Les Cahiers du GERAD ISSN: 0711–2440

Addressing orientation-symmetry in the Time Window Assignment Vehicle Routing Problem

K. Dalmeijer,
G. Desaulniers

G–2018–48

July 2018

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2018
– Bibliothèque et Archives Canada, 2018

Legal deposit – Bibliothèque et Archives nationales du Québec, 2018
– Library and Archives Canada, 2018

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Addressing orientation-symmetry in the Time Window Assignment Vehicle Routing Problem

Kevin Dalmeijer [a]

Guy Desaulniers [b]

[a] *Erasmus School of Economics, 3000DR Rotterdam, The Netherlands*

[b] *GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7*

dalmeijer@ese.eur.nl
guy.desaulniers@gerad.ca

July 2018
Les Cahiers du GERAD
G–2018–48

Abstract: The Time Window Assignment Vehicle Routing Problem (TWAVRP) is the problem of assigning time windows for delivery before demand volume becomes known. This implies that vehicle routes in different demand scenarios have to be synchronized, such that the same client is visited around the same time in each scenario. For TWAVRP instances that are difficult to solve by current methods, we observe many similar solutions in which one or more routes have a different orientation, i.e., the clients are visited in the reverse order. We introduce a new branching method that eliminates this orientation-symmetry from the search tree, and we present enhancements to make this method efficient in practice. Next, we present a branch-price-and-cut algorithm based on the new branching method. Through computational experiments, we show that our algorithm outperforms other solution methods, and we solve 29 previously unsolved benchmark instances to optimality. We conclude that properly addressing orientation-symmetry significantly improves the computational tractability of the TWAVRP. The main ideas of this paper are not TWAVRP specific and can be applied to other vehicle routing problems with consistency considerations or synchronization requirements.

Keywords: Vehicle routing, time window assignment, symmetry, consistency, branch-price-and-cut

1 Introduction

We consider the Time Window Assignment Vehicle Routing Problem (TWAVRP), the problem of assigning time windows to clients in a distribution network before the demand of the clients is revealed. It is assumed, however, that a list of possible demand scenarios and corresponding probabilities is given. The TWAVRP asks for an assignment of time windows to the clients and, for each scenario, a set of feasible routes that adhere to the assigned time windows and the scenario specific demand, such that the expected cost of distribution is minimized. That is, each client is assigned a single time window, which should be respected, regardless of which scenario occurs. Each assigned time window, or endogenous time window, is of fixed width. Furthermore, the endogenous time windows must fall within given exogenous time windows.

The TWAVRP was first introduced by Spliet and Gabor (2015), and was inspired by distribution networks of retail chains. In this setting, the clients are retail stores that are supplied from a central depot and the exogenous time windows represent the opening hours of the stores. In retail, stores are often assigned a time window that does not change for a longer period of time (e.g. one year). The TWAVRP was introduced to assign time windows in this setting, taking the uncertainty of the future demand into account.

Spliet and Gabor (2015) present a branch-price-and-cut (BPC) algorithm to solve TWAVRP instances with up to 25 clients and three demand scenarios, within one hour of computation time. Spliet and Desaulniers (2015) present a BPC algorithm for the DTWAVRP, a variant of the TWAVRP where each endogenous time window is chosen from a discrete set of options. With their exact method, the authors are able to solve instances of similar size as the instances by Spliet and Gabor (2015), and they present a heuristic for larger instances. Another variant of the TWAVRP is the TWAVRP with time-dependent travel times, for which a BPC algorithm is presented in Spliet et al. (2018).

Taking a different approach, Dalmeijer and Spliet (2018) introduce a branch-and-cut (BC) algorithm for the TWAVRP, which makes use of a new class of valid inequalities; the precedence inequalities. This algorithm solves the benchmark instances introduced by Spliet and Gabor (2015) on average 193.9 times faster than the original BPC algorithm, and allows for instances with up to 35 clients and three demand scenarios to be solved.

For an increasing number of companies, client satisfaction has become a priority, as satisfied clients provide a better lifetime value. As client satisfaction is often the result of consistent service, various types of consistency have been considered in the literature (see Kovacs et al. (2014) for a recent survey). In the case of the TWAVRP, clients are always visited within the assigned time window, which can be classified as *time-consistency*.

A problem closely related to the TWAVRP is the Consistent Vehicle Routing Problem (ConVRP), introduced by Groër et al. (2009). Where the TWAVRP only imposes time-consistency, the ConVRP is more restrictive and imposes both time-consistency and driver-consistency, i.e., each client should always be visited by the same driver. Lian (2017) presents a branch-and-price algorithm for the ConVRP, in which driver-consistency is enforced in the pricing subproblem. Subramanyam and Gounaris (2018) focus on the single vehicle variant of the ConVRP, and present an exact algorithm in which they branch on the endogenous time windows, such that in each step the problem decomposes in multiple instances of the Traveling Salesman Problem with Time Windows (TSPTW).

The TWAVRP can also be seen as a vehicle routing problem with *operation synchronization* constraints (Drexl, 2012). That is, the routes in the different scenarios have to be synchronized, such that the visits to a client (the operations) in different scenarios are at approximately the same time.

It is well known that symmetry has a negative impact on branch-and-bound algorithms, as it can lead to multiple branches for which symmetric optimal solutions are computed. In this paper, we introduce the concept of *orientation-symmetry* and we show that orientation-symmetry has a big impact on both the algorithms by Spliet and Gabor (2015) and Dalmeijer and Spliet (2018).

We then propose a branching method that eliminates orientation-symmetry from the search tree. As a part of this branching method, we present an algorithm for finding the best solution to the TWAVRP out

of a set of orientation-symmetric solutions. We make use of the precedence inequalities to speed up this algorithm and make it efficient in practice.

Based on this new branching method, we construct a BPC algorithm to solve the TWAVRP to optimality. Furthermore, we apply the new branching method to the BC algorithm by Dalmeijer and Spliet (2018). Surprisingly, we find from computational experiments that BPC can outperform BC, if orientation-symmetry is properly addressed. This is in contrast with the earlier work of Dalmeijer and Spliet (2018), in which they find that BC outperforms BPC.

This paper is structured as follows. In Section 2, we present the set-partitioning formulation for the TWAVRP. We also state the precedence inequalities, as they will be used in a later section of the paper. In Section 3 we define orientation-symmetry, and we present a new branching method to eliminate orientation-symmetry from the search tree. Section 4 presents the BPC algorithm, and Section 5 details our computational experiments. In the final section, we give a conclusion and present some directions for further research.

2 Mathematical formulation and precedence inequalities

In this section, we present the set-partitioning formulation for the TWAVRP introduced by Spliet and Gabor (2015) and the precedence inequalities proposed by Dalmeijer and Spliet (2018).

2.1 Set-partitioning formulation

The TWAVRP is defined on a graph $G = (V, A)$ with $V = \{0, 1, \ldots, n+1\}$. Vertices 0 and $n+1$ correspond to the depot and are referred to as the *starting depot* and the *ending depot*, respectively. The other vertices correspond to the n clients. For convenience, let $V' = \{1, \ldots, n\}$. The set A consists of all arcs from the starting depot to the clients, all arcs between clients, and all arcs from the clients to the ending depot. Each arc in $(i, j) \in A$ is associated with a cost c_{ij} and a travel time t_{ij}. Both the costs and the travel times are non-negative and satisfy the triangle inequality. Service times at the clients are included in the travel times by adding the service time at client i to the travel time of all outgoing arcs.

The demand uncertainty is modeled through Ω, a finite set of demand scenarios. Each demand scenario $\omega \in \Omega$ occurs with probability p_ω. Naturally, $\sum_{\omega\in\Omega} p_\omega = 1$. For client $i \in V'$, demand in scenario $\omega \in \Omega$ is given by d_i^ω. We have access to an unlimited number of identical vehicles with capacity Q. For all $i \in V'$ and $\omega \in \Omega$, we assume that $0 < d_i^\omega \leq Q$.

Each vertex $i \in V$ is associated with an exogenous time window with starting time $s_i \geq 0$ and ending time $e_i > s_i$. Vertices 0 and $n+1$ both refer to the depot and thus $s_0 = s_{n+1}$ and $e_0 = e_{n+1}$. Each client $i \in V'$ has to be assigned a time window of given non-negative width u_i, which has to be within the exogenous time window. Service must start, but not necessarily complete, within the assigned time window. For a given scenario, the size of the arc set A may be reduced by removing arcs that are infeasible due to time constraints or due to capacity constraints in that scenario.

