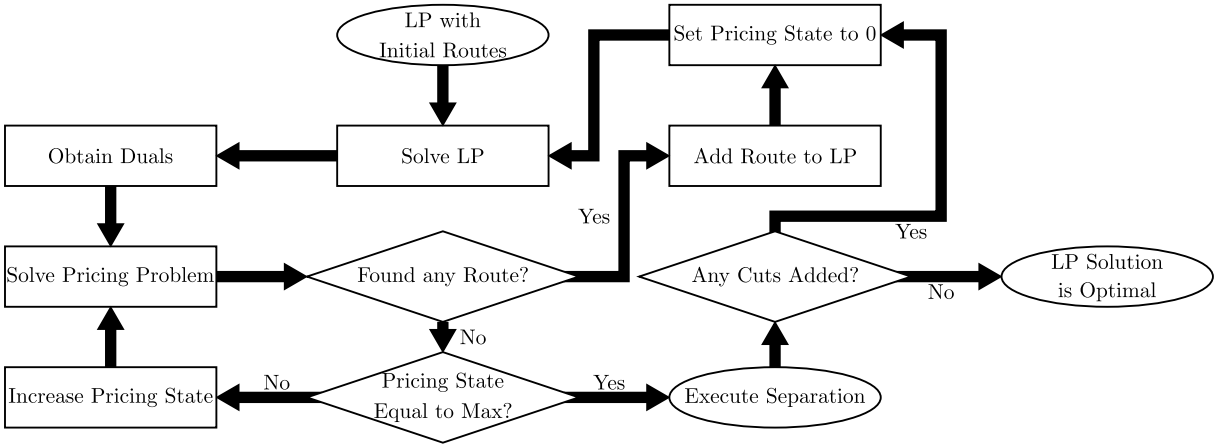



Figure 2: Flow chart for solving a CVRP or VRPTW to LP optimality.

As the node "LP with Initial Routes" suggests, we initialise $\mathcal{R}'$ with certain routes. We use the $n$ trivial routes $(0, i, n+1)$ for all $i \in \mathcal{V}$. The node "Solve LP" refers to solving RMP with the current choice of $\mathcal{R}'$, which is done by using CPLEX. We solve the pricing problem in node "Solve Pricing Problem" with Algorithm 1. In the "Execute Separation" phase we try to find violated valid inequalties. We either add no cuts at all (NoCuts), find and add CCs with CVRPSEP (CCSEP) (Lysgaard, 2003), or find NGCCs using any of the separation algorithms presented in Section 4.2.

## 3.6 Obtaining an Integer Solution

With the techniques described in the previous sections, one can solve the LP relaxation of the (SPP) and strengthen it with valid inequalities. However, rarely are the optimal values of $x_r$, $r \in \mathcal{R}'$ binary. So, in order to solve the CVRP or the VRPTW to integer optimality, we use a branch-price-and-cut algorithm. We will describe the methods used and this algorithm in the next few sections.

### 3.6.1 Branching Variable

An obvious, but flawed, choice of branching variables would be $x_r$. While forcing $x_r = 1$ for some route $r \in \mathcal{R}$ would be simple, fixing $x_r = 0$ is not as straightforward. After all,

one has then to prevent this very specific route from being generated again in the pricing problem. This complicates the pricing problem noticeably.

Instead, as is commonly done for these problems, we branch on the arc flows $z_{ij}$ (8), as introduced in Desrochers et al. (1992). Note that, if all arc flows $z_{ij}$ are binary, for all $(i,j) \in \mathcal{V} \times \mathcal{V}$, then $x_r$ are also binary for all $r \in \mathcal{R}$. We do not have to branch on the arcs coming from depot 0 and going to depot $n+1$, as the integrality of those arc flows is also implied by the arc flows between customers.

Branching on those arc flows can easily be incorporated into the pricing problem. If we force $z_{ij} = 0$ for some $(i,j) \in \mathcal{V} \times \mathcal{V}$, then we delete arc $(i,j)$ from the graph. If we set $z_{ij} = 1$ for some $(i,j) \in \mathcal{V} \times \mathcal{V}$, then we delete arcs $(i,v)$, $(u,j)$ and $(j,i)$, for all $v \in \mathcal{N} \setminus \{j\}$ and $u \in \mathcal{N} \setminus \{i\}$. In other words, any route that visits $i$, must immediately visit $j$ afterwards. We also delete arc $(j,i)$, as no elementary route will visit $i$ and $j$ more than once. With these modified arcs, the labelling algorithm can be used as normal to generate new routes while certain arc restrictions are active.

Note that setting $z_{ij}$ to 1 means we can infer that a path has visited $i$ if and only if it has visited $j$. In other words, if a (partial) path's $ng$-memory contains $i$, it must have visited $j$ and vice versa. In other words, this means we can add $j$ to any neighbourhood $\mathcal{V}_v$, $v \in \mathcal{V}$ if $i$ is in that neighbourhood and vice versa. This increases the number of routes that dominate each other, thus speeding up the labelling algorithm. To be more precise, we update the neighbourhoods as follows:

$$\mathcal{V}_v \leftarrow \begin{cases} \mathcal{V}_v \cup \{j\}, & \text{if } i \in \mathcal{V}_v \\ \mathcal{V}_v \cup \{i\}, & \text{if } j \in \mathcal{V}_v \\ \mathcal{V}_v, & \text{otherwise} \end{cases}, \qquad \forall v \in \mathcal{V}.$$

As the $ng$-capacity cuts (see Section 4) depend on the neighbourhoods $\mathcal{V}_v$, it is crucial to recalculate them every time the neighbourhoods are updated.

### 3.6.2 Branching Scheme

Now that we know our branching variable, we can describe our branch-price-and-cut algorithm. The main idea of the algorithm is to solve LP relaxations of (RMP) while introducing restrictions on the branching variables in order to obtain an integer optimal solution.

At the start of the algorithm, we solve the *root node*, which is the original LP relaxation of (RMP), with no restrictions on the branching variables. This usually gives a non-integer, fractional solution. If such a solution is fractional, it means that at least one arc $(i, j) \in \mathcal{V} \times \mathcal{V}$ has a fractional $z_{ij}$-value. We then choose one such arc $(i, j)$ to branch on. We choose the most fractional arc, that is, the arc $(i, j) \in \mathcal{A}$ that maximizes $|z_{ij} - \frac{1}{2}|$. This is a popular rule that was first introduced in Padberg and Rinaldi (1991) in the context of symmetric travelling salesman problems, a special case of the CVRP. Branching means we create two child nodes connected to the parent, one in which $z_{ij} = 0$ is enforced and one in which we set $z_{ij} = 1$. Then, the children are solved to LP optimality, which gives rise to more nodes. The only time we cannot create two child nodes is when a solution is integer, as then all $z_{ij}$-values are integer as well.

Nodes in which the optimal LP solution value is higher than the best known integer solution value, need not to be explored further. After all, the LP solution value is a lower bound on any integer solution value that could possibly be obtained from any descendent of that node. The quality of the LP solution values can thus significantly speed up the branch-price-and-cut algorithm, as higher LP bounds mean more branches are cut off early in the tree. This also advocates the use of valid inequalities in the nodes of the search tree.

Note that the routes generated to solve the root node to LP optimality may not be enough to solve any other node to LP optimality. In other words, there may still be routes with negative reduced costs in any other node. Therefore, it is necessary to solve the pricing problem in these nodes as well. Luckily, we can incorporate the restrictions on $z_{ij}$ into the pricing problem, as described in the previous section. Also, adding a restriction on some $z_{ij}$ may make some existing routes infeasible. It is therefore also needed to restrict $x_r$ for certain routes to 0. Note that these restrictions do not affect the pricing

problem, as those infeasible routes cannot be generated by the labelling algorithm.

### 3.6.3 Farkas Pricing

As mentioned before, branching on some $z_{ij}$ may make some routes infeasible. It is possible that too many routes are rendered infeasible, thus making it impossible to solve the RMP for the current set of routes $\mathcal{R}'$. To overcome such an infeasibility, we use a technique called Farkas pricing (Lübbecke, 2010). The idea is based on a well-known fact that an infeasible LP, implies that the dual of the LP is unbounded. This means that there is a dual extreme direction $\boldsymbol{\pi}$, which can usually be obtained by an LP solver. Let us denote the column for a route $r \in \mathcal{R}$ in the constraint matrix with $\boldsymbol{\alpha}_r$. Then, an application of Farkas Lemma states that adding a route $r$ with $\boldsymbol{\pi}^\top \boldsymbol{\alpha}_r > 0$ makes the dual bounded and thus the LP feasible. In other words, we have to find a route $r$ such that $-\boldsymbol{\pi}^\top \boldsymbol{\alpha}_r < 0$. Note that the left-hand size side of the expression is identical to the definition of reduced costs, with the route costs $c_r$ omitted and the use of the dual extreme direction $\boldsymbol{\pi}$ instead of the ordinary duals. This means we can use our labelling algorithm to find a route that repairs feasibility; we only have to change the cost component of the REF. Indeed, just omitting the $c_{ij}$ from the costs and replacing the duals with the dual extreme direction is enough.

### 3.6.4 The Branch-Price-and-Cut algorithm

The full Branch-Price-and-Cut algorithm used in this thesis is given in Algorithm 2.

**Algorithm 2:** Branch-Price-and-Cut.

1 $B_0 \leftarrow$ branch node representing the root node;
2 $\mathcal{B}_U \leftarrow \{B_0\}$, the set of unprocessed branching nodes;
3 $\mathcal{B}_P \leftarrow \emptyset$, the set of processed branching nodes;
4 $C_{int} \leftarrow +\infty$, the best integer solution value;
5 **while** $\mathcal{B}_A \neq \emptyset$ **do**
6 $\quad B \leftarrow$ some node in $\mathcal{B}_U$;
7 $\quad \mathcal{B}_U \leftarrow \mathcal{B}_U \setminus \{B\}$;
8 $\quad \mathcal{B}_P \leftarrow \mathcal{B}_P \cup \{B\}$;
9 $\quad$ Solve the LP of node $B$ using the algorithm in Figure 2;
10 $\quad C_{LP} \leftarrow$ LP solution value;
11 $\quad$ **if** solution is integer **then**
12 $\quad\quad C_{int} \leftarrow \min\{C_{int}, C_{LP}\}$;
13 $\quad$ **else**
14 $\quad\quad$ **if** $C_{LP} \leq C_{int}$ **then**
15 $\quad\quad\quad$ Choose the arc $(i, j)$ that maximizes $|z_{ij} - \frac{1}{2}|$;
16 $\quad\quad\quad$ Create child nodes $B_1$ and $B_2$, one with $z_{ij} = 1$ and one with $z_{ij} = 0$;
17 $\quad\quad\quad \mathcal{B}_U \leftarrow \mathcal{B}_U \cup \{B_1, B_2\}$;
18 $\quad\quad$ **end**
19 $\quad$ **end**
20 **end**
21 Return best found integer solution;

Our node selection (line 6) is based on a depth-first traversal of the branching tree, where we first investigate the child with $z_{ij} = 1$.

# 4 $ng$-Capacity Cuts

The usability of valid inequalities such as the CCs and SCCs in a column generation framework is the ability to include the duals of those cuts into the pricing problem, as otherwise we cannot calculate the reduced costs of a route $r \in \mathcal{R}$ accurately. Ideally, we do not want to increase the complexity of the pricing problem. As we mentioned in Section 3.4, the SCCs are not robust, if one uses the $ng$-route relaxation. This section introduces a new type of valid inequalities, the $ng$-capacity cuts (NGCCs), which is robust if used with the $ng$-route relaxation.

Let us define $\zeta_r^{NG}(\mathcal{S})$ as the number of times route $r \in \mathcal{R}$ travels into $\mathcal{S}$ for the first time, *according to ng-memory*. To be more exact, we can write $\zeta_r^{NG}(\mathcal{S})$ for some route $r = (i_0 = 0,\, i_1,\, \ldots,\, i_{|r|},\, i_{|r|+1} = n+1)$, $|r| \geq 1$, as

$$\zeta_r^{NG}(\mathcal{S}) = \left|\left\{k :\, i_k \in \mathcal{S},\, \Pi_{k-1}(r) \cap \mathcal{S} = \emptyset,\, k \in \{1, \ldots, |r|\}\right\}\right|. \tag{13}$$

Then, we can define the NGCCs as

$$\sum_{r \in \mathcal{R}} \zeta_r^{NG}(\mathcal{S}) x_r \geq \left\lceil \frac{\sum_{j \in \mathcal{S}} q_j}{Q} \right\rceil \qquad \forall \mathcal{S} \subseteq \mathcal{V} :\, |\mathcal{S}| \geq 2. \tag{14}$$

As $\zeta_r^{SCC}(\mathcal{S}) \leq \zeta_r^{NG}(\mathcal{S}) \leq \zeta_r^{CC}(\mathcal{S})$, it follows that the NGCCs are stronger than the CCs, but the SCCs are stronger than the NGCCs. Furthermore, If we set $\mathcal{V}_i = \{i\}$, for all $i \in \mathcal{V}$, the NGCCs are equivalent to the CCs, as then $\zeta_r^{NG}(\mathcal{S}) = \zeta_r^{CC}(\mathcal{S})$. On the other end of the spectrum, setting $\mathcal{V}_i = \mathcal{V}$ for all $i \in \mathcal{V}$ makes the NGCCs identical to the SCCs, as then $\zeta_r^{NG}(\mathcal{S}) = \zeta_r^{SCC}(\mathcal{S})$. We prove these facts in Lemma 3 of Appendix A.

**Incorporation into the Pricing Problem** As mentioned before, equation (12) shows that we cannot include the SCC duals into our pricing problem robustly if we use the $ng$-route relaxation. After all, the elementarity resources $R_E^1, \ldots, R_E^n$ are not available when using this relaxation. The NGCCs mitigate this problem. If we denote the duals of the NGCCs, (14), with $\gamma_{\mathcal{S}}^{NG} \geq 0$, we can compute the reduced costs $\rho_r$ of route $\in \mathcal{R}$ as

$$\rho_r = \sum_{(i,j) \in \mathcal{A}} \bar{c}_{ij} b_r^{ij} - \sum_{k=1}^{K} \gamma_{\mathcal{S}_k}^{NG} \zeta_r^{NG}(\mathcal{S}_k).$$

The number of times we subtract $\gamma_{\mathcal{S}}^{NG}$ from the reduced costs equals the number of arcs $(i, j)$ $r$ visits with $j \in \mathcal{S}$ and the $ng$-memory not containing any nodes of $\mathcal{S}$. In terms of resources, whether a discount is applied on an arc thus depends on the $ng$-memory resources $R_{NG}^1, \ldots, R_{NG}^n$. The cost REFs become

$$f_{ij}^{R_C}(\mathbf{u}) = u^{R_C} + \bar{c}_{ij} - \sum_{\substack{k:\, i \notin \mathcal{S}_k,\, j \in \mathcal{S}_k, \\ u^{R_{NG}^v} = 0 \;\forall v \in \mathcal{S}_k, \\ k \in \{1,\ldots,K\}}} \gamma_{\mathcal{S}}^{NG} \qquad \forall (i,j) \in \mathcal{A}. \tag{15}$$

Inspecting (15), we can see that the formula is identical to (12), except that the elementarity resources $R_E^1, \ldots, R_E^n$ are replaced with the $ng$-memory resources $R_{NG}^1, \ldots, R_{NG}^n$, thus making it compatible with the $ng$-route relaxation.

## 4.1 A Small Example

In this section, we show a small example for which the NGCCs raise the LP lower bound more than the CCs. This example instance, Instance B, is shown in Figure 3.

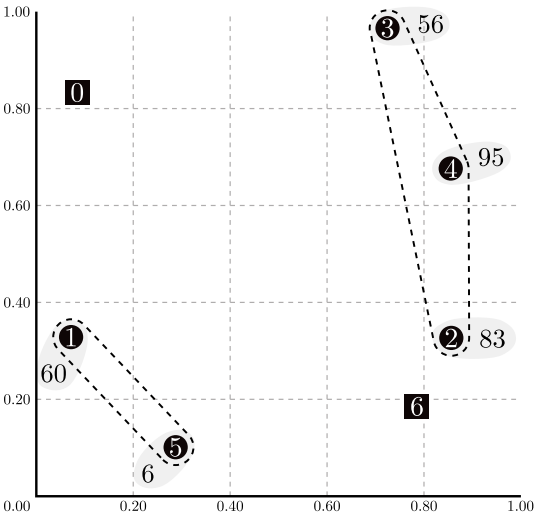



Figure 3: Instance B.

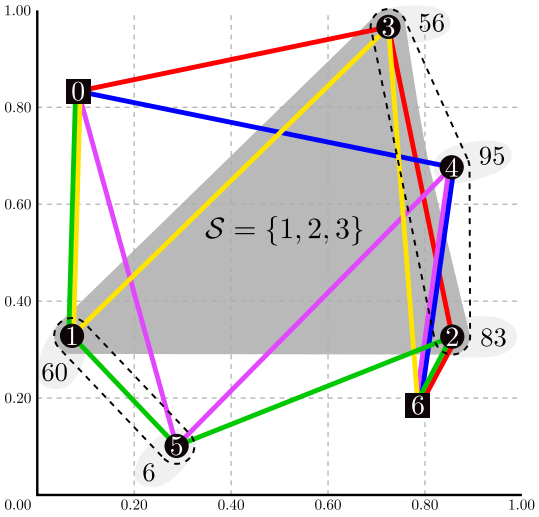



Figure 4: Optimal LP solution of Instance B, adhering to CCs.

The square nodes denote the start and end depot, labelled 0 and 6, respectively. The round nodes are the customers, with their demands denoted next to them. The position of the nodes correspond to their exact position on the graph, as travel costs between nodes are equal to the Euclidean distance between nodes, rounded down to 2 decimals. We partitioned the customers into two groups who share the same neighbourhoods. That

is, $\mathcal{V}_1 = \mathcal{V}_5 = \{1, 5\}$ and $\mathcal{V}_2 = \mathcal{V}_3 = \mathcal{V}_4 = \{2, 3, 4\}$, as is also shown in Figure 3. The costs of using a vehicle are 10, which means $c_{0i}$ equals the Euclidean distance between 0 and $i$, rounded to 2 decimals, plus 10, for $i \in \mathcal{V}$. The vehicle capacity is $Q = 150$.

The unique optimal IP solution of Instance B uses 3 vehicles that follow routes $(0, 3, 2, 6)$, $(0, 4, 6)$ and $(0, 1, 5, 6)$, with objective 32.95. The optimal LP objective of 28.065 is considerably lower and "uses" 2.5 vehicles. If we add all CCs to the problem, we get the solution presented in Figure 4 with objective 28.36. In this figure, a line between nodes denotes the usage of 0.5 vehicles on that particular arc. Figure 4 also highlights the set $\mathcal{S} = \{1, 2, 3\}$. The total demand of this set equals 199, so we need at least $\lceil \frac{199}{150} \rceil = 2$ vehicles to serve the customers in $\mathcal{S}$. The LP solution in Figure 4 adheres to the CC of this set, as arc $(0, 1)$ is used 1 times and $(0, 3)$ and $(5, 2)$ are used 0.5 times.