In this paper, a *route* refers to a pair (P, t), with P an elementary path in G from the starting depot to the ending depot, and t a vector of times at which service starts at the clients. We define t_r^i to be the time at which service starts at client i if route r is used. To allow for a concise formulation, t_r^i is defined to be equal to zero for all clients i that are not in the path P corresponding to route r. For each $\omega \in \Omega$, $\mathcal{R}(\omega)$ is the set of all feasible routes in scenario ω. A route is feasible if it satisfies the exogenous time windows and respects the vehicle capacity, with respect to the demand in scenario ω. For all routes r and clients i we define a_r^i to be equal to one if client i is served by route r, and zero otherwise. Similarly, let b_r^{ij} be equal to one if arc (i, j) is contained in route r, and zero otherwise. The cost of route r is given by c_r.

For each client $i \in V'$ the variable y_i indicates the start of the endogenous time window. As the endogenous time window is of fixed width u_i, it follows that the time window ends at $y_i + u_i$. Recall that each endogenous time window should be contained in the exogenous time window, and hence $y_i \in [s_i, e_i - u_i]$.

The route variables θ_r^ω give the contribution of route r to the solution in scenario ω. The flow on arc (i,j) in scenario ω is given by the flow variables x_{ij}^ω. We obtain a feasible solution to the TWAVRP if all flow variables are integral, even if the route variables are fractional. This follows from the fact that a convex combination of routes corresponding to the same path is feasible.

Given this notation, the TWAVRP can be expressed as the following set-partitioning model:

$$\min \sum_{\omega\in\Omega} p_\omega \sum_{r\in\mathcal{R}(\omega)} c_r\theta_r^\omega, \tag{1}$$

$$\text{s.t.} \sum_{r\in\mathcal{R}(\omega)} a_r^i\theta_r^\omega = 1 \qquad \forall i\in V',\ \omega\in\Omega, \tag{2}$$

$$\sum_{r\in\mathcal{R}(\omega)} t_r^i\theta_r^\omega \geq y_i \qquad \forall i\in V',\ \omega\in\Omega, \tag{3}$$

$$\sum_{r\in\mathcal{R}(\omega)} t_r^i\theta_r^\omega \leq y_i + u_i \quad \forall i\in V',\ \omega\in\Omega, \tag{4}$$

$$y_i \in [s_i, e_i - u_i] \qquad \forall i\in V', \tag{5}$$

$$\theta_r^\omega \geq 0 \qquad \forall\omega\in\Omega,\ r\in\mathcal{R}(\omega), \tag{6}$$

$$x_{ij}^\omega = \sum_{r\in\mathcal{R}(\omega)} b_r^{ij}\theta_r^\omega \qquad \forall\omega\in\Omega,\ (i,j)\in A, \tag{7}$$

$$x_{ij}^\omega \in \{0,1\} \qquad \forall\omega\in\Omega,\ (i,j)\in A. \tag{8}$$

The objective function (1) is to minimize the expected cost of distribution. Constraints (2) ensure that every customer is visited exactly once per scenario. Constraints (3) and (4) make sure that in each scenario, the time of service for each client is within the endogenous time window. Constraints (5) define the continuous variables that indicate the starting times of the endogenous time windows. The route variables are defined by Constraints (6) and are linked to the flow variables by Constraints (7). Finally, Constraints (8) state that the flow variables are subject to binary requirements.

Recall that each route r consists of a pair (P,t). As waiting is allowed, and t is a continuous vector of times of service, it follows that the number of routes, and thus the number of variables in the above formulation, is typically infinite.

2.2 Precedence inequalities

The precedence inequalities are inequalities based on the following fact. Consider an integral solution to the TWAVRP that contains route $r=(P,t)\in R(\omega)$ in scenario ω and route $r'=(P',t')\in R(\omega')$ in scenario ω'. Assume that there exist two client vertices $i,j\in V'$ such that P contains a subpath p from i to j and P' contains a subpath p' from j to i. Let A_p and $A_{p'}$ be the arc sets of p and p', respectively. It then holds that

$$\sum_{(k,l)\in A_p} t_{kl} + \sum_{(k,l)\in A_{p'}} t_{kl} \leq u_i + u_j. \tag{9}$$

This fact follows from the following observation. For route r to visit both i and j within their endogenous time windows, client i should be served after time y_i and client j should be served before time y_j+u_j. Hence, an upper bound on the time between the visits to i and j is given by $y_j+u_j-y_i$. That is, $\sum_{(k,l)\in A_p} t_{kl} \leq y_j+u_j-y_i$. Analogously, route r' implies that $\sum_{(k,l)\in A_{p'}} t_{kl} \leq y_i+u_i-y_j$. By adding up these two inequalities, y_i and y_j cancel, and Inequality (9) follows.

Though not presented in Dalmeijer and Spliet (2018), it is straightforward to extend this idea to the case where the depot is involved. Define an endogenous time window for the depot that has the same width as the exogenous time window. That is, define $u_0 = e_0 - s_0$. Now for a subpath p of P from $i=0$ to $j\in V'$ and a subpath p' of P' from $j\in V'$ to $n+1$, Inequality (9) holds by the same argument as before.

Dalmeijer and Spliet (2018) use inequalities of the form (9) to derive the precedence inequalities. In this paper, we only state the *path precedence inequalities*, a subclass of the precedence inequalities.

Consider a pair of elementary paths p and p', as described above, such that Inequality (9) does *not* hold. Furthermore, consider a pair of distinct scenarios $\omega, \omega' \in \Omega$. The path-precedence inequalities can then be stated as:

$$\sum_{(k,l)\in A_p} x^{\omega}_{kl} + \sum_{(k,l)\in A_{p'}} x^{\omega'}_{kl} \leq |A_p| + |A_{p'}| - 1. \tag{10}$$

As p and p' are chosen such that Inequality (9) does not hold, it must be that $\sum_{(k,l)\in A_p} x^{\omega}_{kl} < |A_p|$ or $\sum_{(k,l)\in A_{p'}} x^{\omega'}_{kl} < |A_{p'}|$. By integrality of the x-variables, Inequality (10) follows.

3 Orientation-symmetry

Given a feasible solution to the TWAVRP, it is often possible to find another feasible solution by first changing the orientation of one or more routes, and then reassigning the endogenous time windows. *Changing the orientation of a route* here means that the clients are visited in the reverse order. E.g., if we change the orientation of a route in scenario ω that starts at the depot, visits clients 1, 2 and 3, and then returns to the depot, we obtain a route in scenario ω that starts at the depot, visits clients 3, 2 and 1, and then returns to the depot.

If one feasible solution can be turned into another feasible solution by changing route orientations and reassigning the endogenous time windows, then we say that these solutions are *orientation-symmetric*. Reassigning the endogenous time windows may indeed be necessary: by changing the orientation of one or more routes, the current endogenous time windows can become infeasible. If the arc costs are symmetric, then orientation-symmetric solutions have the same objective value.

Out of the forty benchmark instances introduced by Spliet and Gabor, instances 12, 36 and 37 are among the most difficult to solve with both the algorithm by Spliet and Gabor (2015) and the algorithm by Dalmeijer and Spliet (2018). Interestingly, these instances all contain a number of orientation-symmetric optimal solutions: 11, 112 and 40, respectively.

It is well known that the presence of symmetric solutions has a negative impact on the performance of branch-and-bound methods, which may in part explain why instances 12, 36 and 37 are difficult to solve. In the remainder of Section 3, we propose a new branching method that ensures that orientation-symmetric solutions are always in the same branch, thereby eliminating orientation-symmetry in the search tree. In Section 5 we show computationally that this new branching method improves algorithmic performance.

3.1 Branching method to eliminate orientation-symmetry

We make the distinction between (directed) arcs and (undirected) edges. An edge is given by a pair (i, j), with $i < j$ and $i, j \in \{0\} \cup V'$. Each edge (i, j) corresponds to at most two arcs in A. When i and j are client vertices ($i, j \in V'$), edge (i, j) corresponds to the arcs (i, j) and (j, i), if they exist. When i is the depot vertex ($i = 0$) and j is a client vertex ($j \in V'$), edge $(0, j)$ corresponds to the arcs $(0, j)$ and $(j, n+1)$, if they exist. We use E to denote the set of all edges.