However, Figure 4 does not adhere to the NGCC of set $\mathcal{S}$. After all, arc $(5, 2)$ does not count, as we remember having visited node $1 \in \mathcal{S}$, while currently in node 5. So, we only travel 1.5 times into $\mathcal{S}$ for the first time, according to *ng*-memory. Adding the NGCCs to this problem, raises the objective to 28.515. This example shows that there are instances for which adhering to the CCs does not imply adhering to the NGCCs. Furthermore, it shows that adding NGCCs can indeed tighten the bounds between the LP and IP objectives more than adding the CCs.

## 4.2 Separation of Capacity Cuts

CCs can be separated by using the CVRPSEP package (Lysgaard, 2003; Lysgaard, Letchford, & Eglese, 2004). However, there seems to be no widely used methods for the separation of SCCs. Indeed, Baldacci et al. (2008) also use the CVRPSEP package for the separation of SCCs, which means that not all violated SCCs are found, as CVRPSEP can only separate violated CCs. Furthermore, as NGCCs are more complex in their definition than CCs or SCCs, we will not devise an algorithm to separate NGCCs from scratch. Instead, we will base our different separation algorithms on the CVRPSEP package, to ease implementation of these methods.

The most straightforward way to separate NGCCs is using the CVRPSEP package without

any modifications. Of course, we will not find all violated NGCCs this way. Furthermore, there are multiple ways one can separate these NGCCs. Two straightforward ways are as follows:

1. (NGSEP1) When the LP is solved to optimality, find all violated CCs using CVRPSEP, but add then as NGCCs. Then, re-solve the LP to optimality and find CCs again. Repeat until no more CCs are found.

2. (NGSEP2) When the LP is solved to optimality, find and add all violated CCs using CVRPSEP. Then, re-solve the LP to optimality and find CCs again. Repeat until no more CCs are found. Then, convert all CCs to NGCCs. Re-solve and repeat the whole process until no more additional CCs can be converted into NGCCs.

Note that neither of the methods dominates each other. That is, there are instances in which NGSEP1 achieves a higher objective value than NGSEP2 and instances in which the opposite is true. We give examples of such instances and discuss them in detail in Section B.1 of Appendix B.

As mentioned before, NGCCs are weaker than the SCCs. So, one can separate NGCCs by investigating all $\mathcal{S}$ for which the SCCs are violated, and then adding all those sets for which the corresponding NGCC is violated. However, as mentioned before, no specialised algorithms for separating SCCs seem to exist. Because of this, we will attempt to separate SCCs using the existing techniques to separate CCs.

As mentioned in (10), we can write the capacity cut for subset $\mathcal{S} \subseteq \mathcal{V}$ as

$$\sum_{r \in \mathcal{R}} \zeta_r^{CC}(\mathcal{S}) x_r \geq \left\lceil \frac{\sum_{j \in \mathcal{S}} q_j}{Q} \right\rceil,$$

with $\zeta_r^{CC}(\mathcal{S})$ as the number of times route $r \in \mathcal{R}$ travels into set $\mathcal{S}$. A CC can be turned into an SCC by capping the coefficients of $x_r$ to 1. In other words, by replacing $\zeta_r^{CC}(\mathcal{S})$ with $\zeta_r^{SCC}(\mathcal{S}) = \min\{\zeta_r^{CC}(\mathcal{S}), 1\}$. We first note that the upper bound for any $\zeta_r^{CC}(\mathcal{S})$ is $\left\lceil \frac{|r|}{2} \right\rceil$, where $|r|$ is the number of customers route $r$ visits. After all, this is the maximum number of times a route can enter an arbitrary subset of customers, by repeatedly exiting

and entering the set. So, because we know that

$$\zeta_r^{CC}(\mathcal{S}) \leq \left\lceil \frac{|r|}{2} \right\rceil, \qquad \text{and} \qquad \zeta_r^{SCC}(\mathcal{S}) = \min\{\zeta_r^{CC}(\mathcal{S}), 1\}$$

we can derive that

$$\begin{aligned} \left\lceil \frac{|r|}{2} \right\rceil \zeta_r^{SCC}(\mathcal{S}) &= \min\left\{ \left\lceil \frac{|r|}{2} \right\rceil \zeta_r^{CC}(\mathcal{S}), \left\lceil \frac{|r|}{2} \right\rceil \right\} \\ &\geq \zeta_r^{CC}(\mathcal{S}) \end{aligned}$$

provided that $\left\lceil \frac{|r|}{2} \right\rceil \geq 1$, which is true for all routes $r \in \mathcal{R}$.

So, let us assume some SCC is violated. That is, there exists an $\mathcal{S} \subseteq \mathcal{V}$ with

$$\sum_{r \in \mathcal{R}} \zeta_r^{SCC}(\mathcal{S}) x_r < \left\lceil \frac{\sum_{j \in \mathcal{S}} q_j}{Q} \right\rceil.$$

Then, we can derive the following:

$$\begin{aligned} \sum_{r \in \mathcal{R}} \zeta_r^{SCC}(\mathcal{S}) x_r &= \sum_{r \in \mathcal{R}} \zeta_r^{SCC}(\mathcal{S}) \left\lceil \frac{|r|}{2} \right\rceil \frac{x_r}{\left\lceil \frac{|r|}{2} \right\rceil} \\ &\geq \sum_{r \in \mathcal{R}} \zeta_r^{CC}(\mathcal{S}) \frac{x_r}{\left\lceil \frac{|r|}{2} \right\rceil} \\ &= \sum_{r \in \mathcal{R}} \zeta_r^{CC}(\mathcal{S}) \bar{x}_r, \end{aligned}$$

with $\bar{x}_r \equiv \frac{x_r}{\left\lceil \frac{|r|}{2} \right\rceil}$ the reduced arc flow. Using the above derivation, we can derive that, if an SCC is violated, then

$$\sum_{r \in \mathcal{R}} \zeta_r^{SCC}(\mathcal{S}) x_r < \left\lceil \frac{\sum_{j \in \mathcal{S}} q_j}{Q} \right\rceil \qquad \Longrightarrow \qquad \sum_{r \in \mathcal{R}} \zeta_r^{CC}(\mathcal{S}) \bar{x}_r < \left\lceil \frac{\sum_{j \in \mathcal{S}} q_j}{Q} \right\rceil.$$

In other words, a violated SCC means a CC is violated whose arc flows are reduced (reduced violation, or $r$-violated). This implies we can find all violated SCCs (and thus NGCCs) by reducing the arc flow and searching for $r$-violated CCs. Of course, we would find too many $r$-violated CCs that do not necessarily correspond to violated SCCs, but one can always check afterwards whether the corresponding SCC (or NGCC) is actually

violated.

However, simply reducing the arc flows is usually not enough, as then the coverage constraint (1b) does not hold anymore. To solve this problem, one can modify the graph as in Figure 5. This, in effect, triples the number of customers in the graph by introducing two dummy nodes per customer. Note that the dummy nodes have a demand of zero. Furthermore, if any of the dummy nodes appear in a found set $\mathcal{S}$, one can delete them without any consequences.

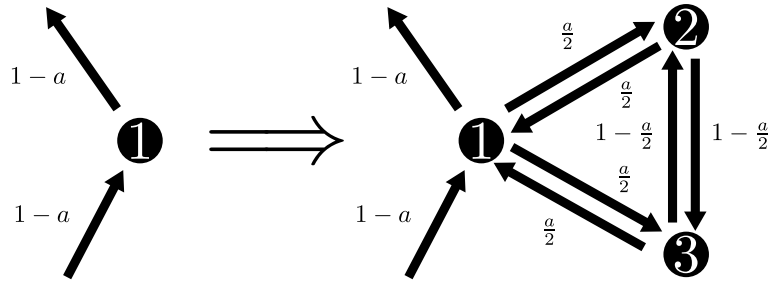



Figure 5: A method to restore the inflow and outflow of a node to 1.

So, using the reduced arc flows $\bar{x}_r$ and the technique in Figure 5 to restore the inflow and outflow, we can separate SCCs and NGCCs. This gives us the separation technique NGSEP3:

3. (NGSEP3) When the LP is solved to optimality, find all $r$-violated CCs using CVRPSEP with reduced arc flows $\bar{x}_r \equiv \frac{x_r}{\left\lceil \frac{|r|}{2} \right\rceil}$ and dummy nodes (Figure 5). Then, add all of the found CCs as NGCCs if they are violated in the original solution. Then, re-solve and repeat until no more CCs are found.

We have to point out that NGSEP3 is guaranteed to find all violated SCCs or NGCCs, given that all $r$-violated CCs can be separated (perfect separation of CCs). However, CVRPSEP is not perfect, as discussed in Lysgaard et al. (2004), on which the CVRPSEP package is based. Nevertheless, NGSEP3 is almost guaranteed to find as much or more NGCCs than NGSEP1 and NGSEP2, given the near-perfect performance of CVRPSEP (Lysgaard et al., 2004).

One drawback of NGSEP3 is that it can find too many $r$-violated CCs which can degrade the performance of the separation. It may be of interest to lower the arc flows in order to find more violated SCCs (or NGCCs), but not as much $r$-violated CCs as

NGSEP3. However, the guarantee of finding all violated SCCs and NGCCs of NGSEP3, given the finding of CCs is perfect, is desirable. So, we can devise a separation method that tries to find $r$-violated CCs, and reduces the arc flows further if no violated NGCCs are found. The resulting technique is NGSEP4:

4. (NGSEP4) Set $\kappa = 1$. Solve the LP to optimality. Find all $r$-violated CCs using CVRPSEP with reduced arc flows $\frac{x_r}{\min\left\{\kappa, \left\lceil \frac{|r|}{2} \right\rceil\right\}}$ and dummy nodes (Figure 5). Then, add all of the found CCs as NGCCs if they are violated in the original solution. If there are no NGCCs found, increase $\kappa$ with 1. Else, reset $\kappa$ to 1. Then, re-solve and repeat until no more violated NGCCs are found and $\kappa$ equals $\max_{r \in \mathcal{R}'} \left\{ \left\lceil \frac{|r|}{2} \right\rceil \right\}$.

Note that we do not want to reduce the arc flows with a factor higher than $\left\lceil \frac{|r|}{2} \right\rceil$, so we divide by $\min\left\{\kappa, \left\lceil \frac{|r|}{2} \right\rceil\right\}$. Furthermore, we stop whenever no NGCCs are found and increasing $\kappa$ does not reduce any arc flow further. If the finding of $r$-violated CCs is perfect, this method is guaranteed to find all violated NGCCs (and SCCs), just like NGSEP3. However, it may save time by not always dividing the arc flow with the upper bound. Observe that NGSEP3 is actually equal to NGSEP4 with $\kappa$ fixed to any number higher than $\max_{r \in \mathcal{R}} \left\{ \left\lceil \frac{|r|}{2} \right\rceil \right\}$.

According to Lysgaard et al. (2004), CVRPSEP executes four different heuristics. If the first one finds some $r$-violations, the other three are never executed. This is problematic, as we desire to find as many $r$-violations as possible, to increase the number of NGCC violations found. So, for NGSEP3 and NGSEP4, we actually use an adjusted version of CVRPSEP that always executes its four heuristics.

Furtermore, NGSEP4 is guaranteed to perform at least as good as NGSEP1, as the first few iterations with $\kappa = 1$ are identical to NGSEP1, and actually increases $\kappa$ when NGSEP1 would have terminated.

## 4.3 Symmetry and $ng$-Capacity Cuts

Now, we will discuss the matter of symmetry and $ng$-capacity cuts. Bi-directional labelling algorithms to solve the SPPRC are very popular in recent literature (Baldacci et al., 2011; Pecin et al., 2017). Thus, we feel the need to discuss the implications of wanting to use

such an algorithm in combination with the NGCCs, despite this thesis using a mono-directional labelling algorithm.

### 4.3.1 Bi-Directional Labelling

Algorithm 1 is sometimes called a mono-directional labelling algorithm, as we only create paths from the starting depot 0 to the ending depot $n+1$. In the worst case, we need to keep exponentially more labels in memory the longer the partial paths become. To limit this problem, many recent algorithms to solve the SPPRC use bi-directional labelling. Bi-directional labelling, as shown in Righini and Salani (2006), limits the number of generated labels and reduces the runtime significantly. The main idea is to simultaneously create forward and backward paths, starting at the depots 0 and $n+1$ respectively. Then, one joins backward and forward partial paths together to create full feasible routes.

As mentioned before, a bi-directional labelling algorithm creates two types of paths: forward and backward paths. As the forward paths are identical to the paths discussed in previous sections, we will now introduce the concept of backward paths briefly.

### 4.3.2 Backwards Paths

Similar to a forward path, a backward path is defined as a $q+1$-tuple of nodes $\bar{P} = (j_0 = n+1, j_1, \ldots, j_q)$, with $j_k \in \mathcal{N}$ for all $k \in \{0, \ldots, q\}$ and $(j_k, j_{k-1}) \in \mathcal{A}$ for all $k \in \{1, \ldots, q\}$. Note that the arcs are pointed "backwards" in such a path, as the name suggests. In a bi-directional labelling algorithm, one also represents backward paths using a label $K = (j, \mathbf{u}, K_{prev})$, with the "empty" label $K_{n+1} = (n+1,\ \mathbf{a}'_{n+1},\ NO)$.

Extending a backward label is identical to extending a forward label, except that resource intervals and REFs can differ. One actually transposes the cost and time matrices $\bar{c}_{ij}$ and $t_{ij}$, as arcs point backwards in a backward path. Furthermore, we use the inverted time windows, $e'_i$ and $l'_i$ of node $i \in \mathcal{N}$ instead of $e_i$ and $l_i$, defined as $e'_i = l_{n+1} - l_i$ and $l'_i = l_{n+1} - e_i$ respectively (Baldacci et al., 2011). Elementarity, capacity and $ng$-memory are updated identically as in the forward case.

Note that, even though $ng$-memory is updated identically in a backward path, it ac-

tually corresponds to the concept of inverse *ng*-memory, as introduced in Baldacci et al. (2011). The reason we can still join forward and backward paths is because *ng*-feasibility is symmetric. However, the *ng*-memory along a backward path can be drastically different than the *ng*-memory along a forward path that travels over the same edges. These two facts are summarized in Proposition 1.

**Proposition 1.** *ng-feasibility is symmetric, whereas ng-memory itself is not.*

*Proof.* See Appendix A. □

To merge a forward path $P$ implied by label $L = (i, \mathbf{u}, L_{prev})$ and a backward path $\bar{P}$ implied by $K = (j, \mathbf{w}, K_{prev})$ into a single, feasible route Baldacci et al. (2011) give conditions.

### 4.3.3 Including Capacity Cut Duals

The duals of the CCs can easily be incorporated into backward labelling, as only the arc costs need to be adjusted. However, the duals of the NGCCs are much more difficult to include. The reason follows from Proposition 1: *ng*-memory is not symmetric. The discount of a set $\mathcal{S}$ is applied whenever we travel into $\mathcal{S}$ for the first time, according to *ng*-memory. However, backward paths have different *ng*-memory, due to the asymmetry, and can thus apply the discount a different number of times. This is shown in Instance C, Figure 6, where round nodes represent customers, the square nodes depots and the set next to a node is the neighbourhood $\mathcal{V}_i$ of that node.

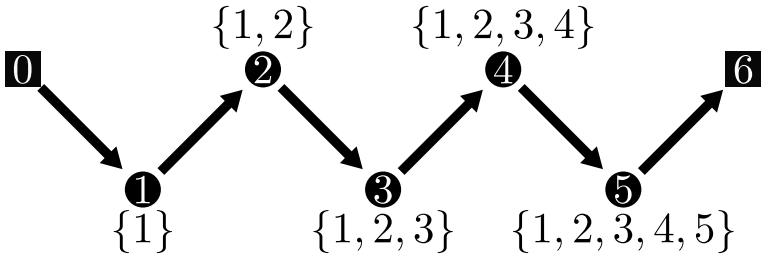



Figure 6: Instance C, containing a route with asymmetric *ng*-memory.

Let us consider the subset $\mathcal{S} = \{2, 4\}$. Then, the path $P = (0, 1, 2, 3, 4, 5, 6)$ only enters $\mathcal{S}$ one time, according to *ng*-memory, as it remembers having visited node 2 when travelling over arc $(3, 4)$. However, the backward path $\bar{P} = (6, 5, 4, 3, 2, 1, 0)$, which visits the same customers and travels over the same arcs, actually enters $\mathcal{S}$ twice according to

$ng$-memory. Because backward paths are created from the ending depot, 6 in this case, and are extended in the reverse direction, the $ng$-memory is also updated in the reverse order. In this example, backward updating actually means the path does not remember any previous nodes visited. So, as $ng$-memory is not symmetric, one cannot correctly estimate how many times a backward path enters some set $\mathcal{S}$ for the first time, according to (forward) $ng$-memory.

This means we cannot accurately compute the reduced costs of a partial backward path, as its costs depend on the forward path it will be connected to. More specifically, it depends on the $ng$-memory of the last node in the forward path. Instance D in Figure 7 illustrates this problem.

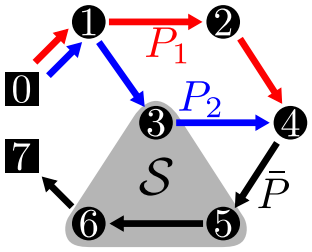



Figure 7: Instance D, the problem of joining forward and backward paths.

Assuming perfect $ng$-memory in Instance D (that is, $\mathcal{V}_i = \mathcal{V}$ for all $i \in \mathcal{V}$) the reduced costs of backward path segment $\bar{P} = (7, 6, 5, 4)$ depend on which forward partial path it is connected to. If it is connected to $P_1 = (0, 1, 2, 4)$, the costs of $\bar{P}$ include the discount of travelling into $\mathcal{S}$ for the first time. If connected to $P_2 = (0, 1, 3, 4)$, this discount should instead be applied to the costs of $P_2$, not $\bar{P}$.

We introduce two ways to mitigate this problem, but both will reduce the effectiveness of using the backward labelling in the first place. We call these two methods "memory enumeration" and "cost bounding" respectively.

**Memory Enumeration** In memory enumeration, we add $n$ additional binary resources to labels, which we call the "starting $ng$-memory". This corresponds to the *forward* $ng$-memory of the last node in the forward path it can be connected to. With this starting $ng$-memory, we can calculate the reduced costs of a backwards path exactly. For instance, in Instance D, if the starting $ng$-memory of $\bar{P}$ contains 3, we do not include the discount

of travelling into $\mathcal{S}$. If not, we do include the discount. In essence, this means we create at most $2^n$ duplicates of any backwards label, corresponding to the different starting $ng$-memories. Of course, the starting $ng$-memory cannot contain any nodes that the backwards path will visit and remember. In Instance D, we only need $2^3 = 8$ duplicates of the label corresponding to $\bar{P}$, as the starting $ng$-memory can contain nodes 1, 2 or 3, or any combination of those. Dominance can still occur, but $n$ additional inequalities need to be satisfied, due to the extra resources. Note that this method drastically increases the number of stored labels.