Let the variable z^{ω}_{ij} represent the total flow on edge $(i, j) \in E$ in scenario $\omega \in \Omega$. The flow on an edge in a given scenario is obtained by summing the flows on the corresponding arcs in the same scenario. For example, for edge $(1, 3) \in E$ in scenario ω we have $z^{\omega}_{13} = x^{\omega}_{13} + x^{\omega}_{31}$, assuming that both arcs $(1, 3)$ and $(3, 1)$ exist.

Observation 1. *The edge flows of any feasible solution to the TWAVRP are integral and correspond, per scenario, to disjoint undirected elementary paths that start and end at the depot.*

If branching decisions are based on z^{ω}_{ij}, then orientation-symmetric solutions always end up in the same branch. This follows from the fact that, by definition, orientation-symmetric solutions have the same edge flows. In Spliet and Gabor (2015) and Dalmeijer and Spliet (2018) branching decisions are based on x^{ω}_{ij} instead of z^{ω}_{ij}.

To solve the TWAVRP, it is insufficient to branch on fractional z_{ij}^{ω} variables only. Recall that a solution with integral arc flows corresponds to a feasible solution to the TWAVRP. However, if the edge flows are integral, the arc flows may be fractional. In general, we do not obtain a feasible solution to the TWAVRP if the arc flows are fractional, even if the edge flows are integral and correspond to disjoint undirected elementary paths. Undirected paths do not properly represent the passing of time. As a result, we cannot guarantee that the endogenous time windows are satisfied.

Consider a node in the search tree for which we compute a solution to the linear programming (LP) relaxation of (1)–(8). If the edge flows are not integral, we branch on the variable z_{ij}^{ω} that corresponds to the non-depot edge that has flow closest to 0.5. It can be shown that if the non-depot edges have integral flows, then the depot edges also have integral flows. If the edge flows and the arc flows are both integral, we have obtained a feasible solution to the TWAVRP and the current node can be pruned by integrality. Finally, if the edge flows are integral and the arc flows are fractional, we use Procedure 1 to continue the search.

Procedure 1 Processing a node with integral edge flows and fractional arc flows

1: Solve a restricted TWAVRP in which the edge flows are fixed to their current values.
2: **if** Restricted TWAVRP is feasible and has cost lower than the incumbent solution **then**
3: Make the optimal solution to the restricted TWAVRP the incumbent solution.
4: **end if**
5: Add a constraint to forbid the current integral edge flows.

In Step 1, we solve a restricted TWAVRP in which the edge flows are fixed. By definition, solutions that have the same edge flows are orientation-symmetric. Hence, solving the restricted TWAVRP amounts to finding the best solution among a set of orientation-symmetric solutions. In Section 3.2 we present an algorithm for solving the restricted TWAVRP.

In Step 5, a constraint is added to forbid the current integral edge flows. This constraint forces at least one of the z_{ij}^{ω} variables to take a different value. On the other hand, solutions to the TWAVRP with different integral edge flows should not be cut off. In Section 3.3, we present this nontrivial constraint.

Note that we do not assume that the arc costs are symmetric. If they are symmetric, however, then the objective value can be obtained directly from the edge flows. It follows that if the restricted TWAVRP is feasible, we obtain an incumbent solution with cost equal to the lower bound of the current node. As a result, the current node can immediately be pruned by optimality.

3.2 Solving the restricted TWAVRP with fixed edge flows

In this section, we present an algorithm for solving the TWAVRP with fixed edge flows. We will refer to this problem as the *restricted TWAVRP*. From Observation 1 it follows directly that if the edge flows do not correspond to disjoint undirected elementary paths, then there does not exist a solution to the TWAVRP that is consistent with the fixed edge flows. We therefore assume without loss of generality that the integral edge flows can be represented by m disjoint undirected elementary paths.

It follows that the restricted TWAVRP can be seen as a TWAVRP on a restricted arc set and can thus be solved by the algorithms in Spliet and Gabor (2015) and Dalmeijer and Spliet (2018). In this section, however, we exploit the fact that all solutions to the restricted TWAVRP are orientation-symmetric to construct an algorithm that is more efficient in practice.

To obtain arc flows consistent with the fixed edge flows, all m undirected paths must be given an orientation. First, arbitrarily assign a default orientation to each of the undirected paths. Then, for all $k \in \{1, \ldots, m\}$, let the variable o_k be equal to 1 if the orientation of path k is equal to the default orientation, or -1 otherwise. For single client paths, we set $o_k = 1$ without loss of generality. Note that the edge flows z_{ij}^{ω} and the path orientations o_k together define the arc flows x_{ij}^{ω}. For example, if edge $(1, 3)$ in scenario ω is on path k, then the value of o_k determines whether $x_{13}^{\omega} = 1$ or $x_{31}^{\omega} = 1$.

We can now enumerate all possible assignments of the o_k variables. Each assignment $(o_1, o_2, \ldots, o_m) \in \{-1, 1\}^m$ defines the arc flows of a potential solution. When the arc flows are fixed, the MIP formulation

of Dalmeijer and Spliet (2018) reduces to a linear program, which can be solved to find a feasible solution to the TWAVRP if one exists. In Appendix A we propose an alternative method to solve the TWAVRP for fixed arc flows that does not rely on linear programming.

When all possible assignments of the o_k variables have been enumerated, we have either found the minimum cost solution to the restricted TWAVRP, or we have proven that no solution exists. Hence, we have solved the restricted TWAVRP, as required. Recall that for instances with symmetric costs, all orientation-symmetric solutions yield the same objective value. In that case, the enumeration can be stopped after the first feasible solution has been found.

3.2.1 More efficient enumeration

The enumeration algorithm presented above enumerates all 2^m route orientations. Some assignments of the o_k variables, however, lead to violated path precedence inequalities. As the path precedence inequalities are valid, these assignments cannot lead to a feasible solution to the TWAVRP and can thus be ignored. In this section, we use this observation to significantly reduce the number of route orientations that has to be enumerated.

We introduce a *conflict graph* to represent conflicts between the o_k variables due to violated path precedence inequalities. The vertex set of the conflict graph is given by $\{1, \ldots, m\}$, in which vertex k corresponds to o_k.

The edges of the conflict graph correspond to violations of the path precedence inequalities. Recall that o_k and o_l define the arc flows on paths k and l, respectively. This implies that for given values of o_k and o_l we can test for a violation of Inequality (10). If $o_k \neq o_l$ results in a violated path precedence inequality (i.e., both $(o_k, o_l) = (1, -1)$ and $(o_k, o_l) = (-1, 1)$ result in a violation), then k and l are connected by a *positive edge*: an edge with value 1. If $o_k = o_l$ results in a violated path precedence inequality (i.e., both $(o_k, o_l) = (1, 1)$ and $(o_k, o_l) = (-1, -1)$ result in a violation), then k and l are connected by a *negative edge*: an edge with value -1. If neither case is applicable, then edge (k, l) does not exist.

An assignment of the o_k variables is said to be *conflict-free* if for every edge (k, l) the value of $o_k o_l$ is equal to the edge value. If the conflict graph contains a positive edge (k, l), we require $o_k = o_l$ in a conflict-free assignment. Similarly, for a negative edge (k, l), we require $o_k \neq o_l$ for the assignment to be conflict-free.

Figure 1 presents an example of a conflict graph which allows for different conflict-free assignments. Positive edges are shown as dotted lines and negative edges are shown as solid lines. One of the conflict-free assignments is given by $o_1 = o_2 = o_4 = o_5 = 1$ and $o_3 = o_6 = -1$.

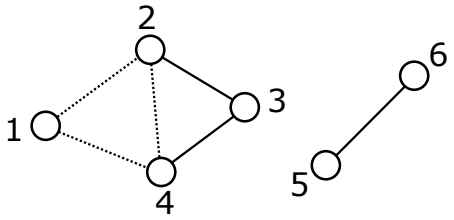

Figure 1: Example of a conflict graph.

Proposition 2. *Only a conflict-free assignment of the o_k variables can lead to a feasible solution to the TWAVRP.*

Proof. *Assume that the conflict graph is not conflict-free. Without loss of generality, this conflict is due to edge (k, l). If (k, l) is a positive edge, then $o_k \neq o_l$, or there would not be a conflict. By definition, k and l are connected by a positive edge if $o_k \neq o_l$ results in a violated path precedence inequality. As the path precedence inequalities are valid, it follows that the current assignment of the o_k variables cannot lead to a feasible solution to the TWAVRP.*

Likewise, it can be seen that if (k, l) is a negative edge, the conflict also results in a violated path precedence inequality. It follows that only a conflict-free assignment of the o_k variables can lead to a feasible solution to the TWAVRP.