**Cost Bounding** A second technique is cost bounding, in which we store an upper and lower bound on the costs in a label. We want to determine the upper and lower bound such that the costs of that backward path segment are always between those bound, regardless of the forward path it is connected to. One way to calculate an upper bound on the costs of a backward path is not including any duals $\gamma_{\mathcal{S}}^{NG}$ or $\gamma_{\mathcal{S}}^{SCC}$ into the cost calculation. As all of these duals are positive, leaving out these duals can only increase the costs, giving an upper bound. A lower bound can be computed in two ways. The first is to apply the duals, $\gamma_{\mathcal{S}}^{NG}$ or $\gamma_{\mathcal{S}}^{SCC}$, as if we were using CCs, so at any arc $(i, j)$ with $i \notin \mathcal{S}$ and $j \in \mathcal{S}$. As this applies the discount too many times, this leaves a lower bound to the reduced costs. The second way is, if the partial backward path is $\bar{P} = (j_0 = n + 1, j_1, \ldots, j_q)$, to calculate the reduced costs of the forward path $\bar{P}' = (0, j_q, \ldots, j_1, j_0)$, subtracting $\bar{c}_{0j_q}$ and adding $\bar{c}_{j_1,n+1}$. Domination of backward paths would occur if the upper bound of one path is smaller than the lower bound of another, as we can then be sure the reduced costs of the first path are always smaller than of the second path. Note that, while we do not need any duplication of labels, this technique could lessen the number of dominated paths, thus decreasing the effectiveness of backward labelling.

So, while it is possible to include the duals of the SCCs or NGCCs into backward labelling, it does reduce the effectiveness of the bi-directional labelling algorithm. However, it is not clear whether bi-directional labelling itself saves time compared to mono-directional labelling. Further research is needed to investigate this.

## 4.4 Feillet et al. (2004) and $ng$-Capacity Cuts

As we mentioned in Section 3.2.2, Feillet et al. (2004) recommend setting an elementarity resource $R^i_E$, $i \in \mathcal{V}$ to 1 if the partial path cannot reach that customer anymore. In a similar fashion, one can also set the $ng$-resource $R^i_{NG}$, $i \in \mathcal{V}$ to 1 if a customer cannot be reached anymore. This technique leads to more domination of labels and thus a faster labelling algorithm.

However, this technique is incompatible with the duals of the NGCCs. Let us consider Instance E in Figure 8. In the figure, square nodes denote the depots and round nodes the customers. The demand of the customers is displayed next to the node. The capacity of a vehicle is $Q = 14$. Let us assume the $ng$-memory of this instance is perfect. That is, $\mathcal{V}_i = \mathcal{V}$ for all $i \in \mathcal{V}$. The figure also highlights the set $\mathcal{S} = \{2, 3\}$.

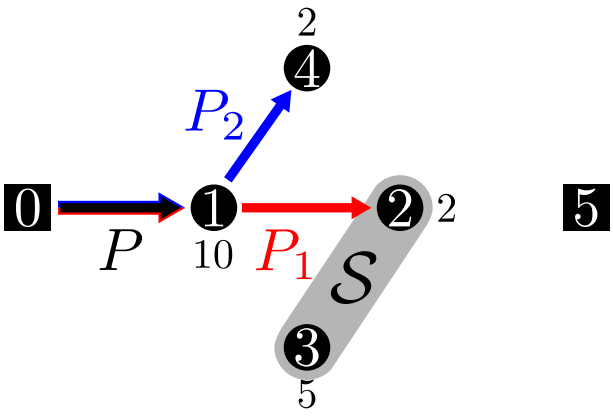



Figure 8: Instance E, a counterexample for the technique of Feillet et al. (2004).

Figure 8 highlights three different partial paths in Instance E: $P = (0, 1)$, $P_1 = (0, 1, 2)$ and $P_2 = (0, 1, 4)$, where $P_1$ and $P_2$ are single-node extensions of $P$. Let us assume that we have included the NGCC for customer subset $\mathcal{S} = \{2, 3\}$. Then, $P_1$ should receive the discount $\gamma_{\mathcal{S}}$ and $P_2$ should not. Note that, for all three paths, it is impossible to travel to node 3, as this makes the demand exceed the vehicle capacity $Q$. If we adhere to the technique of Feillet et al. (2004), node 3 enters the $ng$-memory of path $P$. Thus, for path $P$, the resource $R^3_{NG}$ equals 1. This is problematic, as extending $P$ to $P_1$ now means $P_1$ does not receive the discount $\gamma_{\mathcal{S}}$ (see formula (15)). But, we cannot grant the discount prematurely to $P$, as then $P_2$ unrighteously receives the discount as well.

Thus, it is not obvious how to combine the technique of Feillet et al. (2004) with NGCC

duals robustly. One can probably introduce new resources to combat this, but that would ultimately destroy the motivation behind the technique of Feillet et al. (2004).

## 4.5 Other Cuts

In this section, we discuss other popular families of inequalities for the CVRP or VRPTW and investigate whether we can derive other cuts using similar techniques as we used by converting CCs to NGCCs.

### 4.5.1 $k$-Path Inequalities

The $k$-path inequalities are introduced in Kohl et al. (1999), and are defined similar to the CCs, but with a stronger right-hand-side:

$$\sum_{(i,j)\in\mathcal{A}:\, i\notin\mathcal{S},\, j\in\mathcal{S}} z_{ij} \geq k(\mathcal{S}) \qquad\qquad \forall \mathcal{S} \subseteq \mathcal{V} :\ |\mathcal{S}| \geq 2,$$

where $k(\mathcal{S})$ is defined as the minimum number of vehicles needed to serve the customers in $\mathcal{S}$. Unlike with the CCs, $k(\mathcal{S})$ depends on both the capacity and time windows and is thus stronger than the CCs. As all the derivations done for the CCs to obtain the SCCs and NGCCs do not depend on the right-hand-side of the inequalities, we can do the same derivations to strengthen the $k$-path inequalities. This is beyond the scope of this thesis, however.

### 4.5.2 Clique Inequalities

Clique inequalities (CIs) are valid for any set partitioning problem, and can thus be used to strengthen the RMP (Baldacci et al., 2008). The inequalities exploit the property that certain routes cannot be used together in an integer solution. To be more exact, according to Baldacci et al. (2008), two distinct routes $r_1, r_2 \in \mathcal{R}$ are said to *conflict* if and only if they serve at least one common customer. That is, if and only if $|\{i \in \mathcal{V} :\ a^i_{r_1} \geq 1,\ a^i_{r_2} \geq 1\}| \geq 1$. With this, we can define conflict graph $\mathcal{G}^C = (\mathcal{R}', \mathcal{A}^C)$, where the nodes represent routes and the arcs connect conflicting routes. Note that this graph is undirected, as the conflict relationship is symmetric. We define a *clique* $\mathcal{C}$ in this graph as a subset of routes whose subgraph in $\mathcal{G}^C$ is complete. That is, $\mathcal{C} \subseteq \mathcal{R}'$ and $\mathcal{C} \times \mathcal{C} \subseteq \mathcal{A}^C$. Given such a clique

$\mathcal{C}$, Baldacci et al. (2008) define a clique inequality as

$$\sum_{r \in \mathcal{C}} x_r \leq 1.$$

We denote the dual of a clique inequality, corresponding to $\mathcal{C}$, with $\omega_{\mathcal{C}} \leq 0$. Furthermore, we assume we have included $C$ clique inequalities in our master problem, corresponding to unique clique sets $\mathcal{C}_1, \ldots, \mathcal{C}_C$. We can incorporate these duals into the pricing problem by including the following term in the REF of the cost resource $f_{ij}^{R_C}(\mathbf{u})$:

$$- \sum_{\substack{\mathcal{C}_k \colon |\{r \in \mathcal{C}_k \colon \sum_{i \in \mathcal{V}} a_r^i u^{R_E^i} \geq 1\}| < |\mathcal{C}_k|, \\ |\{r \in \mathcal{C}_k \colon \sum_{i \in \mathcal{V}} a_r^i u^{R_E^i} \geq 1 \textbf{ or } a_r^j = 1\}| = |\mathcal{C}_k|, \\ k \in \{1, \ldots, C\}}} \omega_{\mathcal{C}_k}.$$

In the above formula, we subtract the dual of any clique $\mathcal{C}$ whenever we fulfil two criteria: we must have a conflict with all routes in $\mathcal{C}$ when adding node $j$, and this conflict must not exist without node $j$ added to our current path.

However, the aforementioned addition to the REF actually makes it violate the non-decreasing assumption. After all, the route is punished by a cost increase of $-\omega_{\mathcal{C}} \geq 0$ if it conflicts with the whole clique $\mathcal{C}$ for the first time. Increasing the elementarity resources $R_E^i$, $i \in \mathcal{V}$, can make this first conflict happen, thus increasing the cost. However, increasing these resources further can cause the first conflict to have happened before, lowering the cost again. Thus, as an SPPRC with general REFs (thus not non-decreasing) is exponentially more difficult to solve (Irnich & Desaulniers, 2005), we do not include these inequalities or attempt to adjust them for $ng$-resources.

### 4.5.3 Subset-Row Inequalities

The subset-row inequalities (SRs), as defined in Jepsen, Petersen, Spoorendonk, and Pisinger (2008), are given by

$$\sum_{r \in \mathcal{R}} \left\lfloor \frac{1}{k} \sum_{i \in \mathcal{S}} a_r^i \right\rfloor x_r \leq \left\lfloor \frac{|\mathcal{S}|}{k} \right\rfloor \qquad \forall \mathcal{S} \subseteq \mathcal{V},\ 0 < k \leq |\mathcal{S}|.$$

The SRs state that the number of routes that serve at least $k$ customers in $\mathcal{S}$ should be at most $\left\lfloor \frac{|\mathcal{S}|}{k} \right\rfloor$. The SRs are valid for general set partitioning problems and thus automatically valid for the set partitioning formulation of the CVRP and VRPTW (Jepsen et al., 2008).

One could include the SRs into the pricing problem by subtracting the dual, $\omega_{\mathcal{S}} \leq 0$, every time a multiple of $k$ customers in $\mathcal{S}$ is visited. Note that, if the partial route is currently in $i \in \mathcal{V}$, arc $(i,j) \in \mathcal{A}$ with $j \in \mathcal{S}$ is punished by $-\omega_{\mathcal{S}} \geq 0$ if the partial path has already visited $k-1$ customers in $\mathcal{S}$, but not if $k$ customers are visited. Thus, the resulting REF is not non-decreasing in the resources. Similar to the clique inequalities, we thus do not include these inequalities in our set partitioning problem. Furthermore, if we try to strengthen the SRs, similar to the CCs, we could only lower the left-hand-side of the inequality, which actually makes the SRs weaker.

Baldacci et al. (2011) consider the special case with $|\mathcal{S}| = 3$ and $k = 2$, which they call the SR3 inequalities. They also define a relaxation of the SR3 inequalities, the weak SR3 (WSR3) inequalities:

$$\sum_{r \in \mathcal{R}} \max \left\{ \sum_{(i,j) \in \mathcal{A}:\, i \in \mathcal{S},\, j \in \mathcal{S}} b_r^{ij},\ 1 \right\} x_r \leq 1 \qquad \forall \mathcal{S} \subseteq \mathcal{V},\ |\mathcal{S}| = 3.$$

The WSR3 inequalities state that the number of routes that traverse at least one arc in $\mathcal{S}$ should be at most 1. Note that we cannot include the duals of the WSR3 into our pricing problem; our resources cannot remember whether we have travelled over an arc $(i,j)$ with $i \in \mathcal{S}$ and $j \in \mathcal{S}$.

# 5 Experiments

To answer the main questions of this thesis, we execute the introduced branch-price-and-cut method on several CVRP and VRPTW instances. We conduct the experiments on the A, B and P classes of the Augerat instances (Augerat, 1995). To limit the runtimes of our methods, we will only use the instances where the number of customers $n$ is smaller than or equal to 50.

To test the performance of the NGCCs on the root node of CVRP and VRPTW instances, we execute the algorithm in Figure 2 with six different separation techniques: no cuts at all (NoCuts), separate and add CCs (CCSEP), NGSEP1, NGSEP2, NGSEP3 and NGSEP4. To test the performance of the NGCCs in the branch-price-and-cut setting (Algorithm 2), we compare the same separation techniques, except for NoCuts. We do not include NoCuts here as this increases the runtime of the branch-price-and-cut algorithm by an order of magnitude. Furthermore, if the runtime of the branch-price-and-cut algorithm exceeds one hour (excluding the root node relaxation), we terminate the algorithm and return the best found integer solution, which forms an upper bound on the actual optimal value.

We set our neighbourhoods $\mathcal{V}_i$ to contain the 10 nearest neighbours of $i \in \mathcal{V}$ in terms of costs, including $i$ itself. We choose this number as it is similar to the neighbourhood sizes chosen in Baldacci et al. (2011).

All methods are coded and implemented in C++. The algorithms were executed on a PC running Windows 10 with an Intel core i5-4460 processor @3.2GHz. We solve the linear programs with CPLEX version 12.7.1.

# 6 Results

## 6.1 Root Node Relaxation

Table 2 shows the results of solving the root node of several Augerat instances. The first two columns (Inst. and Sep.) show the instance name and cut separation method used to solve that instance. The third column (Obj.) shows the lower bound obtained by solving the root node using price-and-cut, where **boldface** highlights the highest lower bound obtained by any of the separation methods and an asterisk ($*$) indicates that the solution is integer. The remaining columns show the total running time in seconds (Time), the number of routes generated (#Routes), the number of cuts generated, with the number of active cuts at the solution in parenthesis (#Cuts) and the total time the cut separation took in seconds (Sep. Time). The results for all solved instances are given in Section C.1, Appendix C.

Inspecting Table 2, we can see that, in general, the addition of NGCCs raises the LP lower bound in comparison to CCs. However, the relative gain from introducing CCs compared to no cuts is higher than introducing NGCCs compared to CCs. This indicates that there is a relatively small difference between the NGCCs and CCs, though this depends on the instance and choice of the neighbourhoods. Furthermore, we see that in some instances, such as A-n36-k5 and A-n44-k6, NGSEP3 actually attains a lower lower bound than CCSEP. This is probably due to the fact that CVRPSEP does not have a perfect separation of CCs. Let us, for instance, assume we have a CVRP instance whose current solution violates only a single NGCC, and the corresponding CC is also violated. Let us also assume that the current solution actually $r$-violates around 1000 CCs. Because CVRPSEP cannot find all $r$-violated CCs, there is a high probability that the actually violated NGCC will not be found, which means NGSEP3 terminates. NGSEP4 mitigates this problem of NGSEP3 by first separating with $\kappa = 1$.

Furthermore, Table 2 shows that, in general, NGSEP3 creates more NGCCs of all other separation methods and even has more active NGCCs at its solutions. However, such as in A-n44-k6, more active NGCCs does not automatically mean that the found solution has a higher objective value. We can also observe that the separation time is almost neg-

ligible to the solving time. We can still see that searching for $r$-violated CCs in NGSEP3 and NGSEP4 takes several orders of magnitude more time than in the other separation methods. This is of course caused by the lowered arc flows and the tripling of the number of customers. Also, NGSEP4 generally takes more separation time than NGSEP3, as NGSEP4 iterates over the values of $\kappa$, thus executing more separations. Despite this, the separations of NGSEP3 and NGSEP4 only take a few seconds in total.

Table 2: Root node relaxation results for selected Augerat instances.

| Inst. | Sep. | Obj. | Time | #Routes | #Cuts | Sep. Time |
|---|---|---|---|---|---|---|
| A-n34-k5 | No Cuts | 742.456 | 35.1 | 252 | 0 (0) | 0.0 |
| | CCSEP | 775.000 | 59.7 | 349 | 78 (41) | 0.0 |
| | NGSEP1 | 775.000 | 53.2 | 346 | 88 (48) | 0.0 |
| | NGSEP2 | 775.000 | 66.2 | 365 | 78 (50) | 0.1 |
| | NGSEP3 | **776.100** | 74.5 | 375 | 107 (63) | 1.0 |
| | NGSEP4 | 775.000 | 53.8 | 346 | 88 (48) | 0.5 |
| A-n36-k5 | No Cuts | 774.167 | 358.4 | 379 | 0 (0) | 0.0 |
| | CCSEP | 798.302 | 939.5 | 498 | 83 (32) | 0.0 |
| | NGSEP1 | 798.322 | 893.4 | 544 | 70 (50) | 0.0 |
| | NGSEP2 | 798.314 | 1115.7 | 514 | 83 (53) | 0.1 |
| | NGSEP3 | 795.093 | 1450.4 | 496 | 197 (153) | 1.2 |
| | NGSEP4 | **799.000**∗ | 1114.0 | 556 | 72 (71) | 0.5 |
| A-n44-k6 | No Cuts | 927.107 | 117.8 | 392 | 0 (0) | 0.0 |
| | CCSEP | **936.800** | 342.6 | 539 | 155 (98) | 0.0 |
| | NGSEP1 | **936.800** | 456.4 | 543 | 142 (106) | 0.1 |
| | NGSEP2 | **936.800** | 394.7 | 572 | 155 (118) | 0.1 |
| | NGSEP3 | 934.786 | 457.0 | 552 | 244 (140) | 1.0 |
| | NGSEP4 | **936.800** | 460.4 | 543 | 142 (106) | 1.3 |
| B-n41-k6 | No Cuts | 797.033 | 227.2 | 314 | 0 (0) | 0.0 |
| | CCSEP | 828.600 | 801.4 | 619 | 92 (73) | 0.0 |
| | NGSEP1 | **829.000**∗ | 928.8 | 657 | 82 (82) | 0.0 |
| | NGSEP2 | 829.000 | 858.6 | 683 | 92 (91) | 0.1 |
| | NGSEP3 | 829.000 | 802.5 | 692 | 244 (235) | 0.6 |
| | NGSEP4 | **829.000**∗ | 934.0 | 657 | 82 (82) | 0.2 |
| B-n43-k6 | No Cuts | 699.760 | 759.2 | 368 | 0 (0) | 0.0 |
| | CCSEP | 736.880 | 2192.0 | 569 | 129 (42) | 0.0 |
| | NGSEP1 | 737.486 | 2266.0 | 580 | 143 (69) | 0.0 |
| | NGSEP2 | 737.486 | 2380.6 | 597 | 129 (56) | 0.1 |
| | NGSEP3 | 737.418 | 1645.1 | 535 | 185 (94) | 1.3 |
| | NGSEP4 | **737.493** | 1898.8 | 584 | 145 (71) | 4.3 |

We can also see in Table 2 that the number of routes generated is roughly the same for all separation methods, with the exception of No Cuts. This makes sense, as addi-

tional routes have to be generated in order to accommodate for the added CCs or NGCCs.

While the effectiveness of the different separation methods depends entirely on the instance, we can notice some general trends in lower bound performance. Table 3 shows the number of instances for which the separation technique achieved the highest lower bound, per instance class. The percentage of highest lower bounds is shown in parenthesis.

Table 3: Number and percentage of instances in for which the particular separation technique achieved the highest lower bound, per instance class.

| Class | No Cuts | CCSEP | NGSEP1 | NGSEP2 | NGSEP3 | NGSEP4 |
|---|---|---|---|---|---|---|
| A | 0 (0.0%) | 4 (26.7%) | 8 (53.3%) | 8 (53.3%) | 5 (33.3%) | 13 (86.7%) |
| B | 0 (0.0%) | 5 (38.5%) | 9 (69.2%) | 8 (61.5%) | 7 (53.9%) | 11 (84.6%) |
| P | 2 (15.4%) | 4 (30.8%) | 8 (61.5%) | 9 (69.2%) | 7 (53.9%) | 13 (100.0%) |

As Table 3 shows, NGSEP4 achieves the highest lower bounds of all separation methods, even though it is not guaranteed. This shows that NGSEP4 is a good candidate for separating NGCCs if one wants the highest lower bound. We can also observe that NGSEP1 and NGSEP2 perform similarly and that NGSEP3 performs the worst of the NGCC separation methods.