The conflict graph contains a positive edge if both $(o_k, o_l) = (1, -1)$ and $(o_k, o_l) = (-1, 1)$ result in a violated path precedence inequality. Note that if the travel times are symmetric, it follows from (9) and (10) that $(o_k, o_l) = (1, -1)$ leads to a violation if and only if $(o_k, o_l) = (-1, 1)$ leads to a violation. A similar result is true for the negative edges. When solving asymmetric instances, it can be beneficial to use a directed conflict graph that considers the pairs $(o_k, o_l) = (1, -1)$ and $(o_k, o_l) = (-1, 1)$ separately. However, as we focus on instances that are difficult due to symmetry, we do not present this extension here.

The conflict graph can be constructed in polynomial time. This follows from the fact that a feasible solution contains at most $|\Omega|n$ paths. Hence, there are $\mathcal{O}(|\Omega|^2 n^2)$ vertex pairs, and for each pair, at most four possible assignments of (o_k, o_l) have to be tested for violated path precedence inequalities. As path precedence inequalities can be separated in polynomial time (Dalmeijer and Spliet, 2018), the conflict graph can be constructed in polynomial time.

By Proposition 2, only a conflict-free assignment of the o_k variables can lead to a feasible solution to the TWAVRP. Hence, after constructing the conflict graph, the next step is to enumerate all conflict-free assignments.

First, we modify the conflict graph by replacing each positive edge (k, l) by a dummy vertex that is connected to both k and l by negative edges. Note that this forces $o_k = o_l$, just like the original positive edge. Figure 2 shows the result of modifying the conflict graph presented in Figure 1. Clearly, there is a bijection between the conflict-free assignments of the original graph and those of the modified graph. It follows that it is sufficient to find all conflict-free assignments in the modified conflict graph.

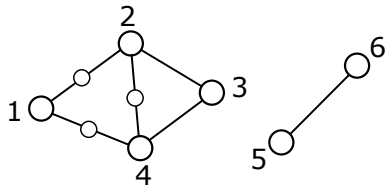

Figure 2: Example of a modified conflict graph.

In the modified conflict graph, adjacent vertices are connected by a negative edge, and must thus be assigned different values. If we associate the value 1 with the color red, and the value -1 with the color blue, then every conflict-free assignment in the modified conflict graph corresponds to a vertex coloring using at most two colors.

It is well known that we can use breadth-first search to either find a two-coloring, or to find an odd cycle, which proves that no two-coloring exists (e.g., see Kleinberg and Tardos (2006), Chapter 3.4). This algorithm takes polynomial time.

When a single two-coloring is known, we can easily find additional two-colorings by switching the roles of red and blue in any of the connected components. In fact, all two-colorings can be obtained in this way. This follows from the fact that any connected graph allows for at most two distinct two-colorings.

In conclusion, we can now enumerate all two-colorings of the modified conflict graph, or equivalently, enumerate all conflict-free assignments of the original conflict graph. The number of conflict-free assignments is equal to two to the power of the number of connected components of the conflict graph. Note that this can be significantly smaller than the total number of possible assignments, which is equal to 2^m.

If no conflict-free assignment can be found, we obtain an odd cycle in the modified conflict graph. This cycle indicates which undirected paths cause the TWAVRP to be infeasible. In Section 3.3, we use this information to strengthen the constraint that forbids the current integral edge flows, which is added in Step 5 of Procedure 1.

3.3 Cutting off integral edge flows

In this section we present constraints that cut off the current integral edge flows. A constraint of this type is needed in Step 5 of Procedure 1. As in the previous section, we assume that integral edge flows are represented by disjoint undirected elementary paths. Let the current edge flows be given by $z^\omega_{ij} = \bar{z}^\omega_{ij}$ for all $(i,j) \in E$ and $\omega \in \Omega$.

Figure 3 gives an example of the edge flows in a single scenario. Vertices 1 up to 7 correspond to the clients, and vertex 0 corresponds to the depot. This example contains three disjoint undirected elementary paths. Note that client 1 is contained in a single client path. By definition $z^\omega_{01} = x^\omega_{01} + x^\omega_{1,n+1}$, and hence in the example we have $\bar{z}^\omega_{01} = 2$. The other undirected paths contain multiple clients. As these paths are elementary, it follows that the corresponding edge flows are equal to one.

To obtain a constraint that cuts off the current integral edge flows, we make use of the more general Proposition 3. For a given selection of edges with non-zero flow, Proposition 3 provides an inequality that can only be satisfied by changing at least one of the flows on the selected edges. For technical reasons we impose that if a depot edge is selected that is part of a path with multiple clients, then the adjacent non-depot edge is also selected.

Let $\hat{E} \subseteq E$ be a set of edges. Given a pair of clients $i, j \in V'$ $(i \neq j)$ we define $\lambda^{\hat{E}}_{ij}$ to be the number of depot edges in $\hat{E}$ that are incident to i or j. That is, $\lambda^{\hat{E}}_{ij} = |\{(0,i),(0,j)\} \cap \hat{E}|$. Given this notation, Proposition 3 can be stated as follows.

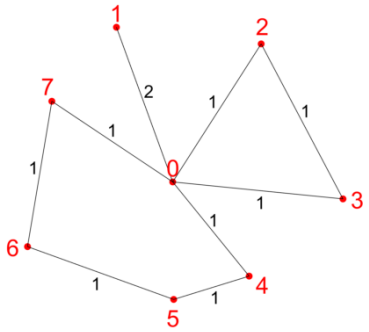

Figure 3: Example current edge flows $\bar{z}^\omega_{ij}$ for a given scenario ω.

Proposition 3. *Let the current integral edge flows be given by $\bar{z}^\omega_{ij}$ for all $(i,j) \in E$ and $\omega \in \Omega$. For each scenario $\omega \in \Omega$, let $E^\omega \subset E$ be a selection of edges with non-zero flow in scenario ω. Furthermore, let each E^ω satisfy a technical condition: if $(0,j) \in E^\omega$ and $\bar{z}^\omega_{0j} = 1$ then E^ω contains an edge adjacent to $(0,j)$. Then, the constraint*

$$\sum_{\omega\in\Omega}\left(\sum_{(i,j)\in E^\omega} z^\omega_{ij} + \sum_{(i,j)\in E^\omega | i,j\in V'} \lambda^{E^\omega}_{ij} z^\omega_{ij}\right) \leq \sum_{\omega\in\Omega}\left(\sum_{(i,j)\in E^\omega} \bar{z}^\omega_{ij} + \sum_{(i,j)\in E^\omega | i,j\in V'} \lambda^{E^\omega}_{ij} \bar{z}^\omega_{ij}\right) - 1 \qquad (11)$$

is violated by an integral edge flow if and only if $z^\omega_{ij} = \bar{z}^\omega_{ij}$ for all $(i,j) \in E^\omega$ and $\omega \in \Omega$.

Proof. *See Appendix B.*

To illustrate Proposition 3, we construct an inequality for the example in Figure 3. Note that $\lambda^{E^\omega}_{ij}$ can be seen as an additional weight that is assigned to non-depot edges that are adjacent to depot edges. To forbid the current flow on $E^\omega = \{(0,2),(2,3),(0,3)\}$, we obtain the constraint $(z^\omega_{02} + z^w_{23} + z^w_{03}) + 2z^\omega_{23} \leq 4$. The edge $(2,3)$ gets additional weight $(\lambda^{E^\omega}_{23} = 2)$ because it is adjacent to two depot edges in E^ω. This additional weight is necessary: the constraint $z^\omega_{02} + z^w_{23} + z^w_{03} \leq 2$ (obtained by setting $\lambda^{E^\omega}_{ij} = 0$) would cut off the current solution but is not valid, as it would also cut off the solution $z^\omega_{02} = z^\omega_{03} = 2$.

To obtain a constraint that cuts off the current integral edge flows, we can now apply Proposition 3. For each scenario $\omega \in \Omega$, choose E^ω to be the set of *all* edges with non-zero flow in scenario ω. Proposition 3

then provides a constraint that can be used in Step 5 of Procedure 1. For example, the scenario shown in Figure 3 contributes $z^\omega_{01} + (z^\omega_{02} + z^\omega_{23} + z^\omega_{03}) + 2z^\omega_{23} + (z^\omega_{04} + z^\omega_{45} + z^\omega_{56} + z^\omega_{67} + z^\omega_{07}) + z^\omega_{45} + z^\omega_{67}$ to the left-hand side of this constraint and 14 to the right-hand side. The technical condition in Proposition 3 is satisfied because the edges with non-zero flow make disjoint undirected elementary paths.

It is often possible to add a constraint that is stronger than the one proposed in the previous paragraph. Recall from Section 3.2.1 that if there is no conflict-free assignment, we obtain a cycle of conflicting undirected paths. These paths conflict due to the path precedence inequalities.

Let a *conflict edge* be an edge that is contained in at least one of these path precedence inequalities. It can be seen that if we replace each path in the conflict by its smallest subpath containing all conflict edges, then the conflict remains.

It follows that, for each scenario $\omega \in \Omega$, we may choose E^ω to represent these smallest subpaths. For example, if $(2,3)$, $(4,5)$ and $(6,7)$ are conflict edges in Figure 3, then we can choose $E^\omega = \{(2,3),(4,5),(5,6),(6,7)\}$. If necessary, additional edges may be added to E^ω to ensure that the technical condition is satisfied. As the selected edge flows lead to a conflict, Proposition 3 provides a valid inequality.

The constraint that we construct based on the conflict is stronger than the constraint that we obtain in case that no conflict is found. This follows from the fact that the latter only cuts off the current integral edge flows, while the former can also cut off other integral edge flows that would generate the same conflict in the conflict graph.

4 Branch-price-and-cut algorithm

In this section we present the proposed BPC algorithm to solve the TWAVRP. First, we detail how we solve the LP relaxation of (1)–(8) with column generation. Next, we discuss the route relaxations that we use and the valid inequalities that we apply. We end this section with an overview of the BPC algorithm.

4.1 Column generation

The name *branch-and-price* is due to Barnhart et al. (1998), who introduce the term for branch-and-bound algorithms that make use of column generation to solve the LP relaxations. In our case, model (1)–(8) can be seen as the *integer master problem*, which contains a huge number of variables. Its LP relaxation, given by (1)–(6), is referred to as the *master problem*. The master problem is solved by column generation.

Column generation is iterative. In each iteration we first solve a *restricted master problem* (RMP), which is obtained by restricting the master problem to a subset of the variables. Next, we solve a subproblem, the *pricing problem*, to obtain variables with negative reduced costs that can be added to the RMP. If no variable with negative reduced cost can be found, the solution to the RMP is an optimal solution to the master problem.

For all $i \in V'$ and $\omega \in \Omega$, let β^ω_i, $\gamma^\omega_i \geq 0$ and $\delta^\omega_i \leq 0$ be the dual variables corresponding to Constraints (2), (3) and (4), respectively. For convenience, let $\pi^\omega_i = \gamma^\omega_i + \delta^\omega_i$. Note that both β^ω_i and π^ω_i are unrestricted. The reduced cost $\bar{c}^\omega_r$ corresponding to the route variable θ^ω_r can now be expressed as

$$\bar{c}^\omega_r = p_w c_r - \sum_{i \in V'} \beta^\omega_i a^i_r - \sum_{i \in V'} \pi^\omega_i t^i_r. \tag{12}$$

The pricing problem is to find a route variable with negative reduced cost.

Note that the pricing problem decomposes into a separate problem for each scenario. For a given scenario $\omega \in \Omega$, we model the pricing problem as an Elementary Shortest Path Problem with Resource Constraints (ESPPRC) and linear node costs. The goal is to find a minimum cost elementary path from the starting depot to the ending depot such that both vehicle capacity and the exogenous time windows are respected.

To each arc $(i,j) \in A$ we assign the cost $\bar{c}_{ij}(t^j_r) = p_w c_{ij} - \beta^\omega_j - \pi^\omega_j t^j_r$ if $j \in V'$ and $\bar{c}_{ij}(t^j_r) = p_w c_{ij}$ otherwise. It follows that a path with negative costs corresponds to a route with negative reduced cost. Note that $\bar{c}_{ij}(t^j_r)$ depends linearly on the time at which client j is visited, which is a decision variable in the pricing problem.

Liberatore et al. (2011) encounter a similar pricing problem in the context of the Vehicle Routing Problem with Soft Time Windows and they present a bi-directional labeling algorithm for solving it. To solve the pricing problem of the TWAVRP, we implement a mono-directional variant of their algorithm with a simplified dominance rule.

Labeling algorithms maintain a set of *labels* that each corresponds to a path starting at the depot. These paths are extended along all arcs, which results in new labels being generated according to the *extension functions*. Labels that are infeasible are eliminated. Furthermore, a *dominance rule* is applied to eliminate labels that are non Pareto-optimal. Eventually, we obtain a set of feasible elementary paths from the starting depot to the ending depot that includes the shortest elementary path.

Next, we present the definition of the labels, the extension functions and the dominance rule. Our notation is consistent with Contardo et al. (2015), to which we refer for more information on labeling algorithms for vehicle routing problems.

A label is given by a tuple $L = (i, \bar{c}(T), d, t, U, p)$, where $i \in V$ is the end vertex of the path associated with L, $\bar{c}(T)$ is the cost function that returns the minimum possible cost of the path if i is visited at or before time T, d is the cumulative vehicle load, t is the earliest arrival time in i, $U \subseteq V'$ is the set of clients that are visited by the path or that are unreachable due to capacity or time constraints, and p is the predecessor label of L, i.e., the label that has been extended to obtain L. The components of label L are denoted by $i(L)$, $\bar{c}(L,T)$, $d(L)$, $t(L)$, $U(L)$ and $p(L)$, respectively.

The function $\bar{c}(L,T)$ is defined on the domain $T \in [t(L), e_{i(L)}]$. It is shown by Ioachim et al. (1998) that $\bar{c}(L,T)$ is convex, non-increasing and piece-wise linear in T. Hence, $\bar{c}(L,T)$ can be represented by a list of coordinates.

The initial label is given by $L = (0, 0, 0, s_0, \emptyset, \emptyset)$. A label for which $i(L) = i$ can be extended to a label L' over the arc $(i,j) \in A$ if $j \notin U(L)$. For the extended label, $i(L')$, $d(L')$, $t(L')$, $U(L')$ and $p(L')$ are given by the following straightforward extension functions:

$$
\begin{aligned}
i(L') &= j && (13)\\
d(L') &= d(L) + d_j^{\omega} && (14)\\
t(L') &= \max\{s_j, t(L) + t_{ij}\} && (15)\\
U(L') &= U(L) \cup \{j\} \cup \{k \in V' | d(L') + d_k^{\omega} > Q \vee t(L') + t_{jk} > e_k\} && (16)\\
p(L') &= L. && (17)
\end{aligned}
$$

The label L' is feasible if $d(L') \leq Q$ and $t(L') \leq e_j$.

Next, we give the extension function for $\bar{c}(L',T)$. Let $T^* \in [t(L'), e_j]$ be the time at which vertex j must be visited to minimize the cost of the path corresponding to L'. If vertex j is visited at time t_r^j then vertex i must be visited at or before time $t_r^j - t_{ij}$. Furthermore, vertex i must be visited at or before time e_i. It follows that

$$
T^* = \underset{t_r^j \in [t(L'), e_j]}{\arg\min} \ \{\bar{c}(L, \min\{t_r^j - t_{ij}, e_i\}) + \bar{c}_{ij}(t_r^j)\}. \tag{18}
$$

Hence, T^* can be determined by minimizing a one-dimensional, convex and piece-wise linear function. This is straightforward, as there is always a minimum at one of the breakpoints of the function or at one of the two boundaries of the domain.

If vertex j must be visited at or before some time T, Ioachim et al. (1998) show that it is optimal to visit vertex j at time $t_r^j = \min\{T, T^*\}$. It follows that the extension function for $\bar{c}(L',T)$ is given by

$$
\bar{c}(L',T) = \begin{cases} \bar{c}(L, \min\{T - t_{ij}, e_j\}) + \bar{c}_{ij}(T) & \text{if } t(L') \leq T \leq T^* \\ \bar{c}(L, \min\{T^* - t_{ij}, e_j\}) + \bar{c}_{ij}(T^*) & \text{if } T^* \leq T \leq e_j. \end{cases} \tag{19}
$$

To reduce the number of labels, we apply a dominance rule to remove non Pareto-optimal labels. We say that label L_1 dominates label L_2 if all of the following conditions are met:

i. $i(L_1) = i(L_2)$
ii. $\bar{c}(L_1, T) \leq \bar{c}(L_2, T) \quad \forall T \in [t(L_1), e_{i(L_1)}] \cap [t(L_2), e_{i(L_2)}]$
iii. $d(L_1) \leq d(L_2)$
iv. $t(L_1) \leq t(L_2)$
v. $U(L_1) \subseteq U(L_2)$.