It may also be of interest to investigate for which values of $\kappa$ the most violated NGCCs are found in NGSEP4. After all, if one wants to limit the number of times CVRPSEP is run, you could repeat the separation until $\kappa$ equals some number smaller than $\max_{r \in \mathcal{R}'} \left\{ \left\lceil \frac{|r|}{2} \right\rceil \right\}$. Table 4 shows the number of violated NGCCs found per value of $\kappa$, for all instances in which violated NGCCs are found by NGSEP4 for any $\kappa > 1$.

Table 4: Number of violated NGCCs found per value of $\kappa$ for NGSEP4.

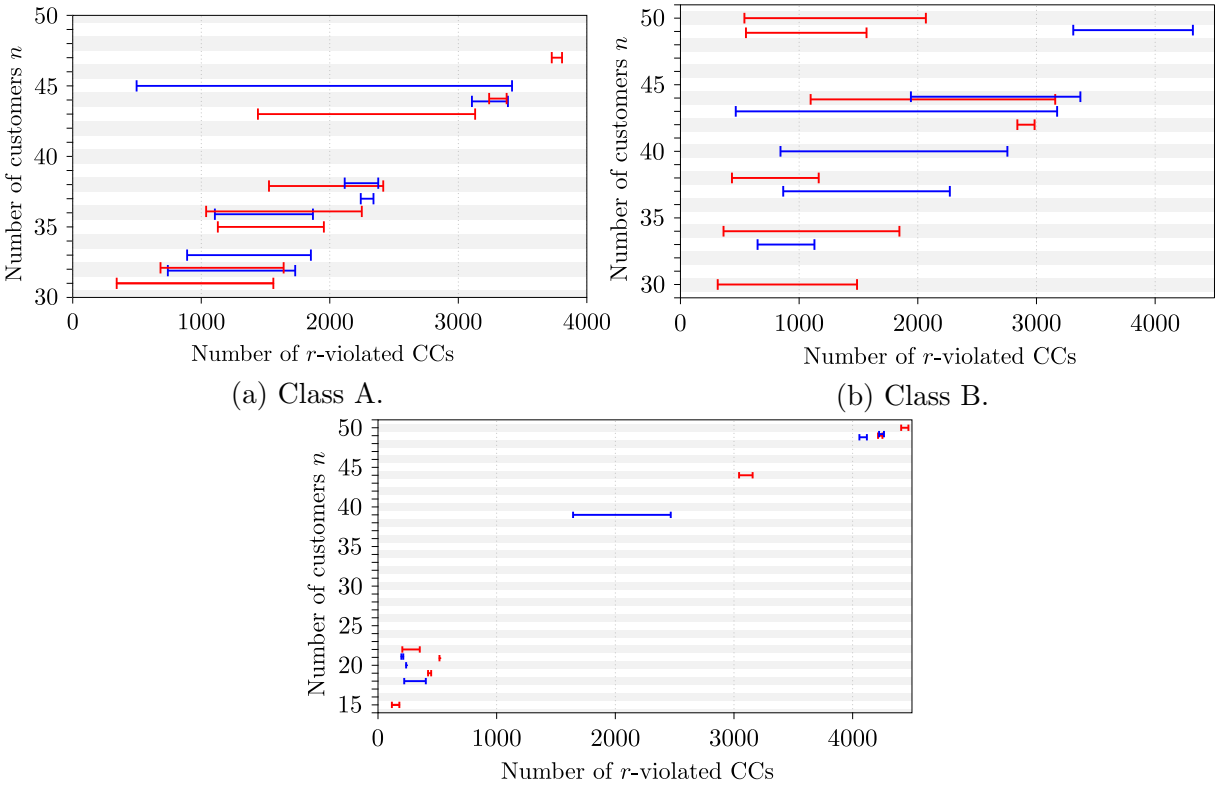
| $\kappa$ | 1 | 2 | 3 | 4 | $> 4$ |
|---|---|---|---|---|---|
| A-n36-k5 | 71 | 1 | — | — | — |
| A-n37-k5 | 68 | 3 | 1 | — | — |
| A-n37-k6 | 37 | 32 | — | — | — |
| A-n39-k5 | 146 | 1 | — | — | — |
| A-n39-k6 | 128 | 1 | — | — | — |
| A-n45-k6 | 96 | 2 | — | — | — |
| A-n45-k7 | 307 | 8 | — | — | — |
| B-n34-k5 | 108 | 2 | — | — | — |

| $\kappa$ | 1 | 2 | 3 | 4 | $> 4$ |
|---|---|---|---|---|---|
| B-n43-k6 | 143 | 2 | — | — | — |
| B-n45-k6 | 39 | 9 | — | — | — |
| B-n50-k8 | 217 | 1 | — | — | — |
| P-n16-k8 | 13 | 2 | — | — | — |
| P-n40-k5 | 67 | 1 | — | — | — |
| P-n50-k7 | 131 | 3 | — | 2 | — |
| P-n50-k8 | 78 | 2 | — | — | — |
| P-n51-k10 | 90 | 1 | — | — | — |

As we can see from Table 4, no violated NGCCs are found with $\kappa > 4$ for any instance.

This means that we can save time on the separation by terminating NGSEP4 if $\kappa = 4$ and no cuts are found. This can save quite some iterations of NGSEP4, as $\max_{r \in \mathcal{R}'} \left\{ \left\lceil \frac{|r|}{2} \right\rceil \right\}$ can be as high as 25 (for B-n34-k5). Of course, Table 4 does not give a guarantee that no cuts will be found for $\kappa > 4$, but it does show that the vast majority of NGCCs are found for relatively low values of $\kappa$.

As we know, NGSEP3 finds a lot of $r$-violated CCs and searches for violated NGCCs among them. However, one may ask how many $r$-violated CCs are found, for instance for memory preallocation. Figures 9a - 9c show the maximum and minimum number of $r$-violated CCs for every solved Augerat instance. Every horizontal line (red or blue) represents one instance. Instances with the same number of customers, for instance A-n33-k5 and A-n33-k6, are ordered by number of vehicles from the bottom to the top. That is, the lower (blue) line is A-n33-k5 and the higher (red) line is A-n33-k6.




(a) Class A.

(b) Class B.

(c) Class P.

Figure 9: Ranges of the number of $r$-violated CCs found for Augerat instances.

Inspecting Figures 9a - 9c, we can see that the number of found $r$-violated CCs does increase with the instance size. Of course, this is just a general trend, and the ranges themselves can be quite volatile. If we also look at the actual number of generated

NGCCs in Table 2, we can conclude that only a small fraction of $r$-violated CCs are actually violated NGCCs, thus further supporting the hypothesis why NGSEP3 sometimes performs worse than CCSEP, NGSEP1 and NGSEP2.

## 6.2 Branch-Price-and-Cut

Table 5 shows the results of the branch-price-and-cut algorithm for some Augerat instances. The full results are given in Section C.2, Appendix C. The first two columns (Inst. and Sep.) show the instance name and cut separation algorithm used to solve the instance. The third column (Obj.) shows the best found integer solution by the branch-price-and-cut algorithm. If the algorithm was terminated (and thus *not* proven to be optimal), a dagger ($^\dagger$) is added to the objective value. **Boldface** highlights the lowest solution value obtained by any of the separation methods. The fourth column (Time) shows the total running time in seconds, where **boldface** indicates the lowest running time if the algorithm was not terminated. The next three columns show the number of routes (#R) generated, the number of cuts (#C) generated and the total separation time (ST) in seconds. The remaining columns show the number of created nodes in the tree (#N), with **boldface** indicating the least number of nodes (if not terminated), the number of nodes that are pruned by bounds (#B) and the number of nodes that are pruned by integer (#Int).

Inspecting Table 5, we can see that adding NGCCs is still beneficial, as this results, in general, in lower upper bounds and shorter runtimes. However, unlike with the root node, one has to choose the NGCC separation method a lot more careful. For instance, NGSEP4 in A-n34-k5 takes more than twice as long as CCSEP and more than 10 times as long as NGSEP3. This is probably caused by the fact that the separation takes longer for NGSEP4 than the other considered separation methods. This does not fully explain the absurdly high runtime of NGSEP4 in A-n34-k5. It could also be caused by the fact that, by coincidence, NGSEP4 branches on an arc which leads to a suboptimal part of the tree. Such a curiosity may not be present in a more balanced branching scheme.

We can also see from Table 5 that adding NGCCs results in fewer explored nodes. Of course, as we know that including NGCCs, compared to CCs, raises the LP lower bound,

nodes are either quicker integral or have a higher lower bound than the best integer solution. However, it is not as clear-cut which of the NGCC separation methods achieves the highest node reduction.

Table 5: Branch-price-and-cut results for selected Augerat instances.

| Ins. | Sep. | Obj. | Time | #R | #C | ST | #N | #B | #Int |
|---|---|---|---|---|---|---|---|---|---|
| A-n34-k5 | CCSEP | **778.000** | 1174.5 | 4584 | 184 | 4.6 | 621 | 235 | 76 |
| | NGSEP1 | **778.000** | 520.3 | 2134 | 132 | 1.8 | 249 | 93 | 32 |
| | NGSEP2 | **778.000** | 821.7 | 2657 | 139 | 2.4 | 301 | 121 | 30 |
| | NGSEP3 | **778.000** | **251.3** | 1093 | 121 | 17.3 | **65** | 27 | 6 |
| | NGSEP4 | **778.000** | 2521.1 | 3984 | 175 | 614.4 | 633 | 245 | 72 |
| A-n36-k5 | CCSEP | 813.000$^{\dagger}$ | 4524.4 | 5847 | 188 | 3.8 | 499 | 226 | 19 |
| | NGSEP1 | **799.000** | 1267.6 | 714 | 73 | 0.1 | 11 | 3 | 3 |
| | NGSEP2 | **799.000** | 3240.5 | 2250 | 125 | 0.9 | 95 | 38 | 10 |
| | NGSEP3 | 807.000$^{\dagger}$ | 5022.0 | 3528 | 270 | 58.0 | 257 | 112 | 14 |
| | NGSEP4 | **799.000** | **1114.2** | 556 | 72 | 0.0 | **1** | 0 | 1 |
| B-n38-k6 | CCSEP | **805.000** | 412.9 | 1222 | 64 | 0.2 | 27 | 8 | 6 |
| | NGSEP1 | **805.000** | **391.5** | 699 | 53 | 0.0 | **9** | 0 | 5 |
| | NGSEP2 | **805.000** | 419.1 | 890 | 60 | 0.2 | 23 | 4 | 8 |
| | NGSEP3 | **805.000** | 415.5 | 872 | 65 | 1.7 | 15 | 1 | 7 |
| | NGSEP4 | **805.000** | 395.2 | 699 | 53 | 3.5 | **9** | 0 | 5 |
| B-n43-k6 | CCSEP | 749.000$^{\dagger}$ | 5781.2 | 6124 | 260 | 5.8 | 781 | 352 | 26 |
| | NGSEP1 | 745.000$^{\dagger}$ | 5827.0 | 3517 | 194 | 2.8 | 385 | 167 | 18 |
| | NGSEP2 | **742.000**$^{\dagger}$ | 5956.4 | 3694 | 172 | 1.7 | 229 | 84 | 27 |
| | NGSEP3 | 743.000$^{\dagger}$ | 5275.2 | 2623 | 230 | 95.0 | 241 | 89 | 26 |
| | NGSEP4 | 743.000$^{\dagger}$ | 5465.2 | 2587 | 200 | 756.3 | 215 | 76 | 26 |
| P-n40-k5 | CCSEP | **458.000** | 241.4 | 2009 | 122 | 0.8 | 87 | 29 | 15 |
| | NGSEP1 | **458.000** | 227.0 | 1256 | 113 | 0.5 | 51 | 15 | 11 |
| | NGSEP2 | **458.000** | **178.9** | 944 | 88 | 0.3 | **33** | 9 | 8 |
| | NGSEP3 | **458.000** | 290.1 | 1015 | 264 | 6.0 | 35 | 12 | 6 |
| | NGSEP4 | **458.000** | 324.9 | 1263 | 117 | 70.0 | 59 | 20 | 10 |

Just as with the root node relaxation, we may be able to detect general performance patterns by inspecting the number of times a separation technique performs "best". To be more precise, Table 6 shows the number of instances for which some separation technique achieved the lowest upper bound (Obj.), lowest running time (Time, if not terminated) or lowest number of nodes (#N, if not terminated).

Table 6: Number of instances in for which the particular separation technique achieved the lowest upper bound, running time or number of nodes, per instance class.

| Criterium | Class | CCSEP | NGSEP1 | NGSEP2 | NGSEP3 | NGSEP4 |
|---|---|---|---|---|---|---|
| Obj. | A | 10 (66.7%) | 11 (73.3%) | 13 (86.7%) | 12 (80.0%) | 11 (73.3%) |
| | B | 10 (76.9%) | 11 (84.6%) | 13 (100.0%) | 9 (69.2%) | 10 (76.9%) |
| | P | 11 (84.6%) | 10 (76.9%) | 8 (61.5%) | 10 (76.9%) | 10 (76.9%) |
| Time | A | 2 (13.3%) | 2 (13.3%) | 3 (20.0%) | 3 (20.0%) | 1 (6.7%) |
| | B | 4 (30.8%) | 4 (30.8%) | 1 (7.7%) | 1 (7.7%) | 0 (0.0%) |
| | P | 2 (15.4%) | 2 (15.4%) | 2 (15.4%) | 2 (15.4%) | 0 (0.0%) |
| #N | A | 1 (6.7%) | 3 (20.0%) | 4 (26.7%) | 4 (26.7%) | 4 (26.7%) |
| | B | 5 (38.5%) | 9 (69.2%) | 7 (53.8%) | 6 (46.2%) | 9 (69.2%) |
| | P | 4 (30.8%) | 5 (38.5%) | 7 (53.8%) | 6 (46.2%) | 6 (46.2%) |

As we can see from Table 6, NGSEP2 performs best on A and B instances in terms of objectives. However, the margins are too small to notice any strong patterns. The same holds for time. With the number of nodes, we can notice that, indeed, the NGCC separation methods create less nodes on average. That the creation of less nodes does not lead to better runtimes may be explained by the fact that the NGCC separation methods have higher solving times per node. Another explanation might be that we have to recalculate the coefficients of the NGCCs whenever we branch on $z_{ij} = 1$. As the number of cuts and routes increases with time, the time to adjust the NGCCs increases as well.

Comparing the results for the root node relaxation and the full branch-price-and-cut, it is not as clear-cut which separation technique performs best. It may be best to combine several separation techniques to exploit their individual strengths. For instance, one could use NGSEP4 on the root node, which has an excellent overall performance, and NGSEP1, NGSEP2 or even CCSEP in the branching tree.

# 7 Conclusion

This thesis introduced a new type of robust valid inequalities for the CVRP and VRPTW: the $ng$-capacity cuts. The inequalities are a lifting of the ordinary capacity cuts and can be included robustly into a branch-price-and-cut scheme using the $ng$-route relaxation. We have tested the performance of the NGCCs in a generic brach-price-and-cut setting against using CCs and no cuts at all. To separate NGCCs, we devised four different separation algorithms. We have tested the different separation methods on Augerat instances (Augerat, 1995). We focused on the performance of the NGCCs in the root node relaxation of the instances and the full branch-price-and-cut algorithm.

We have found that including NGCCs raises the LP lower bound in comparison to CCs. In some instances, introducing NGCCs may even lead to an integer (optimal) solution. NGSEP4 produces in general the highest lower bounds, at the cost of a longer separation time. However, for the root node, this separation time is negligible. NGSEP3 performs sometimes worse than CCSEP, which can be explained by the fact that reducing the flow with the maximum factor may lead to CVRPSEP missing violated NGCCs. NGSEP4 mitigates this problem by first separating $r$-violated CCs with a low arc flow reduction factor. The other methods, NGSEP1 and NGSEP2, perform roughly the same and have a shorter separation time than NGSEP3 or NGSEP4, while having only slightly worse performance than NGSEP4.

For NGSEP4 itself, we can actually save separation time by limiting the maximum value of $\kappa$. We found that, in the considered instances, no violated NGCCs were found for $\kappa > 4$. The relation between the number of $r$-violated CCs found by CVRPSEP and instance size was clear: the more customers a problem contains, the more $r$-violated CCs are found. For instance, 30 customers means around 1000 $r$-violated CCs while 50 customers already gives us about 4000 $r$-violated CCs per separation.

While NGSEP4 performed best for solving the root node relaxation, no separation method performed clearly best in a branch-price-and-cut setting. We did find that the inclusion of NGCCs reduced the number of branching nodes needed to obtain an optimal solution, compared to using CCs. However, as the solving of the individual nodes for the NGCCs

took longer and we recalculated the NGCCs if we branched on $z_{ij} = 1$, no clear-cut time saving was achieved. Furthermore, for NGSEP4, cut separation now took a significant amount of time compared to the other methods. For many instances, the solving time took over an hour and we had to terminate the algorithm. Comparing the best-found upper bound between the separation methods, we found that NGSEP1 and NGSEP2 performed best on class A and B instances. However, the margins are too small to notice a general pattern. We hypothesise that using a different separation method in the root node and branching tree may lead to shorter running times. For instance, using NGSEP4 in the root node and NGSEP1, NGSEP2 or CCSEP in the tree.

Of course, there are still many unanswered questions regarding the NGCCs. For instance, it may be of interest to evaluate the performance of the NGCCs in a more state-of-the-art branch-price-and-cut algorithm, such as the one in Pecin et al. (2017). Furthermore, following our suggestion above, one could investigate which combination of separation techniques in the root node and branching tree yields the best results. Another idea is to investigate how to choose the neighbourhoods $\mathcal{V}_i$, $i \in \mathcal{V}$, as these neighbourhoods impact both the performance of the pricing algorithm and the NGCCs. Also, as we deduced that we can strengthen the $k$-path inequalities in Section 4.5, one could research the performance of those cuts, especially in VRPTW instances. As we mentioned in Section 4.3, more research is needed to evaluate the impact of including NGCC duals in a bi-directional labelling algorithm, together with the proposed fixes.

# References


Augerat, P. (1995). *Approche polyèdrale du problème de tournées de véhicules* (Unpublished doctoral dissertation). Institut National Polytechnique de Grenoble-INPG.

Augerat, P., Belenguer, J. M., Benavent, E., Corberán, A., Naddef, D., & Rinaldi, G. (1998). Computational results with a branch-and-cut code for the capacitated vehicle routing problem.

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

Baldacci, R., Hadjiconstantinou, E., & Mingozzi, A. (2004). An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, *52*(5), 723–738.

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

Balinski, M. L., & Quandt, R. E. (1964). On an integer program for a delivery problem. *Operations Research*, *12*(2), 300–304.

Bard, J. F., Kontoravdis, G., & Yu, G. (2002). A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, *36*(2), 250–269.

Beasley, J. E., & Christofides, N. (1989). An algorithm for the resource constrained shortest path problem. *Networks*, *19*(4), 379–394.

Bell, W. J., Dalberto, L. M., Fisher, M. L., Greenfield, A. J., Jaikumar, R., Kedia, P., ... Prutzman, P. J. (1983). Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. *Interfaces*, *13*(6), 4–23.

Carter, M. W., Farvolden, J. M., Laporte, G., & Xu, J. (1996). Solving an integrated logistics problem arising in grocery distribution. *INFOR: Information Systems and Operational Research*, *34*(4), 290–306.

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, *6*(1), 80–91.

de Aragao, M. P., & Uchoa, E. (2003). Integer program reformulation for robust branch-and-cut-and-price algorithms. In *Mathematical program in rio: a conference in honour of nelson maculan* (pp. 56–61).