Conditions (i)-(v) imply that L_2 is not Pareto-optimal for any $T \in [t(L_1), e_{i(L_1)}] \cap [t(L_2), e_{i(L_2)}]$ (Liberatore et al., 2011). By Conditions (i) and (iv) we have $[t(L_1), e_{i(L_1)}] \cap [t(L_2), e_{i(L_2)}] = [t(L_2), e_{i(L_2)}]$. It follows that L_2 is not Pareto-optimal for any value of T in the domain of $\bar{c}(L_2, T)$. Hence, label L_2 can be eliminated.

With our dominance rule, label L_1 can only dominate label L_2 if $\bar{c}(L_1, T) \leq \bar{c}(L_2, T)$ for all T in the intersection of the two domains. The dominance rule presented by Liberatore et al. (2011) also allows L_1 to dominate L_2 on a subinterval of $[t(L_1), e_{i(L_1)}]$, which results in more labels being eliminated. The advantage of our dominance rule is that comparing labels is easier, as no subintervals have to be determined. Furthermore, if the exogenous time windows and the travel times are all integers, it can be shown that the breakpoints of $\bar{c}(L, T)$ are integer (Ioachim et al., 1998). This improves the numerical stability of the algorithm.

4.2 Route relaxations

Enforcing elementarity in the pricing problem can result in a large amount of Pareto-optimal labels, because there are 2^n possible subsets of the clients. To reduce the number of labels, elementarity is often (partially) relaxed and cyclic routes are allowed to be added to the RMP. For a cyclic route $r = (P, t)$, a_r^i is the number of times that client $i \in V'$ is visited, b_r^{ij} is the number of times that arc $(i, j) \in A$ is used and t_r^i is the sum of all the times at which client $i \in V'$ is visited.

Adding cyclic routes to the RMP may decrease the value of the lower bound provided by (1)–(8). In an integral solution to the TWAVRP, however, cyclic routes cannot be selected due to Constraints (2). It follows that the branch-price-and-cut algorithm remains exact.

We incorporate the *ng*-route relaxation introduced by Baldacci et al. (2011) and the Strong Degree Constraints (SDCs) introduced by Contardo et al. (2013). Both have been used to solve various vehicle routing problem. E.g., the *ng*-route relaxation has been applied to the TWAVRP with Time-Dependent Travel Times by Spliet et al. (2018) and the combination of *ng*-route relaxation and SDCs has been shown to be effective for the VRPTW by Contardo et al. (2015).

We initialize the *ng*-route relaxation with neighborhoods of size 10 and we allow for dynamically adding clients to these neighborhoods. The number of times that a client can be added to a neighborhood is limited to 30 per scenario. We also allow for dynamically adding at most 30 SDCs per scenario.

To use the *ng*-route relaxation and the SDCs, both the RMP and the labeling algorithm have to be modified. These modifications are not TWAVRP specific and are detailed in Contardo et al. (2015).

4.3 Heuristic pricing

To speed up the column generation algorithm we make use of heuristic dynamic programming as described by Desaulniers et al. (2008), among others.

At first, we ignore arcs that do not seem promising according to their reduced cost. Furthermore, we ignore some of the resources when using the dominance rule. This simplifies the pricing problem, which can now be solved quickly by our labeling algorithm. If the labeling algorithm identifies a route with negative reduced cost, it can be added to the RMP.

If no more routes with negative reduced cost can be found, we take back into account some of the arcs and some of the resources that we previously ignored. Eventually we solve the full pricing problem, and as a result our column generation algorithm remains exact.

4.4 Valid inequalities

In this section we introduce the valid inequalities that we use as cutting planes to strengthen the lower bound of (1)–(8).

The first class of valid inequalities that we use is the rounded capacity inequalities, or the *capacity cuts* for short, which are known to be effective for vehicle routing problems (Baldacci et al., 2012). The capacity cuts are given by

$$\sum_{(i,j)\in A|i\in S,j\in V\setminus S} x_{ij}^{\omega} \geq \left\lceil \frac{\sum_{i\in S} d_i^{\omega}}{Q} \right\rceil \quad \forall S \subseteq V', |S| \geq 2, \omega \in \Omega. \tag{20}$$

The left-hand side of Inequality (20) is the total arc flow from the clients in a given subset S to the other vertices, which is an upper bound on the number of vehicles visiting the clients in S. This follows from the fact that the same vehicle can enter and exit S multiple times. The right-hand side of Inequality (20) is a lower bound on the number of vehicles that is required to satisfy the demand of the clients in S.

We use the heuristic separation algorithm by Lysgaard (2003) to find violated capacity cuts. The details of this algorithm are presented in Lysgaard et al. (2004). Any inequality defined on the x-variables can be included in the master problem without complicating the pricing problem, see Desaulniers et al. (2011) for details.

Next to the capacity cuts, we make use of the three client subset-row inequalities (3SR-inequalities), which is a subclass of the more general class introduced by Jepsen et al. (2008). The 3SR-inequalities can be expressed as follows:

$$\sum_{r\in\mathcal{R}(\omega)} \left\lfloor \frac{\sum_{i\in S} a_r^i}{2} \right\rfloor \theta_r^{\omega} \leq 1 \quad \forall S \subseteq V', |S| = 3, \omega \in \Omega, \tag{21}$$

that is, for each subset of three clients, there can be at most one route that visits at least two of them.

3SR-inequalities can readily be added to the master problem, but they do complicate the pricing problem as their duals cannot be incorporated in the modified arc costs. Such inequalities are said to be non-robust (Pessoa et al., 2008). Jepsen et al. (2008) detail how the labeling algorithm can be modified to allow for 3SR-inequalities in the master problem. This involves introducing additional resources to correctly model the reduced cost.

We do not add precedence inequalities as valid inequalities. The precedence inequalities are used in Section 3 to derive a branching method that eliminates orientation-symmetry. From preliminary experiments we found that if this new branching method is used, then adding precedence inequalities as cutting planes does not have a significant effect on the performance of our algorithm.

4.5 Branching

We explore the nodes of the search tree using best-first search. We first try to branch on the number of vehicles used in a given scenario. Next, we branch using the new branching method introduced in Section 3.1. Note that branching on the number of vehicles does not introduce orientation-symmetry into the search tree because orientation-symmetric solutions use the same number of vehicles.

With the branching method from Section 3.1 we either branch on a fractional edge or we add a constraint to forbid the current integral edge flow. Enforcing that the edge $(i,j) \in E$ is not used in scenario $\omega \in \Omega$ is achieved by removing the edge from the pricing problem. To force flow on edge $(i,j) \in E$ in scenario $\omega \in \Omega$ we use the constraint $z_{ij}^{\omega} = 1$. The constraints that are used to cut off the integral edge flows are also stated in terms of z_{ij}^{ω}. By definition, constraints in terms of the z-variables can be rewritten in terms of the x-variables, and can thus be added to the master problem without complicating the pricing problem.

5 Computational experiments

The BPC algorithm is implemented in GENCOL, which is a general purpose solver for solving routing and scheduling problems through decomposition and column generation. GENCOL is coded in C and C++.

All experiments are run on a server with an Intel Xeon E3-1226 v3 3.30GHz processor and 16GB of RAM. For a fair comparison with earlier work we use only a single thread and we set a time limit of one hour per instance for all experiments. All linear programs are solved with the commercial solver CPLEX version 12.6.3. To prevent problems with numerical stability of the simplex method, we enable the *numerical emphasis* setting.

5.1 Test instances

We make use of the benchmark instances for the TWAVRP as introduced by Spliet and Gabor (2015) and extended by Dalmeijer and Spliet (2018). These instances are available in the vehicle routing problem repository VRP-REP (Mendoza et al., 2014).

In total there are 90 benchmark instances, consisting of ten instances with 10 clients, ten instances with 15 clients, etc., up to 50 clients. These clients are uniformly distributed over a square with sides of five hours. The starting depot and the ending depot are both located at the center of the square. The travel costs and the travel times are given by the Euclidean distances between the locations.

Each instance contains three demand scenarios with the same probability of occurrence. The demands of different clients are uncorrelated and the average demand is about 1/6 of the vehicle capacity. The exogenous time windows have a width of 10.8 hours on average, while the endogenous time windows have a width of two hours.

To also be able to test our algorithm on larger instances, we extend the instance set by Dalmeijer and Spliet (2018). The new instances are generated in the same way, but contain more clients and demand scenarios. The extended instance set will be made available on VRP-REP.

5.2 Comparison with Dalmeijer and Spliet (2018)

We first test our BPC algorithm on the benchmark instances used by Dalmeijer and Spliet (2018) and we compare to their results. Note that Dalmeijer and Spliet (2018) obtain their results on an Intel i7 3.5GHz processor, which is comparable to, if not better than, the processor that we use.

The results are summarized in Table 1 and the full table is presented as Table 5 in Appendix C. The algorithm by Dalmeijer and Spliet (2018) is denoted by BC. The BPC algorithm presented in this paper is denoted by BPC+OS to stress the fact that this algorithm addresses orientation-symmetry.

The columns labeled 'Seconds' state the average times in seconds to (attempt to) solve the benchmark instances to optimality, with a maximum time of 3600 seconds allowed per instance. The number of nodes in the search tree that have been explored during this time is given by the 'Nodes' columns.

The 'Optimality gap' columns present the percentage gap between the optimal objective value and the lower bound after the algorithm terminates. Similarly, 'Root gap' presents the percentage gap between the optimal objective value and the lower bound after processing the root node. If the optimal objective value is unavailable we use the best known upper bound from either BC or BPC+OS to calculate the gaps. Finally, the columns labeled 'Solved' state the total number of instances that could be solved within one hour of computation time.

Table 1 shows that for the given benchmark instances, BPC+OS outperforms BC. By using BPC+OS instead of BC, the average solution time is decreased by 78%. All instances that can be solved by BC can also be solved by BPC+OS (Table 5). Additionally, 29 instances that could not be solved before are now solved to optimality. Only four of the benchmark instances remain unsolved.

Table 1: Comparison between BC and BPC+OS on the Dalmeijer and Spliet (2018) benchmark instances.

	Seconds		Nodes		Optimality gap		Root gap		Solved	
Clients	BC	BPC+OS	BC	BPC+OS	BC	BPC+OS	BC	BPC+OS	BC	BPC+OS
10	0.1	0.2	9.3	1.2	0	0	0.17	0.04	10	10
15	4.6	2.3	1,498.4	10.8	0	0	0.59	0.10	10	10
20	2.2	2.5	116.9	2.6	0	0	0.35	0.05	10	10
25	12.4	11.6	524.3	5.4	0	0	0.69	0.03	10	10
30	544.0	70.6	9,336.6	12.5	0.15	0	1.67	0.13	9	10
35	1531.7	421.4	16,846.9	23.3	0.33	0.02	1.47	0.12	6	9
40	3252.0	542.6	22,903.4	15.3	0.82	0.03	2.18	0.12	2	9
45	3600.0	705.8	10,089.7	9.3	1.72	0.03	2.53	0.09	0	9
50	3600.0	1028.7	4,904.6	16.0	2.35	0.14	2.90	0.23	0	9
Avg.	1394.1	309.5	7,358.9	10.7	0.60	0.02	1.39	0.10	57/90	86/90

BC and BPC+OS use different strategies to solve the TWAVRP. BC has a relatively weak LP relaxation that is easy to calculate, while BPC+OS relies on a strong bound that takes more computational effort to determine. It is thus no surprise that BPC+OS explores less nodes and has smaller root gaps than BC. What is interesting to observe, however, is how big these differences are. The average root gap for BPC+OS is only 0.10% and on average only 10.7 nodes have to be explored to close this gap. This is in stark contrast with the 1.39% average root gap for BC, which requires 7,358.9 nodes on average to close the gap.

5.3 Effect of addressing orientation symmetry

In this section we consider the effect of addressing orientation symmetry. To this end, we also test BC with the new branching method (denoted by BC+OS) and BPC+OS without the new branching method (denoted by BPC).

For BPC we add precedence inequalities in the same way as for BC (see Dalmeijer and Spliet (2018)). For BC+OS we do not add precedence inequalities; preliminary experiments suggest that adding precedence inequalities does not improve performance when the new branching method is used. The four algorithms are compared in Table 2, which is a summary of Table 6 in Appendix C.

Table 2: Comparison between BC, BC+OS, BPC and BPC+OS on the Dalmeijer and Spliet (2018) benchmark instances.

	Seconds				Solved			
Clients	BC	BC+OS	BPC	BPC+OS	BC	BC+OS	BPC	BPC+OS
10	0.1	0.0	0.3	0.2	10	10	10	10
15	4.6	1.1	22.5	2.3	10	10	10	10
20	2.2	0.4	364.4	2.5	10	10	9	10
25	12.4	3.4	751.5	11.6	10	10	8	10
30	544.0	461.6	1894.7	70.6	9	9	5	10
35	1531.7	1208.3	1514.3	421.4	6	7	6	9
40	3252.0	3069.3	2548.0	542.6	2	2	4	9
45	3600.0	3600.0	2203.2	705.8	0	0	6	9
50	3600.0	3600.0	3002.8	1028.7	0	0	3	9
Avg.	1394.1	1327.1	1366.8	309.5	57/90	58/90	61/90	86/90

Table 2 shows that, for the given instances, addressing orientation symmetry has a positive effect on the performance of both BC and BPC. For BC the average solution time decreases from 1394.1 seconds to 1327.1 seconds. Note, however, that the average is heavily influenced by the instances that cannot be solved before the time limit. If we only consider the instances that can be solved by both BC and BC+OS, we actually see a decrease of 37.5% in average solution time (Table 6).

For BPC, addressing orientation symmetry clearly has a bigger effect: the average solution time decreases from 1366.8 seconds to 309.5 seconds. Furthermore, we see that the number of instances that can be solved to optimality increases from 61 to 86.

If we compare BC with BPC we observe that BC outperforms BPC if the number of clients is at most 30. For 35 clients or more, BPC has a better performance on average. If we compare BC+OS with BPC+OS we see that instances with up to 25 clients only take a short time to solve, and for instances with 30 clients or more, BPC+OS outperforms BC+OS.

One of the reasons that BPC+OS is so effective, is because the new branching method significantly reduces the amount of nodes that have to be processed. For BPC+OS, the average number of nodes that is processed for each benchmark instance is only 10.7, while for BPC the average is 145.0. This shows that orientation-symmetry is indeed prominent, and that properly addressing orientation-symmetry reduces the required computational effort.

5.4 Instances with additional clients

We have shown that the BPC algorithm that addresses orientation-symmetry can solve 86 out of the 90 benchmark instances by Dalmeijer and Spliet (2018). To test the limits of our new algorithm, we also perform computational experiments with the extended instance set. In this section, we first increase the number of clients while keeping the number of demand scenarios constant at three. In the next section, we increase the number of scenarios.

The results for instances with 55 clients up to 65 clients are presented in Table 3. We have chosen to only report the optimality gap and the root gap for instances that are solved to optimality; as we do not put effort into generating good upper bounds, the optimality gap and the root gap are otherwise uninformative.

Table 3: Computational results for BPC+OS on instances with 55 up to 65 clients and three demand scenarios.

Inst.	Clients	Seconds	Nodes	Optimality gap	Root gap
91	55	3600.0	17	-	-
92	55	3600.0	17	-	-
93	55	3600.0	38	-	-
94	55	3600.0	7	-	-
95	55	306.0	10	0	0.01
96	55	2082.7	9	0	0.06
97	55	1938.0	21	0	0.09
98	55	3600.0	5	-	-
99	55	3216.5	25	0	0.14
100	55	3600.0	8	-	-
101	60	3600.0	16	-	-
102	60	2042.6	7	0	0.04
103	60	3600.0	5	-	-
104	60	3600.0	15	-	-
105	60	3600.0	19	-	-
106	60	358.4	1	0	0
107	60	3600.0	5	-	-
108	60	3600.0	16	-	-
109	60	3600.0	5	-	-
110	60	3600.0	14	-	-
111	65	3600.0	2	-	-
112	65	3600.0	13	-	-
113	65	3600.0	3	-	-
114	65	3600.0	7	-	-
115	65	2742.1	7	0	0.02
116	65	3600.0	7	-	-
117	65	3600.0	12	-	-
118	65	3600.0	3	-	-
119	65	3600.0	13	-	-
120	65	3600.0	7	-	-

Based on Table 3 we conclude that our algorithm cannot consistently solve the benchmark instances with more than 50 clients. We manage to solve four out of the ten instances with 55 clients, two out of the ten instances with 60 clients, and a single instance with 65 clients.