Desaulniers, G., Lessard, F., & Hadjar, A. (2008). Tabu search, partial elementarity, and

generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, *42*(3), 387–404.

Desrochers, M., Desrosiers, J., & Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, *40*(2), 342–354.

Desrosiers, J., & Lübbecke, M. E. (2005). A primer in column generation. In *Column generation* (pp. 1–32). Springer.

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

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

Fisher, M. L., & Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks*, *11*(2), 109–124.

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

Irnich, S., & Desaulniers, G. (2005). Shortest path problems with resource constraints. In *Column generation* (pp. 33–65). Springer.

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

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

Kallehauge, B., Larsen, J., & Madsen, O. (2006). Lagrangean duality and non-differentiable optimization applied on routing with time windows. *Computers and Operations Research*, *33*(5), 1464–1487.

Kohl, N. (1995). Exact methods for time constrained routing and related scheduling problems.

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

Laporte, G., Nobert, Y., & Desrochers, M. (1985). Optimal routing under capacity and distance restrictions. *Operations Research*, *33*(5), 1050–1073.

Lenstra, J. K., & Kan, A. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, *11*(2), 221–227.

Lübbecke, M. E. (2010). Column generation. *Wiley Encyclopedia of Operations Research and Management Science, John Wiley and Sons, Chichester, UK*.

Lysgaard, J. (2003). Cvrpsep: A package of separation routines for the capacitated vehicle routing problem.

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

Martinelli, R., Pecin, D., & Poggi, M. (2014). Efficient elementary and restricted non-elementary route pricing. *European Journal of Operational Research*, *239*(1), 102–111.

Naddef, D., & Rinaldi, G. (2002). Branch-and-cut algorithms for the capacitated vrp. In *The vehicle routing problem* (pp. 53–84). SIAM.

Nemhauser, G. L., & Park, S. (1991). A polyhedral approach to edge coloring. *Operations Research Letters*, *10*(6), 315–322.

Padberg, M., & Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, *33*(1), 60–100.

Pecin, D., Pessoa, A., Poggi, M., & Uchoa, E. (2017). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, *9*(1), 61–100.

Rao, M., & Zionts, S. (1968). Allocation of transportation units to alternative trips–A column generation scheme with out-of-kilter subproblems. *Operations Research*, *16*(1), 52–63.

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

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

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with

time window constraints. *Operations Research*, *35*(2), 254–265.

# A Proofs

**Lemma 1.** *Recursion scheme* (6) *is equivalent to the one stated in Baldacci et al. (2011).*

*Proof.* Let us define a path $P = (i_0 = 0, i_1, \ldots, i_p)$. Baldacci et al. (2011) define $\Pi(P)$ as

$$\Pi(P) = \left\{ i_k : i_k \in \bigcap_{s=k+1}^{p} \mathcal{V}_{i_s},\ k \in \{1, \ldots, p-1\} \right\} \cup \{i_p\}. \tag{16}$$

We will show that $\Pi(P) = \Pi_p(P)$ from (6). Using our recursive scheme (6), we can investigate the event $i_k \in \Pi_p(P)$, for some $k \in \{1, \ldots, p-1\}$. We get

$$\begin{aligned}
i_k \in \Pi_p(P) &\iff i_k \in \Big( \{i_p\} \cup \big( \Pi_{p-1}(P) \cap \mathcal{V}_{i_p} \big) \Big) \\
&\iff i_k \in \big( \Pi_{p-1}(P) \cap \mathcal{V}_{i_p} \big) \\
&\iff i_k \in \Pi_{p-1}(P) \quad \wedge \quad i_k \in \mathcal{V}_{i_p} \\
&\iff i_k \in \Big( \{i_{p-1}\} \cup \big( \Pi_{p-2}(P) \cap \mathcal{V}_{i_{p-1}} \big) \Big) \quad \wedge \quad i_k \in \mathcal{V}_{i_p} \\
&\ \ \vdots \\
&\iff i_k \in \Big( \{i_k\} \cup \big( \Pi_{k-1}(P) \cap \mathcal{V}_{i_k} \big) \Big) \ \wedge \ i_k \in \mathcal{V}_{i_{k+1}} \ \wedge \ i_k \in \mathcal{V}_{i_{k+2}} \ \wedge \ \ldots \ \wedge \ i_k \in \mathcal{V}_{i_p} \\
&\iff i_k \in \mathcal{V}_{i_{k+1}} \ \wedge \ i_k \in \mathcal{V}_{i_{k+2}} \ \wedge \ \ldots \ \wedge \ i_k \in \mathcal{V}_{i_p} \\
&\iff i_k \in \bigcap_{s=k+1}^{p} \mathcal{V}_{i_s} \\
&\iff i_k \in \Pi(P).
\end{aligned}$$

Furthermore, as both $i_p \in \Pi_p(P)$ and $i_p \in \Pi(P)$ trivially, we have that $\Pi_p(P) = \Pi(P)$, which completes the proof. □

**Lemma 2.** *REFs* (7) *are non-decreasing.*

*Proof.* Let us consider $f_{ij}^{R_{NG}^v}$ for some $(i,j) \in \mathcal{A}$ and $v \in \mathcal{V}$. If $v \notin \mathcal{V}_j$, then $f_{ij}^{R_{NG}^v}(\mathbf{u}) = 0$ for all resource vectors $\mathbf{u}$ and is thus non-decreasing.

If $v \in \mathcal{V}_j$ and $j \neq v$, then $f_{ij}^{R_{NG}^v}(\mathbf{u}) = u^{R_{NG}^v}$. For any pair of resource vectors $\mathbf{u}, \mathbf{w}$ with $\mathbf{u} \leq \mathbf{w}$, we have $u^{R_{NG}^v} \leq w^{R_{NG}^v}$ and thus $f_{ij}^{R_{NG}^v}(\mathbf{u}) \leq f_{ij}^{R_{NG}^v}(\mathbf{w})$. So, it is non-decreasing.

A similar argument holds for the case $j = v$, as $f_{ij}^{R_{NG}^v}(\mathbf{u}) = u^{R_{NG}^v} + 1$. This thus concludes

the proof. □

**Lemma 3.** *For all $\mathcal{S} \subseteq \mathcal{V}$, $|\mathcal{S}| \geq 2$, for all $r \in \mathcal{R}$ and for all $\mathcal{V}_i \subseteq \mathcal{V}$, $i \in \mathcal{V}_i$, $\forall i \in \mathcal{V}$, it holds that $\zeta_r^{SCC}(\mathcal{S}) \leq \zeta_r^{NG}(\mathcal{S}) \leq \zeta_r^{CC}(\mathcal{S})$.*

*Proof.* Let us assume route $r$ equals $(i_0 = 0, i_1, \ldots, i_{|r|}, i_{|r|+1} = n+1)$. We know that

$$\zeta_r^{NG}(\mathcal{S}) = \left|\left\{k : i_k \in \mathcal{S},\, \Pi_{k-1}(r) \cap \mathcal{S} = \emptyset,\, k \in \{1, \ldots, |r|\}\right\}\right|. \tag{13}$$

Furthermore, we can rewrite $\zeta_r^{CC}(\mathcal{S})$ as

$$\begin{aligned}
\zeta_r^{CC}(\mathcal{S}) &= \sum_{(i,j)\in\mathcal{A}} \beta_{\mathcal{S}}^{ij} b_r^{ij} \\
&= \sum_{(i,j)\in\mathcal{A}:\, i\notin\mathcal{S},\, j\in\mathcal{S}} b_r^{ij} \\
&= \left|\left\{k : i_k \in \mathcal{S},\, i_{k-1} \notin \mathcal{S},\, k \in \{1, \ldots, |r|\}\right\}\right| \\
&= \left|\left\{k : i_k \in \mathcal{S},\, \{i_{k-1}\} \cap \mathcal{S} = \emptyset,\, k \in \{1, \ldots, |r|\}\right\}\right|. \qquad (17)
\end{aligned}$$

Because it always holds that $i_{k-1} \in \Pi_{k-1}(r)$, it holds that $\{i_{k-1}\} \subseteq \Pi_{k-1}(r)$, which implies the following relation:

$$\Pi_{k-1}(r) \cap \mathcal{S} = \emptyset \implies \{i_{k-1}\} \cap \mathcal{S} = \emptyset.$$

This, in turn, leads to

$$\forall k \in \{1, \ldots, |r|\} : i_k \in \mathcal{S},\, \Pi_{k-1}(r) \cap \mathcal{S} = \emptyset \implies i_k \in \mathcal{S},\, \{i_{k-1}\} \cap \mathcal{S} = \emptyset,$$

which implies

$$\begin{aligned}
&\left\{k : i_k \in \mathcal{S},\, \Pi_{k-1}(r) \cap \mathcal{S} = \emptyset,\, k \in \{1, \ldots, |r|\}\right\} \\
&\qquad \subseteq \left\{k : i_k \in \mathcal{S},\, \{i_{k-1}\} \cap \mathcal{S} = \emptyset,\, k \in \{1, \ldots, |r|\}\right\} \\
\implies &\left|\left\{k : i_k \in \mathcal{S},\, \Pi_{k-1}(r) \cap \mathcal{S} = \emptyset,\, k \in \{1, \ldots, |r|\}\right\}\right| \\
&\qquad \leq \left|\left\{k : i_k \in \mathcal{S},\, \{i_{k-1}\} \cap \mathcal{S} = \emptyset,\, k \in \{1, \ldots, |r|\}\right\}\right| \\
\implies &\zeta_r^{NG}(\mathcal{S}) \leq \zeta_r^{CC}(\mathcal{S}).
\end{aligned}$$

Using a similar technique, we can rewrite $\zeta_r^{SCC}(\mathcal{S})$ as

$$\begin{aligned}\zeta_r^{SCC}(\mathcal{S}) &= \min\{1,\ \zeta_r^{CC}(\mathcal{S})\} \\ &= \left|\left\{k:\ i_k \in \mathcal{S},\ \{i_1,\ldots,i_{k-1}\} \cap \mathcal{S} = \emptyset,\ k \in \{1,\ldots,|r|\}\right\}\right|. \end{aligned} \tag{18}$$

Because it always holds that $\Pi_k(r) \subseteq \{i_1,\ldots,i_k\}$ for all $k \in \{0,\ldots,|r|\}$, we get

$$\begin{aligned} \{i_1,\ldots,i_{k-1}\} \cap \mathcal{S} = \emptyset \implies \Pi_{k-1}(r) \cap \mathcal{S} = \emptyset, \\ \forall k \in \{1,\ldots,|r|\}:\ i_k \in \mathcal{S},\ \{i_1,\ldots,i_{k-1}\} \cap \mathcal{S} = \emptyset \implies i_k \in \mathcal{S},\ \Pi_{k-1}(r) \cap \mathcal{S} = \emptyset, \end{aligned}$$

which gives

$$\begin{aligned} &\left\{k:\ i_k \in \mathcal{S},\ \{i_1,\ldots,i_{k-1}\} \cap \mathcal{S} = \emptyset,\ k \in \{1,\ldots,|r|\}\right\} \\ &\qquad \subseteq \left\{k:\ i_k \in \mathcal{S},\ \Pi_{k-1}(r) \cap \mathcal{S} = \emptyset,\ k \in \{1,\ldots,|r|\}\right\} \\ \implies &\left|\left\{k:\ i_k \in \mathcal{S},\ \{i_1,\ldots,i_{k-1}\} \cap \mathcal{S} = \emptyset,\ k \in \{1,\ldots,|r|\}\right\}\right| \\ &\qquad \leq \left|\left\{k:\ i_k \in \mathcal{S},\ \Pi_{k-1}(r) \cap \mathcal{S} = \emptyset,\ k \in \{1,\ldots,|r|\}\right\}\right| \\ \implies &\zeta_r^{SCC}(\mathcal{S}) \leq \zeta_r^{NG}(\mathcal{S}). \end{aligned}$$

This proves that

$$\zeta_r^{SCC}(\mathcal{S}) \leq \zeta_r^{NG}(\mathcal{S}) \leq \zeta_r^{CC}(\mathcal{S})$$

for any route $r \in \mathcal{R}$, for any $\mathcal{S} \subseteq \mathcal{V}:\ |\mathcal{S}| \geq 2$ and for any choice of neighbourhoods $\mathcal{V}_i,\ i \in \mathcal{V}$.

Now, let us assume $\mathcal{V}_i = \{i\}$ for all $i \in \mathcal{V}$. Applying equation (6) shows us that $\Pi_k(r) = \{i_k\}$, for all $k \in \{1,\ldots,|r|\}$. Combining (13) and (17) gives us that $\zeta_r^{CC}(\mathcal{S}) = \zeta_r^{NG}(\mathcal{S})$. In other words, this shows that the CCs and NGCCs are equivalent in this case.

Now, let us assume $\mathcal{V}_i = \mathcal{V}$ for all $i \in \mathcal{V}$. Applying equation (6) shows us that $\Pi_k(r) = \{i_1,\ldots,i_k\}$, for all $k \in \{1,\ldots,|r|\}$. Combining (13) and (18) gives us that $\zeta_r^{SCC}(\mathcal{S}) =$

$\zeta_r^{NG}(\mathcal{S})$. In other words, this shows that the SCCs and NGCCs are equivalent in this case, which concludes this proof.

□

**Proposition 1.** *ng-feasibility is symmetric, whereas ng-memory itself is not.*

*Proof.* We first prove that *ng*-feasibility of a path is symmetric. Let us consider the forward path $P = (i_0 = 0, i_1, i_2, \ldots, i_p)$ and the equivalent backward path $\bar{P} = (i_{p+1} = n+1, i_p, i_{p-1}, \ldots, i_1)$. Note that both $P$ and $\bar{P}$ travel over the exact same arcs (except for the arcs to the depots). We define the forward *ng*-memory $\Pi_k(P)$ of the path $P$ as in (6). The backward *ng*-memory $\bar{\Pi}_k(\bar{P})$ at node $i_k$ is defined as in Baldacci et al. (2011):

$$\bar{\Pi}_k(\bar{P}) = \begin{cases} \emptyset & \text{, if } k = p+1 \\ \left(\bar{\Pi}_{k+1}(\bar{P}) \cap \mathcal{V}_{i_k}\right) \cup \{i_k\} & \text{, if } k \in \{1, \ldots, p\} \end{cases}.$$

Note that one can interpret this backward *ng*-memory as the forward *ng*-memory of the reversed path $\bar{P}' = (0, i_p, i_{p-1}, \ldots, i_1)$ so that $\bar{\Pi}_k(\bar{P}) = \Pi_{p+1-k}(\bar{P}')$ for $k \in \{1, \ldots, p, p+1\}$.

The forward *ng*-path $P$ is *ng*-feasible if and only if $i_k \notin \Pi_{k-1}(P)$ for all $k \in \{1, \ldots, p\}$, whereas $\bar{P}$ is *ng*-feasible if and only if $i_k \notin \bar{\Pi}_{k+1}(\bar{P})$ for all $k \in \{1, \ldots, p\}$. Let us consider some $l \in \{1, \ldots, p\}$ such that $i_l$ is *unique* in $P$ and $\bar{P}$. That is, there is no $k \in \{1, \ldots, p\} \setminus \{l\}$ such that $i_k = i_l$. If a node $i_l$ is indeed unique, then it is always true that $i_l \notin \Pi_{l-1}(P)$ and $i_l \notin \bar{\Pi}_{l+1}(\bar{P})$, as $\Pi_k(P) \subseteq \{i_1, \ldots, i_k\}$ and $\bar{\Pi}_k(\bar{P}) \subseteq \{i_{k+1}, \ldots, i_p\}$ for all $k \in \{1, \ldots, p\}$. So, unique nodes in a path do not determine the *ng*-feasibility of paths.

Instead, let us consider some $l_1, l_2 \in \{1, \ldots, p\}$ with $l_1 < l_2$ and $i_{l_1} = i_{l_2} = i$. So, both $P$ and $\bar{P}$ visit $i$ at least twice. Furthermore, assume without loss of generality that $i_k \neq i$ for all $k \in \{l_1 + 1, \ldots, l_2 - 1\}$. Then $P$ is not *ng*-feasible if $i = i_{l_2} \in \Pi_{l_2-1}(P)$. Using the definition of *ng*-memory and the fact that $i = i_{l_1} \in \Pi_{l_1}(P)$, we get

$$i \in \Pi_{l_2-1}(P) \iff i \in \mathcal{V}_{i_{l_1+1}} \cap \mathcal{V}_{i_{l_1+2}} \cap \ldots \cap \mathcal{V}_{i_{l_2-1}}.$$

We also have that $\bar{P}$ is not *ng*-feasible if $i = i_{l_1} \in \bar{\Pi}_{l_1+1}(\bar{P})$. Following the same logic as before, we get

$$i \in \bar{\Pi}_{l_1+1}(\bar{P}) \iff \mathcal{V}_{i_{l_1+1}} \cap \mathcal{V}_{i_{l_1+2}} \cap \ldots \cap \mathcal{V}_{i_{l_2-1}}.$$

So, it follows that $i \in \Pi_{l_2-1}(P) \iff i \in \bar{\Pi}_{l_1+1}(\bar{P})$. As we can repeat this argument for any node that is visited by $P$ and $\bar{P}$ more than once, we can conclude that $ng$-feasibility of $P$ implies $ng$-feasibility of $\bar{P}$ and vice versa. In other words, $ng$-feasibility is indeed symmetric.

Next, we show that $ng$-memory is not symmetric. In other words, we show that $i \in \Pi_k(P)$ does not imply $i \in \bar{\Pi}_k(\bar{P})$ and vice versa, except when $i = i_k$, for any $k \in \{1, \ldots, p\}$. It is obvious that $i_k \in \Pi_k(P)$ and $i_k \in \bar{\Pi}_k(\bar{P})$. Consider any $i \in \Pi_k(P)$ such that $i \neq i_k$. Note that it must hold that $i \in \{i_1, \ldots, i_{k-1}\}$, as per definition $\Pi_k(P) \subseteq \{i_1, \ldots, i_k\}$. We also know that $\bar{\Pi}_k(\bar{P}) \subseteq \{i_{k+1}, \ldots, i_p\}$. So, the presence of $i$ in $\Pi_k(P)$ does not imply it being in $\bar{\Pi}_k(\bar{P})$. In fact, if $i \in \bar{\Pi}_k(\bar{P})$, more conditions should be satisfied. Reversing this argument also shows that $i \in \bar{\Pi}_k(\bar{P})$ does not imply $i \in \Pi_k(P)$. Following an example like Instance C also shows this fact.

□

# B Additional Example Instances

## B.1 Separation Methods NGSEP1 versus NGSEP2

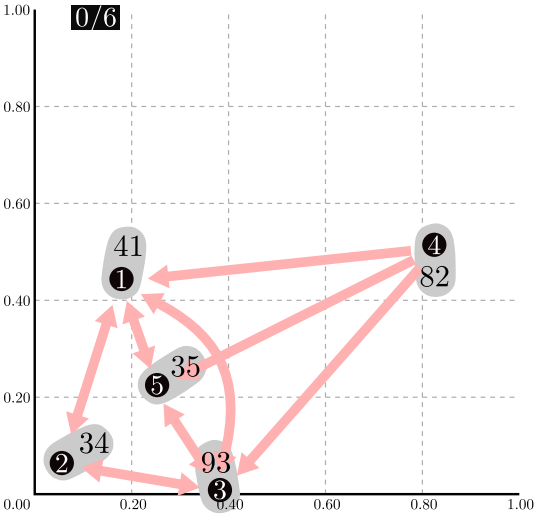



Figure 10: Instance F.

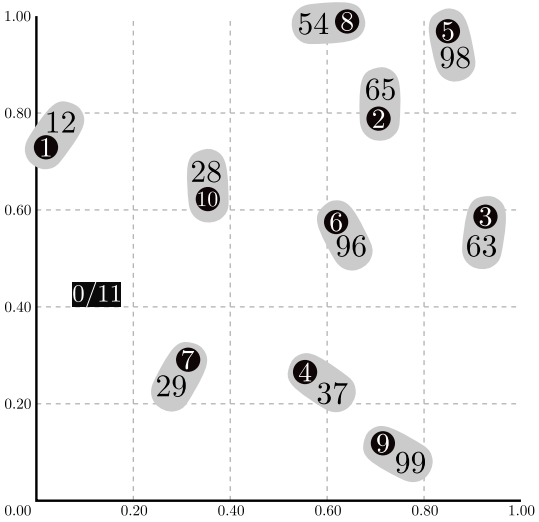



Figure 11: Instance G.

This section shows two instances, one in which NGSEP2 obtains a higher objective value than NGSEP1 (Figure 10, Instance F), and one where the opposite happens (Figure 11, Instance G). Figures 10 and 11 show these instances, in which the five customers are denoted with round nodes and the depot is shown as a square node. Note that in these instances, the starting and ending depot are located at the same position. The nodes are located in Cartesian space, where costs are equal to 100 times the Euclidean distance, rounded down. The capacity of a vehicle equals 150 and customer demand is denoted next to a node. For instance F, the *ng*-memory of a customer includes itself and the 3 nearest customers, denoted with red arrows in the figures. For instance G, the *ng*-memory is perfect. All the pricing problems will be solved exactly, that is, we add the route with the most negative reduced costs.

**Instance F** Let us examine Instance F. Table 7 shows the events that happen when solving the root node using NGSEP1 and NGSEP2. Looking at Table 7, we can see that, up to step 11, the two methods perform identical. It should be noted that there is no difference yet in the NGCC and CC in step 9, as well as the NGCC and CC in step 11. This means that NGSEP2 also has an objective of 501.286 prior to executing step 12. The reason why NGSEP2 does add route $(0, 1, 5, 2, 6)$ and NGSEP1 does not, is because

route $(0, 1, 5, 2, 6)$ receives the discount of CC $\{1, 2, 3\}$ twice, but receives the discount of the NGCC only once. This means route $(0, 1, 5, 2, 6)$ has a lower reduced cost in NGSEP2 than in NGSEP1. In fact, the reduced costs in NGSEP1 are positive and negative in NGSEP2. Because of the extra added route in NGSEP2, another CC is added, which is not found in NGSEP1, resulting in a higher objective of NGSEP2.

Table 7: Executed steps in solving Instance F.

| Step | NGSEP1 | NGSEP2 |
|---|---|---|
| 1 | Add route $(0, 1, 2, 5, 6)$ | Add route $(0, 1, 2, 5, 6)$ |
| 2 | Add route $(0, 2, 4, 2, 6)$ | Add route $(0, 2, 4, 2, 6)$ |
| 3 | Add route $(0, 1, 3, 6)$ | Add route $(0, 1, 3, 6)$ |
| 4 | Add route $(0, 3, 5, 6)$ | Add route $(0, 3, 5, 6)$ |
| 5 | Add route $(0, 2, 3, 6)$ | Add route $(0, 2, 3, 6)$ |
| 6 | Add route $(0, 1, 4, 6)$ | Add route $(0, 1, 4, 6)$ |
| 7 | Add route $(0, 4, 5, 6)$ | Add route $(0, 4, 5, 6)$ |
| 8 | LP solved, objective 491 | LP solved, objective 491 |
| 9 | Add NGCC $\{2, 3, 5\}$ | Add CC $\{2, 3, 5\}$ |
| 10 | LP solved, objective 498.667 | LP solved, objective 498.667 |
| 11 | Add NGCC $\{1, 2, 3\}$ | Add CC $\{1, 2, 3\}$ |
| 12 | LP solved, objective 501.286 | Add route $(0, 1, 5, 2, 6)$ |
| 13 | | LP solved, objective 498.667 |
| 14 | | Add CC $\{1, 4, 5\}$ |
| 15 | | LP solved, objective 500.5 |
| 16 | | Convert CCs to NGCCs |
| 17 | | LP solved, objective 503.25 |

**Instance G** Figure 11 shows Instance G, with 10 customers. Here, NGSEP1 gives us an objective of 730.333 and NGSEP2 730.000. What is interesting, is that NGSEP1 also achieves the objective of 730.000, with almost the exact same solution as NGSEP2's final solution. The only difference is that NGSEP1 uses route $r_1 = (0, 2, 8, 10, 11)$, whereas NGSEP2 uses route $r_2 = (0, 8, 2, 10, 11)$. Because of this, NGSEP1 finds the CC $\{5, 6, 7, 8, 10\}$ and can thus heighten its objective, while NGSEP2 cannot find it. After all, route $r_2$ enters the set twice, but $r_1$ only once. Also, both $r_1$ and $r_2$ have a cost coefficient of 165.

So, why do both of these methods use two different routes with the same costs that serve the exact same customers? Investigating both solutions further, we observe that NGSEP1 did not even generate route $r_2$, and both generate $r_1$ in the very first pricing problem. The reason why NGSEP2 generates and uses $r_2$ is, ironically, because of CCs.

In particular, the CCs $\{1, 5, 6, 7, 8, 10\}$ and $\{1, 4, 5, 6, 7, 8, 9, 10\}$. These CCs give a double discount to $r_2$ but only a single discount to $r_1$, which makes $r_2$ more profitable to use than $r_1$ for NGSEP2. In fact, NGSEP1 also finds the CC $\{1, 5, 6, 7, 8, 10\}$, but because it is added as an NGCC, it does not give an extra discount to $r_2$, making its reduced costs equal to that of $r_1$ and thus making it undesirable to generate.

So, NGSEP2 generates route $r_2$ because it is favoured more by some CCs than $r_1$, but the usage of this route prevents finding another CC that would raise the objective further. NGSEP1 does not generate route $r_2$ and thus achieves in this case a higher objective than NGSEP2.

# C Additional Tables

## C.1 Additional Instances for Table 2

This section holds tables with additional instances for Table 2. The explanation of the columns is found in Section 6.1.

| Inst. | Sep. | Obj. | Time | #Routes | #Cuts | Sep. Time |
|---|---|---|---|---|---|---|
| A-n32-k5 | No Cuts | 758.432 | 34.4 | 289 | 0 (0) | 0.0 |
| | CCSEP | **784.000**∗ | 63.9 | 441 | 47 (43) | 0.0 |
| | NGSEP1 | **784.000**∗ | 91.7 | 396 | 50 (48) | 0.0 |
| | NGSEP2 | **784.000**∗ | 71.5 | 489 | 47 (45) | 0.0 |
| | NGSEP3 | **784.000**∗ | 89.6 | 432 | 86 (80) | 0.1 |
| | NGSEP4 | **784.000**∗ | 93.2 | 396 | 50 (48) | 0.2 |
| A-n33-k5 | No Cuts | 652.618 | 12.8 | 221 | 0 (0) | 0.0 |
| | CCSEP | **661.000** | 20.7 | 274 | 52 (34) | 0.0 |
| | NGSEP1 | **661.000** | 26.9 | 272 | 52 (47) | 0.0 |
| | NGSEP2 | **661.000** | 28.9 | 285 | 52 (47) | 0.0 |
| | NGSEP3 | **661.000** | 29.5 | 260 | 179 (167) | 0.3 |
| | NGSEP4 | **661.000** | 27.4 | 272 | 52 (47) | 0.2 |
| A-n33-k6 | No Cuts | 728.000 | 12.2 | 210 | 0 (0) | 0.0 |
| | CCSEP | 740.000 | 40.9 | 282 | 56 (39) | 0.0 |
| | NGSEP1 | **740.250** | 35.8 | 279 | 62 (52) | 0.0 |
| | NGSEP2 | 740.000 | 43.7 | 307 | 56 (41) | 0.1 |
| | NGSEP3 | **740.250** | 29.3 | 276 | 158 (143) | 0.3 |
| | NGSEP4 | **740.250** | 33.5 | 279 | 62 (52) | 0.4 |
| A-n34-k5 | No Cuts | 742.456 | 35.1 | 252 | 0 (0) | 0.0 |
| | CCSEP | 775.000 | 59.7 | 349 | 78 (41) | 0.0 |
| | NGSEP1 | 775.000 | 53.2 | 346 | 88 (48) | 0.0 |
| | NGSEP2 | 775.000 | 66.2 | 365 | 78 (50) | 0.1 |
| | NGSEP3 | **776.100** | 74.5 | 375 | 107 (63) | 1.0 |
| | NGSEP4 | 775.000 | 53.8 | 346 | 88 (48) | 0.5 |
| A-n36-k5 | No Cuts | 774.167 | 358.4 | 379 | 0 (0) | 0.0 |
| | CCSEP | 798.302 | 939.5 | 498 | 83 (32) | 0.0 |
| | NGSEP1 | 798.322 | 893.4 | 544 | 70 (50) | 0.0 |
| | NGSEP2 | 798.314 | 1115.7 | 514 | 83 (53) | 0.1 |
| | NGSEP3 | 795.093 | 1450.4 | 496 | 197 (153) | 1.2 |
| | NGSEP4 | **799.000**∗ | 1114.0 | 556 | 72 (71) | 0.5 |
| A-n37-k5 | No Cuts | 657.133 | 36.2 | 321 | 0 (0) | 0.0 |
| | CCSEP | 666.625 | 154.7 | 471 | 61 (32) | 0.0 |
| | NGSEP1 | 667.467 | 177.7 | 465 | 68 (33) | 0.1 |
| | NGSEP2 | **667.488** | 177.8 | 498 | 69 (38) | 0.1 |
| | NGSEP3 | 666.563 | 110.2 | 401 | 261 (38) | 0.6 |
| | NGSEP4 | 667.472 | 182.4 | 465 | 72 (39) | 3.7 |
| A-n37-k6 | No Cuts | 921.085 | 61.7 | 278 | 0 (0) | 0.0 |
| | CCSEP | 936.847 | 148.9 | 329 | 66 (31) | 0.0 |
| | NGSEP1 | 937.272 | 194.4 | 324 | 37 (16) | 0.0 |
| | NGSEP2 | 937.391 | 263.0 | 336 | 72 (43) | 0.1 |
| | NGSEP3 | 936.777 | 215.1 | 320 | 61 (23) | 1.0 |
| | NGSEP4 | **937.448** | 275.1 | 324 | 69 (47) | 2.5 |

| Inst. | Sep. | Obj. | Time | #Routes | #Cuts | Sep. Time |
|---|---|---|---|---|---|---|
| A-n38-k5 | No Cuts | 699.145 | 39.5 | 285 | 0 (0) | 0.0 |
| | CCSEP | 723.421 | 97.1 | 502 | 107 (64) | 0.1 |
| | NGSEP1 | **723.714** | 96.4 | 460 | 108 (80) | 0.0 |
| | NGSEP2 | **723.714** | 110.2 | 522 | 107 (86) | 0.1 |
| | NGSEP3 | 716.354 | 170.0 | 419 | 285 (33) | 1.4 |
| | NGSEP4 | **723.714** | 99.3 | 460 | 108 (80) | 1.6 |
| A-n39-k5 | No Cuts | 795.368 | 79.4 | 349 | 0 (0) | 0.0 |
| | CCSEP | 817.455 | 390.9 | 593 | 150 (59) | 0.1 |
| | NGSEP1 | 817.647 | 357.9 | 551 | 146 (99) | 0.1 |
| | NGSEP2 | **817.853** | 426.0 | 624 | 150 (95) | 0.1 |
| | NGSEP3 | 815.411 | 359.4 | 486 | 246 (120) | 1.2 |
| | NGSEP4 | **817.853** | 377.6 | 552 | 147 (98) | 5.1 |
| A-n39-k6 | No Cuts | 803.864 | 49.6 | 340 | 0 (0) | 0.0 |
| | CCSEP | 824.985 | 188.9 | 427 | 133 (33) | 0.1 |
| | NGSEP1 | **825.286** | 157.7 | 431 | 128 (47) | 0.1 |
| | NGSEP2 | 825.143 | 229.2 | 433 | 134 (44) | 0.1 |
| | NGSEP3 | 824.698 | 178.1 | 436 | 231 (56) | 1.6 |
| | NGSEP4 | **825.286** | 174.1 | 432 | 129 (48) | 3.3 |
| A-n44-k6 | No Cuts | 927.107 | 117.8 | 392 | 0 (0) | 0.0 |
| | CCSEP | **936.800** | 342.6 | 539 | 155 (98) | 0.0 |
| | NGSEP1 | **936.800** | 456.4 | 543 | 142 (106) | 0.1 |
| | NGSEP2 | **936.800** | 394.7 | 572 | 155 (118) | 0.1 |
| | NGSEP3 | 934.786 | 457.0 | 552 | 244 (140) | 1.0 |
| | NGSEP4 | **936.800** | 460.4 | 543 | 142 (106) | 1.3 |
| A-n45-k6 | No Cuts | 924.591 | 107.9 | 358 | 0 (0) | 0.0 |
| | CCSEP | 939.206 | 782.6 | 507 | 94 (37) | 0.0 |
| | NGSEP1 | 939.599 | 1039.3 | 498 | 96 (47) | 0.1 |
| | NGSEP2 | 939.599 | 1279.6 | 530 | 102 (49) | 0.1 |
| | NGSEP3 | 938.230 | 852.8 | 464 | 118 (33) | 2.1 |
| | NGSEP4 | **939.605** | 1092.7 | 499 | 98 (48) | 5.3 |
| A-n45-k7 | No Cuts | 1114.440 | 157.8 | 385 | 0 (0) | 0.0 |
| | CCSEP | 1140.880 | 486.4 | 519 | 300 (40) | 0.1 |
| | NGSEP1 | 1141.150 | 738.3 | 525 | 305 (99) | 0.1 |
| | NGSEP2 | 1141.220 | 853.2 | 562 | 304 (97) | 0.1 |
| | NGSEP3 | 1137.390 | 858.5 | 559 | 280 (130) | 2.8 |
| | NGSEP4 | **1141.340** | 871.9 | 529 | 315 (105) | 4.4 |
| A-n46-k7 | No Cuts | 899.956 | 118.8 | 402 | 0 (0) | 0.0 |
| | CCSEP | **914.000** | 315.6 | 528 | 211 (178) | 0.0 |
| | NGSEP1 | **914.000** | 259.6 | 453 | 181 (159) | 0.0 |
| | NGSEP2 | **914.000** | 382.1 | 566 | 211 (189) | 0.0 |
| | NGSEP3 | **914.000** | 241.1 | 476 | 82 (61) | 0.0 |
| | NGSEP4 | **914.000** | 262.6 | 453 | 181 (159) | 0.2 |
| A-n48-k7 | No Cuts | 1047.720 | 403.3 | 481 | 0 (0) | 0.0 |
| | CCSEP | 1071.760 | 2992.4 | 635 | 280 (69) | 0.1 |
| | NGSEP1 | **1072.050** | 2410.1 | 643 | 280 (101) | 0.0 |
| | NGSEP2 | **1072.050** | 3603.1 | 649 | 282 (171) | 0.1 |
| | NGSEP3 | 1069.330 | 2644.9 | 617 | 210 (117) | 2.9 |
| | NGSEP4 | **1072.050** | 2474.5 | 643 | 280 (101) | 7.3 |

| Inst. | Sep. | Obj. | Time | #Routes | #Cuts | Sep. Time |
|---|---|---|---|---|---|---|
| B-n31-k5 | No Cuts | 612.067 | 288.4 | 278 | 0 (0) | 0.0 |
| | CCSEP | **672.000**∗ | 314.8 | 387 | 36 (22) | 0.0 |
| | NGSEP1 | **672.000**∗ | 367.6 | 446 | 40 (31) | 0.0 |
| | NGSEP2 | **672.000**∗ | 324.3 | 431 | 36 (26) | 0.0 |
| | NGSEP3 | **672.000**∗ | 449.8 | 533 | 60 (48) | 0.1 |
| | NGSEP4 | **672.000**∗ | 371.3 | 446 | 40 (31) | 0.2 |
| B-n34-k5 | No Cuts | 744.132 | 731.2 | 292 | 0 (0) | 0.0 |
| | CCSEP | 784.970 | 997.5 | 536 | 89 (65) | 0.0 |
| | NGSEP1 | 785.444 | 1101.6 | 700 | 108 (101) | 0.0 |
| | NGSEP2 | 785.667 | 1034.4 | 619 | 89 (77) | 0.0 |
| | NGSEP3 | **785.833** | 784.8 | 618 | 184 (144) | 0.3 |
| | NGSEP4 | 785.481 | 1115.9 | 700 | 110 (103) | 2.2 |
| B-n35-k5 | No Cuts | 829.088 | 392.3 | 237 | 0 (0) | 0.0 |
| | CCSEP | **955.000**∗ | 437.1 | 606 | 85 (57) | 0.0 |
| | NGSEP1 | **955.000**∗ | 453.3 | 767 | 80 (65) | 0.0 |
| | NGSEP2 | **955.000**∗ | 544.1 | 780 | 85 (69) | 0.0 |
| | NGSEP3 | **955.000**∗ | 460.1 | 454 | 156 (139) | 0.6 |
| | NGSEP4 | **955.000**∗ | 453.6 | 767 | 80 (65) | 0.4 |
| B-n38-k6 | No Cuts | 715.102 | 282.5 | 303 | 0 (0) | 0.0 |
| | CCSEP | 804.143 | 309.0 | 473 | 54 (37) | 0.0 |
| | NGSEP1 | **804.471** | 325.9 | 461 | 53 (39) | 0.1 |
| | NGSEP2 | **804.471** | 316.5 | 511 | 54 (38) | 0.1 |
| | NGSEP3 | 804.412 | 326.0 | 454 | 54 (39) | 0.6 |
| | NGSEP4 | **804.471** | 325.1 | 461 | 53 (39) | 1.1 |
| B-n39-k5 | No Cuts | 513.991 | 4383.3 | 329 | 0 (0) | 0.0 |
| | CCSEP | **549.000**∗ | 7420.6 | 550 | 39 (37) | 0.0 |
| | NGSEP1 | **549.000**∗ | 12213.0 | 467 | 38 (37) | 0.0 |
| | NGSEP2 | **549.000**∗ | 9947.1 | 715 | 39 (38) | 0.0 |
| | NGSEP3 | **549.000**∗ | 23719.2 | 649 | 238 (237) | 0.2 |
| | NGSEP4 | **549.000**∗ | 12199.7 | 467 | 38 (37) | 0.3 |
| B-n41-k6 | No Cuts | 797.033 | 227.2 | 314 | 0 (0) | 0.0 |
| | CCSEP | 828.600 | 801.4 | 619 | 92 (73) | 0.0 |
| | NGSEP1 | **829.000**∗ | 928.8 | 657 | 82 (82) | 0.0 |
| | NGSEP2 | 829.000 | 858.6 | 683 | 92 (91) | 0.1 |
| | NGSEP3 | 829.000 | 802.5 | 692 | 244 (235) | 0.6 |
| | NGSEP4 | **829.000**∗ | 934.0 | 657 | 82 (82) | 0.2 |
| B-n43-k6 | No Cuts | 699.760 | 759.2 | 368 | 0 (0) | 0.0 |
| | CCSEP | 736.880 | 2192.0 | 569 | 129 (42) | 0.0 |
| | NGSEP1 | 737.486 | 2266.0 | 580 | 143 (69) | 0.0 |
| | NGSEP2 | 737.486 | 2380.6 | 597 | 129 (56) | 0.1 |
| | NGSEP3 | 737.418 | 1645.1 | 535 | 185 (94) | 1.3 |
| | NGSEP4 | **737.493** | 1898.8 | 584 | 145 (71) | 4.3 |

| Inst. | Sep. | Obj. | Time | #Routes | #Cuts | Sep. Time |
|---|---|---|---|---|---|---|
| B-n44-k7 | No Cuts | 858.938 | 1513.9 | 340 | 0 (0) | 0.0 |
| | CCSEP | **909.000**∗ | 2897.1 | 555 | 108 (83) | 0.0 |
| | NGSEP1 | **909.000**∗ | 2786.8 | 610 | 103 (81) | 0.0 |
| | NGSEP2 | **909.000**∗ | 3637.5 | 594 | 108 (83) | 0.0 |
| | NGSEP3 | **909.000**∗ | 2101.2 | 609 | 221 (195) | 0.1 |
| | NGSEP4 | **909.000**∗ | 2794.8 | 610 | 103 (81) | 0.4 |
| B-n45-k5 | No Cuts | 683.187 | 4506.9 | 432 | 0 (0) | 0.0 |
| | CCSEP | 750.556 | 6966.2 | 867 | 52 (44) | 0.1 |
| | NGSEP1 | **751.000**∗ | 6977.7 | 872 | 46 (41) | 0.1 |
| | NGSEP2 | **751.000**∗ | 7700.4 | 1091 | 52 (49) | 0.1 |
| | NGSEP3 | 750.684 | 15959.0 | 1012 | 100 (77) | 2.1 |
| | NGSEP4 | **751.000**∗ | 7055.2 | 872 | 46 (41) | 0.7 |
| B-n45-k6 | No Cuts | 654.279 | 570.6 | 341 | 0 (0) | 0.0 |
| | CCSEP | 677.763 | 4924.5 | 585 | 38 (21) | 0.1 |
| | NGSEP1 | 677.804 | 4091.8 | 624 | 37 (26) | 0.1 |
| | NGSEP2 | 677.804 | 5238.5 | 619 | 38 (25) | 0.1 |
| | NGSEP3 | 677.723 | 4228.0 | 610 | 94 (37) | 2.9 |
| | NGSEP4 | **677.839** | 4846.2 | 632 | 48 (32) | 3.8 |
| B-n50-k7 | No Cuts | 665.536 | 1651.1 | 451 | 0 (0) | 0.0 |
| | CCSEP | 741.000 | 1835.7 | 810 | 77 (75) | 0.0 |
| | NGSEP1 | **741.000**∗ | 1816.8 | 957 | 79 (78) | 0.0 |
| | NGSEP2 | 741.000 | 1854.4 | 888 | 77 (75) | 0.0 |
| | NGSEP3 | **741.000**∗ | 4612.2 | 1072 | 105 (100) | 0.2 |
| | NGSEP4 | **741.000**∗ | 1821.1 | 957 | 79 (78) | 0.6 |
| B-n50-k8 | No Cuts | 1255.050 | 1928.2 | 493 | 0 (0) | 0.0 |
| | CCSEP | 1303.580 | 4185.8 | 667 | 205 (51) | 0.0 |
| | NGSEP1 | 1305.470 | 4546.4 | 672 | 217 (98) | 0.0 |
| | NGSEP2 | **1305.550** | 5246.9 | 722 | 217 (115) | 0.1 |
| | NGSEP3 | 1305.090 | 5886.5 | 750 | 272 (138) | 4.1 |
| | NGSEP4 | 1305.500 | 4710.7 | 676 | 218 (100) | 5.8 |
| B-n51-k7 | No Cuts | 953.699 | 3600.3 | 461 | 0 (0) | 0.0 |
| | CCSEP | **1016.000**∗ | 4920.9 | 784 | 38 (36) | 0.0 |
| | NGSEP1 | **1016.000**∗ | 6190.0 | 764 | 57 (54) | 0.1 |
| | NGSEP2 | **1016.000**∗ | 4988.5 | 888 | 38 (38) | 0.1 |
| | NGSEP3 | **1016.000**∗ | 7690.3 | 960 | 111 (111) | 0.7 |
| | NGSEP4 | **1016.000**∗ | 5090.8 | 764 | 57 (54) | 0.2 |

| Inst. | Sep. | Obj. | Time | #Routes | #Cuts | Sep. Time |
|---|---|---|---|---|---|---|
| P-n16-k8 | No Cuts | 441.000 | 0.0 | 38 | 0 (0) | 0.0 |
| | CCSEP | 448.000 | 0.0 | 43 | 11 (11) | 0.0 |
| | NGSEP1 | 448.000 | 0.0 | 43 | 11 (11) | 0.0 |
| | NGSEP2 | 448.000 | 0.0 | 43 | 11 (11) | 0.0 |
| | NGSEP3 | **450.000**∗ | 0.0 | 43 | 33 (33) | 0.0 |
| | NGSEP4 | **450.000**∗ | 0.1 | 44 | 15 (15) | 0.0 |
| P-n19-k2 | No Cuts | 204.286 | 6.4 | 145 | 0 (0) | 0.0 |
| | CCSEP | 211.667 | 10.4 | 204 | 5 (4) | 0.0 |
| | NGSEP1 | **212.000**∗ | 12.9 | 203 | 9 (9) | 0.0 |
| | NGSEP2 | **212.000**∗ | 20.4 | 214 | 9 (9) | 0.1 |
| | NGSEP3 | 211.857 | 13.6 | 191 | 6 (6) | 0.1 |
| | NGSEP4 | **212.000**∗ | 13.3 | 203 | 9 (9) | 0.1 |
| P-n20-k2 | No Cuts | 212.000 | 5.9 | 180 | 0 (0) | 0.0 |
| | CCSEP | 215.500 | 16.4 | 214 | 6 (6) | 0.0 |
| | NGSEP1 | **215.667** | 17.3 | 202 | 8 (8) | 0.0 |
| | NGSEP2 | **215.667** | 33.8 | 230 | 8 (8) | 0.0 |
| | NGSEP3 | 215.000 | 14.0 | 198 | 4 (4) | 0.0 |
| | NGSEP4 | **215.667** | 18.1 | 202 | 8 (8) | 0.2 |
| P-n21-k2 | No Cuts | **211.000**∗ | 4.8 | 203 | 0 (0) | 0.0 |
| | CCSEP | **211.000**∗ | 4.9 | 203 | 0 (0) | 0.0 |
| | NGSEP1 | **211.000**∗ | 4.9 | 203 | 0 (0) | 0.0 |
| | NGSEP2 | **211.000**∗ | 4.9 | 203 | 0 (0) | 0.0 |
| | NGSEP3 | **211.000**∗ | 4.9 | 203 | 0 (0) | 0.0 |
| | NGSEP4 | **211.000**∗ | 5.0 | 203 | 0 (0) | 0.0 |
| P-n22-k2 | No Cuts | **215.500** | 13.4 | 226 | 0 (0) | 0.0 |
| | CCSEP | **215.500** | 13.5 | 226 | 0 (0) | 0.0 |
| | NGSEP1 | **215.500** | 13.5 | 226 | 0 (0) | 0.0 |
| | NGSEP2 | **215.500** | 13.5 | 226 | 0 (0) | 0.0 |
| | NGSEP3 | **215.500** | 13.5 | 226 | 0 (0) | 0.0 |
| | NGSEP4 | **215.500** | 13.6 | 226 | 0 (0) | 0.0 |
| P-n22-k8 | No Cuts | 589.667 | 0.0 | 61 | 0 (0) | 0.0 |
| | CCSEP | **590.000**∗ | 0.0 | 61 | 1 (1) | 0.0 |
| | NGSEP1 | **590.000**∗ | 0.0 | 61 | 1 (1) | 0.0 |
| | NGSEP2 | **590.000**∗ | 0.1 | 61 | 1 (1) | 0.0 |
| | NGSEP3 | **590.000**∗ | 0.1 | 61 | 49 (49) | 0.0 |
| | NGSEP4 | **590.000**∗ | 0.1 | 61 | 1 (1) | 0.0 |
| P-n23-k8 | No Cuts | 521.536 | 0.0 | 66 | 0 (0) | 0.0 |
| | CCSEP | **529.000**∗ | 0.1 | 91 | 32 (32) | 0.0 |
| | NGSEP1 | **529.000**∗ | 0.2 | 95 | 32 (32) | 0.0 |
| | NGSEP2 | **529.000**∗ | 0.2 | 99 | 32 (32) | 0.0 |
| | NGSEP3 | **529.000**∗ | 0.1 | 90 | 40 (40) | 0.0 |
| | NGSEP4 | **529.000**∗ | 0.1 | 95 | 32 (32) | 0.0 |

| Inst. | Sep. | Obj. | Time | #Routes | #Cuts | Sep. Time |
|---|---|---|---|---|---|---|
| P-n40-k5 | No Cuts | 448.278 | 99.0 | 354 | 0 (0) | 0.0 |
| | CCSEP | 456.875 | 103.2 | 422 | 61 (43) | 0.0 |
| | NGSEP1 | 456.875 | 105.7 | 408 | 61 (51) | 0.0 |
| | NGSEP2 | 456.875 | 106.6 | 431 | 61 (51) | 0.0 |
| | NGSEP3 | **457.100** | 134.3 | 417 | 223 (214) | 0.4 |
| | NGSEP4 | **457.100** | 110.6 | 412 | 68 (58) | 1.5 |
| P-n45-k5 | No Cuts | 499.687 | 134.1 | 389 | 0 (0) | 0.0 |
| | CCSEP | 505.850 | 186.6 | 431 | 101 (49) | 0.0 |
| | NGSEP1 | **505.960** | 166.8 | 436 | 92 (68) | 0.0 |
| | NGSEP2 | **505.960** | 216.8 | 453 | 101 (75) | 0.1 |
| | NGSEP3 | 505.661 | 288.3 | 443 | 323 (283) | 1.7 |
| | NGSEP4 | **505.960** | 174.5 | 436 | 92 (68) | 6.6 |
| P-n50-k7 | No Cuts | 542.698 | 104.4 | 397 | 0 (0) | 0.0 |
| | CCSEP | 550.794 | 214.0 | 468 | 133 (38) | 0.1 |
| | NGSEP1 | 550.857 | 209.0 | 473 | 130 (66) | 0.1 |
| | NGSEP2 | **550.876** | 237.1 | 479 | 133 (67) | 0.1 |
| | NGSEP3 | 549.811 | 346.1 | 451 | 211 (180) | 2.7 |
| | NGSEP4 | **550.876** | 256.0 | 476 | 136 (70) | 10.3 |
| P-n50-k8 | No Cuts | 612.263 | 68.5 | 307 | 0 (0) | 0.0 |
| | CCSEP | 615.024 | 206.0 | 351 | 81 (24) | 0.1 |
| | NGSEP1 | 615.181 | 195.5 | 353 | 77 (22) | 0.0 |
| | NGSEP2 | 615.155 | 232.4 | 362 | 81 (26) | 0.1 |
| | NGSEP3 | 615.011 | 178.2 | 342 | 49 (27) | 2.5 |
| | NGSEP4 | **615.283** | 279.1 | 356 | 80 (26) | 6.4 |
| P-n50-k10 | No Cuts | 686.509 | 14.5 | 270 | 0 (0) | 0.0 |
| | CCSEP | 689.580 | 29.2 | 298 | 49 (24) | 0.0 |
| | NGSEP1 | **689.779** | 27.6 | 292 | 49 (31) | 0.0 |
| | NGSEP2 | **689.779** | 35.0 | 300 | 49 (31) | 0.0 |
| | NGSEP3 | **689.779** | 50.7 | 298 | 227 (204) | 1.9 |
| | NGSEP4 | **689.779** | 30.5 | 292 | 49 (31) | 3.2 |
| P-n51-k10 | No Cuts | 732.943 | 19.1 | 304 | 0 (0) | 0.0 |
| | CCSEP | 735.828 | 88.0 | 351 | 82 (24) | 0.1 |
| | NGSEP1 | 736.407 | 123.7 | 350 | 88 (25) | 0.1 |
| | NGSEP2 | 736.407 | 143.2 | 363 | 84 (21) | 0.1 |
| | NGSEP3 | 736.457 | 118.2 | 346 | 77 (24) | 4.5 |
| | NGSEP4 | **736.726** | 170.4 | 353 | 91 (24) | 5.1 |

## C.2 Additional Instances for Table 5

This section holds tables with additional instances for Table 5. The explanation of the columns is found in Section 6.2.

| Inst. | Sep. | Obj. | Time | #R | #C | ST | #N | #B | #Int |
|---|---|---|---|---|---|---|---|---|---|
| A-n32-k5 | CCSEP | **784.000** | **63.9** | 441 | 47 | 0.0 | **1** | 0 | 1 |
| | NGSEP1 | **784.000** | 91.7 | 396 | 50 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **784.000** | 71.5 | 489 | 47 | 0.0 | **1** | 0 | 1 |
| | NGSEP3 | **784.000** | 89.6 | 432 | 86 | 0.0 | **1** | 0 | 1 |
| | NGSEP4 | **784.000** | 93.3 | 396 | 50 | 0.0 | **1** | 0 | 1 |
| A-n33-k5 | CCSEP | **661.000** | 34.5 | 380 | 52 | 0.0 | 7 | 0 | 4 |
| | NGSEP1 | **661.000** | **30.2** | 310 | 52 | 0.0 | **3** | 0 | 2 |
| | NGSEP2 | **661.000** | 37.5 | 334 | 52 | 0.0 | 5 | 0 | 3 |
| | NGSEP3 | **661.000** | 46.1 | 324 | 181 | 0.2 | 5 | 1 | 2 |
| | NGSEP4 | **661.000** | 31.1 | 310 | 52 | 0.2 | **3** | 0 | 2 |
| A-n33-k6 | CCSEP | **742.000** | 76.1 | 1070 | 64 | 0.5 | 71 | 13 | 23 |
| | NGSEP1 | **742.000** | 87.7 | 833 | 78 | 0.7 | 97 | 10 | 39 |
| | NGSEP2 | **742.000** | 84.6 | 866 | 64 | 0.6 | 87 | 12 | 32 |
| | NGSEP3 | **742.000** | **72.2** | 572 | 161 | 1.5 | **27** | 7 | 7 |
| | NGSEP4 | **742.000** | 109.8 | 833 | 78 | 24.6 | 97 | 10 | 39 |
| A-n34-k5 | CCSEP | **778.000** | 1174.5 | 4584 | 184 | 4.6 | 621 | 235 | 76 |
| | NGSEP1 | **778.000** | 520.3 | 2134 | 132 | 1.8 | 249 | 93 | 32 |
| | NGSEP2 | **778.000** | 821.7 | 2657 | 139 | 2.4 | 301 | 121 | 30 |
| | NGSEP3 | **778.000** | **251.3** | 1093 | 121 | 17.3 | **65** | 27 | 6 |
| | NGSEP4 | **778.000** | 2521.1 | 3984 | 175 | 614.4 | 633 | 245 | 72 |
| A-n36-k5 | CCSEP | 813.000† | 4524.4 | 5847 | 188 | 3.8 | 499 | 226 | 19 |
| | NGSEP1 | **799.000** | 1267.6 | 714 | 73 | 0.1 | 11 | 3 | 3 |
| | NGSEP2 | **799.000** | 3240.5 | 2250 | 125 | 0.9 | 95 | 38 | 10 |
| | NGSEP3 | 807.000† | 5022.0 | 3528 | 270 | 58.0 | 257 | 112 | 14 |
| | NGSEP4 | **799.000** | **1114.2** | 556 | 72 | 0.0 | **1** | 0 | 1 |
| A-n37-k5 | CCSEP | **669.000** | 333.3 | 1342 | 77 | 0.4 | 49 | 11 | 14 |
| | NGSEP1 | **669.000** | 319.7 | 995 | 87 | 0.2 | 35 | 9 | 9 |
| | NGSEP2 | **669.000** | **248.8** | 800 | 71 | 0.1 | **11** | 3 | 3 |
| | NGSEP3 | **669.000** | 314.5 | 956 | 270 | 1.9 | 27 | 2 | 12 |
| | NGSEP4 | **669.000** | 312.9 | 1011 | 80 | 25.8 | 25 | 4 | 9 |
| A-n37-k6 | CCSEP | **949.000**† | 3742.9 | 5109 | 234 | 6.1 | 799 | 307 | 89 |
| | NGSEP1 | **949.000**† | 3780.3 | 4398 | 196 | 4.9 | 639 | 252 | 62 |
| | NGSEP2 | 952.000† | 3852.0 | 3818 | 225 | 6.5 | 767 | 272 | 106 |
| | NGSEP3 | **949.000**† | 3803.3 | 3710 | 235 | 168.0 | 653 | 234 | 88 |
| | NGSEP4 | **949.000**† | 3861.4 | 3419 | 279 | 938.4 | 485 | 186 | 50 |
| A-n38-k5 | CCSEP | **730.000**† | 3702.2 | 6042 | 179 | 6.3 | 913 | 352 | 102 |
| | NGSEP1 | **730.000**† | 3694.9 | 5004 | 151 | 3.6 | 519 | 208 | 48 |
| | NGSEP2 | **730.000**† | 3709.3 | 4903 | 160 | 3.6 | 509 | 196 | 55 |
| | NGSEP3 | **730.000**† | 3763.7 | 4024 | 343 | 65.1 | 343 | 110 | 58 |
| | NGSEP4 | **730.000**† | 3703.0 | 4227 | 154 | 718.2 | 413 | 167 | 38 |

| Inst. | Sep. | Obj. | Time | #R | #C | ST | #N | #B | #Int |
|---|---|---|---|---|---|---|---|---|---|
| A-n39-k5 | CCSEP | 822.000† | 3988.2 | 4857 | 291 | 3.0 | 363 | 159 | 20 |
| | NGSEP1 | 822.000† | 3958.2 | 3815 | 262 | 2.2 | 261 | 104 | 24 |
| | NGSEP2 | **822.000** | **3840.4** | 3373 | 257 | 2.2 | **229** | 98 | 17 |
| | NGSEP3 | 822.000† | 3945.5 | 3708 | 419 | 75.0 | 221 | 80 | 26 |
| | NGSEP4 | 822.000† | 3991.3 | 3754 | 278 | 900.1 | 247 | 89 | 32 |
| A-n39-k6 | CCSEP | **831.000** | 3031.8 | 4148 | 207 | 5.2 | 463 | 188 | 44 |
| | NGSEP1 | **831.000** | **2037.7** | 2575 | 173 | 3.7 | **251** | 98 | 28 |
| | NGSEP2 | **831.000** | 2369.5 | 2903 | 180 | 7.8 | 363 | 134 | 48 |
| | NGSEP3 | **831.000** | 2855.9 | 2766 | 301 | 105.5 | 343 | 149 | 23 |
| | NGSEP4 | **831.000** | 2948.0 | 2416 | 177 | 802.4 | 273 | 122 | 15 |
| A-n44-k6 | CCSEP | **937.000** | **508.6** | 942 | 166 | 0.2 | 19 | 5 | 5 |
| | NGSEP1 | **937.000** | 586.2 | 735 | 153 | 0.1 | 11 | 2 | 4 |
| | NGSEP2 | **937.000** | 990.3 | 1308 | 166 | 9.8 | 39 | 12 | 8 |
| | NGSEP3 | **937.000** | 801.7 | 834 | 265 | 5.1 | 17 | 3 | 6 |
| | NGSEP4 | **937.000** | 584.6 | 666 | 149 | 8.3 | **9** | 3 | 2 |
| A-n45-k6 | CCSEP | 964.000† | 4347.6 | 3362 | 193 | 1.6 | 203 | 76 | 15 |
| | NGSEP1 | 948.000† | 4606.7 | 2410 | 183 | 9.6 | 133 | 45 | 15 |
| | NGSEP2 | 953.000† | 4862.2 | 3039 | 174 | 2.0 | 223 | 85 | 20 |
| | NGSEP3 | **944.000**† | 4458.7 | 2053 | 185 | 19.2 | 57 | 8 | 17 |
| | NGSEP4 | 953.000† | 4669.2 | 2202 | 176 | 307.2 | 105 | 40 | 8 |
| A-n45-k7 | CCSEP | 1154.000† | 4063.2 | 4177 | 510 | 3.1 | 315 | 109 | 40 |
| | NGSEP1 | 1154.000† | 4319.6 | 3266 | 429 | 1.9 | 199 | 70 | 20 |
| | NGSEP2 | **1146.000**† | 4428.3 | 2540 | 479 | 2.1 | 155 | 45 | 29 |
| | NGSEP3 | **1146.000**† | 4454.9 | 2821 | 439 | 87.7 | 143 | 49 | 19 |
| | NGSEP4 | 1154.000† | 4447.3 | 2587 | 419 | 256.2 | 131 | 35 | 24 |
| A-n46-k7 | CCSEP | **914.000** | 619.6 | 847 | 211 | 0.1 | 11 | 0 | 6 |
| | NGSEP1 | **914.000** | 605.8 | 865 | 181 | 0.0 | 9 | 1 | 4 |
| | NGSEP2 | **914.000** | 1149.6 | 959 | 211 | 0.1 | 15 | 0 | 8 |
| | NGSEP3 | **914.000** | **396.8** | 557 | 82 | 0.1 | **3** | 0 | 2 |
| | NGSEP4 | **914.000** | 607.7 | 865 | 181 | 2.1 | 9 | 1 | 4 |
| A-n48-k7 | CCSEP | 1074.000† | 6451.1 | 1143 | 313 | 0.3 | 27 | 7 | 5 |
| | NGSEP1 | 1074.000† | 6035.9 | 1267 | 306 | 0.2 | 25 | 8 | 4 |
| | NGSEP2 | **1073.000** | **5499.5** | 932 | 290 | 0.1 | **7** | 2 | 2 |
| | NGSEP3 | 1085.000† | 6101.3 | 1390 | 272 | 74.9 | 41 | 10 | 3 |
| | NGSEP4 | 1074.000† | 6111.7 | 1235 | 305 | 118.3 | 25 | 7 | 4 |

| Inst. | Sep. | Obj. | Time | #R | #C | ST | #N | #B | #Int |
|---|---|---|---|---|---|---|---|---|---|
| B-n31-k5 | CCSEP | **672.000** | **314.8** | 387 | 36 | 0.0 | **1** | 0 | 1 |
| | NGSEP1 | **672.000** | 367.6 | 446 | 40 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **672.000** | 324.3 | 431 | 36 | 0.0 | **1** | 0 | 1 |
| | NGSEP3 | **672.000** | 449.8 | 533 | 60 | 0.0 | **1** | 0 | 1 |
| | NGSEP4 | **672.000** | 371.5 | 446 | 40 | 0.0 | **1** | 0 | 1 |
| B-n34-k5 | CCSEP | **788.000**$^{\dagger}$ | 4599.6 | 7491 | 152 | 3.9 | 565 | 181 | 96 |
| | NGSEP1 | **788.000**$^{\dagger}$ | 4702.0 | 5964 | 123 | 1.6 | 241 | 75 | 44 |
| | NGSEP2 | **788.000**$^{\dagger}$ | 4642.0 | 5958 | 109 | 2.3 | 335 | 103 | 60 |
| | NGSEP3 | **788.000**$^{\dagger}$ | 4384.5 | 4943 | 206 | 20.2 | 245 | 65 | 55 |
| | NGSEP4 | **788.000**$^{\dagger}$ | 4724.8 | 5205 | 136 | 492.0 | 189 | 59 | 32 |
| B-n35-k5 | CCSEP | **955.000** | **437.1** | 606 | 85 | 0.0 | **1** | 0 | 1 |
| | NGSEP1 | **955.000** | 453.3 | 767 | 80 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **955.000** | 544.2 | 780 | 85 | 0.0 | **1** | 0 | 1 |
| | NGSEP3 | **955.000** | 460.1 | 454 | 156 | 0.0 | **1** | 0 | 1 |
| | NGSEP4 | **955.000** | 453.8 | 767 | 80 | 0.0 | **1** | 0 | 1 |
| B-n38-k6 | CCSEP | **805.000** | 412.9 | 1222 | 64 | 0.2 | 27 | 8 | 6 |
| | NGSEP1 | **805.000** | **391.5** | 699 | 53 | 0.0 | **9** | 0 | 5 |
| | NGSEP2 | **805.000** | 419.1 | 890 | 60 | 0.2 | 23 | 4 | 8 |
| | NGSEP3 | **805.000** | 415.5 | 872 | 65 | 1.7 | 15 | 1 | 7 |
| | NGSEP4 | **805.000** | 395.2 | 699 | 53 | 3.5 | **9** | 0 | 5 |
| B-n39-k5 | CCSEP | **549.000** | **7420.6** | 550 | 39 | 0.0 | **1** | 0 | 1 |
| | NGSEP1 | **549.000** | 12213.0 | 467 | 38 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **549.000** | 9947.1 | 715 | 39 | 0.0 | **1** | 0 | 1 |
| | NGSEP3 | **549.000** | 23719.2 | 649 | 238 | 0.0 | **1** | 0 | 1 |
| | NGSEP4 | **549.000** | 12199.9 | 467 | 38 | 0.0 | **1** | 0 | 1 |
| B-n41-k6 | CCSEP | **829.000** | 1275.7 | 1527 | 99 | 0.2 | 33 | 7 | 10 |
| | NGSEP1 | **829.000** | 928.8 | 657 | 82 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **829.000** | **895.1** | 700 | 92 | 0.0 | 3 | 1 | 1 |
| | NGSEP3 | **829.000** | 1094.8 | 1160 | 248 | 1.0 | 15 | 3 | 5 |
| | NGSEP4 | **829.000** | 934.2 | 657 | 82 | 0.0 | **1** | 0 | 1 |
| B-n43-k6 | CCSEP | 749.000$^{\dagger}$ | 5781.2 | 6124 | 260 | 5.8 | 781 | 352 | 26 |
| | NGSEP1 | 745.000$^{\dagger}$ | 5827.0 | 3517 | 194 | 2.8 | 385 | 167 | 18 |
| | NGSEP2 | **742.000**$^{\dagger}$ | 5956.4 | 3694 | 172 | 1.7 | 229 | 84 | 27 |
| | NGSEP3 | 743.000$^{\dagger}$ | 5275.2 | 2623 | 230 | 95.0 | 241 | 89 | 26 |
| | NGSEP4 | 743.000$^{\dagger}$ | 5465.2 | 2587 | 200 | 756.3 | 215 | 76 | 26 |

| Inst. | Sep. | Obj. | Time | #R | #C | ST | #N | #B | #Int |
|---|---|---|---|---|---|---|---|---|---|
| B-n44-k7 | CCSEP | **909.000** | 2897.1 | 555 | 108 | 0.0 | **1** | 0 | 1 |
| | NGSEP1 | **909.000** | 2786.8 | 610 | 103 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **909.000** | 3637.5 | 594 | 108 | 0.0 | **1** | 0 | 1 |
| | NGSEP3 | **909.000** | **2101.2** | 609 | 221 | 0.0 | **1** | 0 | 1 |
| | NGSEP4 | **909.000** | 2795.1 | 610 | 103 | 0.0 | **1** | 0 | 1 |
| B-n45-k5 | CCSEP | **751.000** | 8200.0 | 1185 | 53 | 0.0 | 7 | 1 | 3 |
| | NGSEP1 | **751.000** | **6977.7** | 872 | 46 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **751.000** | 7700.4 | 1091 | 52 | 0.0 | **1** | 0 | 1 |
| | NGSEP3 | 751.000† | 19561.6 | 1117 | 100 | 1.0 | 5 | 1 | 2 |
| | NGSEP4 | **751.000** | 7055.8 | 872 | 46 | 0.0 | **1** | 0 | 1 |
| B-n45-k6 | CCSEP | 679.000† | 8397.7 | 1394 | 46 | 0.3 | 37 | 9 | 8 |
| | NGSEP1 | **678.000** | **6271.2** | 775 | 38 | 0.1 | 13 | 1 | 6 |
| | NGSEP2 | **678.000** | 6490.3 | 702 | 39 | 0.1 | **7** | 1 | 3 |
| | NGSEP3 | 678.000† | 7876.0 | 1010 | 111 | 11.3 | 23 | 6 | 3 |
| | NGSEP4 | 678.000† | 8473.2 | 956 | 53 | 33.5 | 19 | 2 | 8 |
| B-n50-k7 | CCSEP | **741.000** | 1866.0 | 984 | 78 | 0.0 | 3 | 0 | 2 |
| | NGSEP1 | **741.000** | **1816.8** | 957 | 79 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **741.000** | 1981.1 | 1106 | 78 | 0.0 | 3 | 0 | 2 |
| | NGSEP3 | **741.000** | 4612.3 | 1072 | 105 | 0.0 | **1** | 0 | 1 |
| | NGSEP4 | **741.000** | 1821.7 | 957 | 79 | 0.0 | **1** | 0 | 1 |
| B-n50-k8 | CCSEP | 1360.000† | 7750.2 | 2792 | 464 | 1.5 | 139 | 40 | 11 |
| | NGSEP1 | 1321.000† | 8109.2 | 3273 | 435 | 2.5 | 263 | 101 | 16 |
| | NGSEP2 | **1312.000**† | 8800.4 | 2755 | 299 | 1.6 | 123 | 40 | 13 |
| | NGSEP3 | 1340.000† | 9424.4 | 2976 | 498 | 97.2 | 179 | 47 | 22 |
| | NGSEP4 | 1313.000† | 8274.6 | 2638 | 319 | 602.7 | 105 | 36 | 8 |
| B-n51-k7 | CCSEP | **1016.000** | **4921.0** | 784 | 38 | 0.0 | **1** | 0 | 1 |
| | NGSEP1 | **1016.000** | 6190.0 | 764 | 57 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **1016.000** | 4988.5 | 888 | 38 | 0.0 | **1** | 0 | 1 |
| | NGSEP3 | **1016.000** | 7690.3 | 960 | 111 | 0.0 | **1** | 0 | 1 |
| | NGSEP4 | **1016.000** | 5091.0 | 764 | 57 | 0.0 | **1** | 0 | 1 |

| Inst. | Sep. | Obj. | Time | #R | #C | ST | #N | #B | #Int |
|---|---|---|---|---|---|---|---|---|---|
| P-n16-k8 | CCSEP | **450.000** | 0.1 | 50 | 11 | 0.0 | 5 | 0 | 3 |
| | NGSEP1 | **450.000** | 0.1 | 49 | 11 | 0.0 | 5 | 0 | 3 |
| | NGSEP2 | **450.000** | 0.1 | 48 | 11 | 0.0 | 5 | 0 | 3 |
| | NGSEP3 | **450.000** | **0.1** | 43 | 33 | 0.0 | **1** | 0 | 1 |
| | NGSEP4 | **450.000** | 0.1 | 44 | 15 | 0.0 | **1** | 0 | 1 |
| P-n19-k2 | CCSEP | **212.000** | 21.6 | 311 | 7 | 0.1 | 9 | 2 | 3 |
| | NGSEP1 | **212.000** | **12.9** | 203 | 9 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **212.000** | 20.4 | 214 | 9 | 0.0 | **1** | 0 | 1 |
| | NGSEP3 | **212.000** | 19.2 | 234 | 8 | 0.1 | 5 | 0 | 3 |
| | NGSEP4 | **212.000** | 13.3 | 203 | 9 | 0.0 | **1** | 0 | 1 |
| P-n20-k2 | CCSEP | **216.000** | 74.1 | 664 | 9 | 0.2 | 27 | 5 | 9 |
| | NGSEP1 | **216.000** | 40.9 | 586 | 10 | 0.1 | 15 | 0 | 8 |
| | NGSEP2 | **216.000** | 59.1 | 470 | 8 | 0.1 | **11** | 2 | 4 |
| | NGSEP3 | **216.000** | **39.2** | 472 | 7 | 0.2 | **11** | 3 | 3 |
| | NGSEP4 | **216.000** | 43.4 | 586 | 10 | 1.7 | 15 | 0 | 8 |
| P-n21-k2 | CCSEP | **211.000** | 4.9 | 203 | 0 | 0.0 | **1** | 0 | 1 |
| | NGSEP1 | **211.000** | 4.9 | 203 | 0 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **211.000** | **4.9** | 203 | 0 | 0.0 | **1** | 0 | 1 |
| | NGSEP3 | **211.000** | 4.9 | 203 | 0 | 0.0 | **1** | 0 | 1 |
| | NGSEP4 | **211.000** | 5.1 | 203 | 0 | 0.0 | **1** | 0 | 1 |
| P-n22-k2 | CCSEP | **216.000** | **64.1** | 417 | 22 | 0.1 | **9** | 1 | 4 |
| | NGSEP1 | **216.000** | 64.8 | 437 | 22 | 0.1 | **9** | 1 | 4 |
| | NGSEP2 | **216.000** | 65.3 | 437 | 22 | 0.1 | **9** | 1 | 4 |
| | NGSEP3 | **216.000** | 70.3 | 462 | 115 | 0.1 | **9** | 1 | 4 |
| | NGSEP4 | **216.000** | 65.6 | 437 | 22 | 1.0 | **9** | 1 | 4 |
| P-n22-k8 | CCSEP | **590.000** | 0.1 | 61 | 1 | 0.0 | **1** | 0 | 1 |
| | NGSEP1 | **590.000** | **0.1** | 61 | 1 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **590.000** | 0.1 | 61 | 1 | 0.0 | **1** | 0 | 1 |
| | NGSEP3 | **590.000** | 0.1 | 61 | 49 | 0.0 | **1** | 0 | 1 |
| | NGSEP4 | **590.000** | 0.1 | 61 | 1 | 0.0 | **1** | 0 | 1 |
| P-n23-k8 | CCSEP | **529.000** | **0.1** | 91 | 32 | 0.0 | **1** | 0 | 1 |
| | NGSEP1 | **529.000** | 0.2 | 95 | 32 | 0.0 | **1** | 0 | 1 |
| | NGSEP2 | **529.000** | 0.2 | 99 | 32 | 0.0 | **1** | 0 | 1 |
| | NGSEP3 | **529.000** | 0.1 | 90 | 40 | 0.0 | **1** | 0 | 1 |
| | NGSEP4 | **529.000** | 0.2 | 95 | 32 | 0.0 | **1** | 0 | 1 |
| P-n40-k5 | CCSEP | **458.000** | 241.4 | 2009 | 122 | 0.8 | 87 | 29 | 15 |
| | NGSEP1 | **458.000** | 227.0 | 1256 | 113 | 0.5 | 51 | 15 | 11 |
| | NGSEP2 | **458.000** | **178.9** | 944 | 88 | 0.3 | **33** | 9 | 8 |
| | NGSEP3 | **458.000** | 290.1 | 1015 | 264 | 6.0 | 35 | 12 | 6 |
| | NGSEP4 | **458.000** | 324.9 | 1263 | 117 | 70.0 | 59 | 20 | 10 |

| Inst. | Sep. | Obj. | Time | #R | #C | ST | #N | #B | #Int |
|---|---|---|---|---|---|---|---|---|---|
| P-n45-k5 | CCSEP | **510.000**† | 3784.1 | 5518 | 316 | 4.8 | 559 | 255 | 23 |
| | NGSEP1 | **510.000**† | 3764.5 | 4379 | 223 | 3.6 | 445 | 189 | 32 |
| | NGSEP2 | 513.000† | 3814.1 | 5165 | 256 | 3.8 | 383 | 174 | 12 |
| | NGSEP3 | 513.000† | 3890.1 | 3467 | 447 | 155.2 | 257 | 107 | 18 |
| | NGSEP4 | **510.000**† | 3780.5 | 3438 | 201 | 1338.9 | 299 | 115 | 29 |
| P-n50-k7 | CCSEP | **554.000**† | 3813.5 | 4682 | 316 | 13.5 | 581 | 241 | 46 |
| | NGSEP1 | 558.000† | 3803.0 | 2849 | 283 | 8.4 | 451 | 182 | 40 |
| | NGSEP2 | 568.000† | 3831.5 | 2907 | 312 | 4.5 | 421 | 194 | 8 |
| | NGSEP3 | 559.000† | 3935.9 | 3100 | 346 | 327.1 | 273 | 99 | 29 |
| | NGSEP4 | 558.000† | 3852.5 | 2873 | 290 | 1234.1 | 277 | 113 | 21 |
| P-n50-k8 | CCSEP | 647.000† | 3790.7 | 3648 | 521 | 9.7 | 1065 | 453 | 63 |
| | NGSEP1 | 636.000† | 3781.6 | 2644 | 346 | 4.3 | 429 | 171 | 32 |
| | NGSEP2 | 644.000† | 3819.9 | 2586 | 332 | 5.2 | 439 | 187 | 19 |
| | NGSEP3 | **629.000**† | 3778.7 | 2098 | 343 | 590.6 | 357 | 147 | 23 |
| | NGSEP4 | 637.000† | 3867.8 | 2870 | 407 | 1125.3 | 345 | 124 | 38 |
| P-n50-k10 | CCSEP | **696.000**† | 3629.5 | 2773 | 406 | 11.7 | 1191 | 528 | 65 |
| | NGSEP1 | **696.000**† | 3628.0 | 1965 | 310 | 7.4 | 739 | 328 | 36 |
| | NGSEP2 | 697.000† | 3634.5 | 2116 | 293 | 7.7 | 697 | 314 | 29 |
| | NGSEP3 | 697.000† | 3650.6 | 1767 | 411 | 569.2 | 447 | 208 | 11 |
| | NGSEP4 | **696.000**† | 3635.2 | 1437 | 280 | 1999.0 | 365 | 159 | 19 |
| P-n51-k10 | CCSEP | 748.000† | 3685.3 | 2922 | 497 | 7.6 | 783 | 348 | 37 |
| | NGSEP1 | 747.000† | 3724.1 | 1953 | 428 | 4.1 | 375 | 170 | 10 |
| | NGSEP2 | 747.000† | 3738.6 | 2251 | 370 | 4.6 | 371 | 136 | 43 |
| | NGSEP3 | **741.000**† | 3721.5 | 1642 | 291 | 304.5 | 327 | 142 | 20 |
| | NGSEP4 | 745.000† | 3765.1 | 1772 | 335 | 941.0 | 321 | 127 | 29 |