Only a small number of nodes can be processed within the one hour time limit. The long time per node is due to the more complicated pricing problems, but also due to the many valid inequalities that are added.

In Section 5.3 we have seen that addressing orientation-symmetry reduces the number of nodes that have to be processed. This becomes even more important as the time spent per node increases.

5.5 Instances with additional scenarios

In this section we test the performance of our algorithm when the number of demand scenarios increases. We consider instances from the extended instance set with 10 clients up to 50 clients, and with either three, five or seven demand scenarios.

The results of this computational experiment are summarized in Table 4. For two of the instances with five scenarios, CPLEX reported that one of the linear programs could not be solved due to numerical issues. These instances have been reported as unsolved with a solution time of 3600 seconds.

Table 4: Computational results for BPC+OS on instances with 10 up to 50 clients and three up to seven demand scenarios.

	Seconds			Solved		
Clients	**3 scen.**	**5 scen.**	**7 scen.**	**3 scen.**	**5 scen.**	**7 scen.**
10	0.2	0.5	0.9	10	10	10
15	2.3	364.2	1033.4	10	9	8
20	2.5	14.7	56.0	10	10	10
25	11.6	398.2	1294.7	10	9	7
30	70.6	514.5	1281.1	10	9	8
35	421.4	899.5	2531.8	9	9	5
40	542.6	1547.0	2926.9	9	7	5
45	705.8	2507.2	3308.5	9	5	1
50	1028.7	3427.2	3600.0	9	1	0
Avg.	309.5	1074.8	1781.5	86/90	69/90	54/90

As expected, Table 4 shows that the complexity of the TWAVRP increases with the number of scenarios. Out of the 90 instances with three scenarios, 86 instances can be solved to optimality in one hour of computation time. For five scenarios and seven scenarios the number of solved instances is 69 and 54, respectively.

Our algorithm can solve almost all instances with five scenarios and up to 35 clients in one hour of computation time. Out of the instances with seven scenarios, more than half of the instances up to 30 clients can be solved to optimality.

6 Conclusion

In this paper we define orientation-symmetry for the TWAVRP and we observe that orientation-symmetry is common, especially for instances that are difficult to solve by exact methods. To overcome the problem of orientation-symmetry, we introduce a new branching method that eliminates orientation-symmetry from the search tree. Based on this new branching method, we present a branch-price-and-cut algorithm to solve the TWAVRP.

The computational experiments show that on the benchmark instances by Dalmeijer and Spliet (2018) our algorithm outperforms the other solution methods for instances with three demand scenarios and 30 clients or more. For smaller instances, our algorithm is competitive. Out of the 90 benchmark instances, we solve 86 instances to optimality within one hour of computation time. Out of these instances, 29 are solved for the first time.

Our results suggest that addressing orientation-symmetry is effective. Using the new branching method reduces the solution time from 1366.8 seconds to 309.5 seconds per instance on average. One reason for this improvement is that the new branching method decreases the number of nodes that have to be processed from 145.0 to 10.7 per instance on average.

For future work it can be interesting to consider heuristics based on the current algorithm. Another direction for further research is to analyze how many demand scenarios are sufficient to obtain a good

time window assignment under various assumptions about the demand distribution and the structure of the network. Finally, we remark that the main ideas in this paper are not TWAVRP specific and may be applied to other vehicle routing problems with consistency considerations or synchronization requirements. Our computational experiments show that addressing orientation-symmetry improves both the branch-and-cut algorithm and the branch-price-and-cut algorithm. Hence, addressing orientation-symmetry may also be effective for other problems and algorithms.

A Solving the TWAVRP for fixed arc flows

In this section we present a simple algorithm for solving the TWAVRP for fixed arc flows, which does not rely on linear programming.

For each scenario, we assume that the arc flows are integral and give m disjoint directed elementary paths from the starting depot to the ending depot. This implies that a potential solution consists of m routes. The k'th route is given by $r_k = (P_{r_k}, t_{r_k})$ and is used in scenario ω_k. Note that P_{r_k} follows directly from the arc flows. As P_{r_k} is known, we also know $a^i_{r_k}$ for all $i \in V'$ and $b^{ij}_{r_k}$ for all $(i,j) \in A$. By definition, $\theta^{\omega_k}_{r_k} = 1$ for all $k \in \{1,\dots,m\}$ and all other route variables are equal to zero.

Furthermore, we assume that the routes are such that, per scenario, the capacity constraint is satisfied and that each client is visited exactly once. If one of these conditions does not hold, we can immediately conclude that the TWAVRP is infeasible for the given arc flows.

Given these assumptions, the goal is to find values for $t^j_{r_k}$ for all $k \in \{1,\dots,m\}$, $j \in V'$ and values for y_j for all $j \in V'$ such that the routes $r_k = (P_{r_k}, t_{r_k})$ are feasible and the Constraints (2)–(8) are satisfied. For convenience, let $t^0_{r_k}$ be the time that the vehicle assigned to route k leaves the depot and let $t^{n+1}_{r_k}$ be the time that the vehicle returns to the depot.

We present Algorithm 2 for solving the TWAVRP for fixed arc flows. This algorithm repeatedly calculates a lower bound $\underline{t}^j_{r_k}$ on the final value of $t^j_{r_k}$, and updates $t^j_{r_k}$ accordingly. The TWAVRP is proven to be infeasible for the given arc flows if the lower bound on $t^j_{r_k}$ exceeds the upper bound of value e_j. The loop ends after an iteration in which none of the $t^j_{r_k}$ values are updated. Finally, values for y_j are calculated.

Algorithm 2 Solving the TWAVRP for fixed arc flows

1: Set $t^0_{r_k} = s_0$ for all $k \in \{1,\dots,m\}$.
2: Set $t^j_{r_k} = 0$ for all $k \in \{1,\dots,m\}$ and $j \in V \setminus \{0\}$.
3: updated = *true*
4: **while** updated is *true* **do**
5: updated = *false*
6: **for** $k = 1,\dots,m$ **do**
7: **for** $(i,j) \in P_{r_k}$ **do**
8: $\underline{t}^j_{r_k} = \max\{s_j, t^i_{r_k} + t_{ij}, \max_{l\in\{1,\dots,m\}} t^j_{r_l} - u_j\}$
9: **if** $\underline{t}^j_{r_k} > e_j$ **then**
10: **return** "Problem infeasible"
11: **else if** $t^j_{r_k} < \underline{t}^j_{r_k}$ **then**
12: $t^j_{r_k} = \underline{t}^j_{r_k}$
13: updated = *true*
14: **end if**
15: **end for**
16: **end for**
17: **end while**
18: Set $y_j = \max\{s_j, \max_{l\in\{1,\dots,m\}} t^j_{r_l} - u_j\}$ for all $j \in V'$.

In Steps 1 and 2 the values of $t^j_{r_k}$ are initialized. Vehicles are set to leave the depot at time s_0. If client j is not contained in route r_k then, by definition, $t^j_{r_k} = 0$. Hence, we initialize with the value zero and we only update $t^j_{r_k}$ if j is contained in route r_k.

In Step 8 we determine $\underline{t}^j_{r_k}$, which is a lower bound on the final value of $t^j_{r_k}$. Note that $(i,j) \in P_{r_k}$, which implies that route r_k first visits $i \in V$ and then visits $j \in V$ immediately after. For a route to be feasible, we

require that $s_j \leq t^j_{r_k} \leq e_j$ if client j is contained in route r_k. Hence, $t^j_{r_k} \geq s_j$. Furthermore, we require that at least time t_{ij} has passed between the visits to i and j. As the current value of $t^i_{r_k}$ is a lower bound on its final value, it follows that $t^j_{r_k} \geq t^i_{r_k} + t_{ij}$. Time $\max_{l\in\{1,\ldots,m\}} t^j_{r_l}$ is the latest time that j is visited in any of the scenarios. Due to the width of the endogenous time window, the earliest time that j can be visited is $\max_{l\in\{1,\ldots,m\}} t^j_{r_l} - u_j$. Hence, $t^j_{r_k} \geq \max_{l\in\{1,\ldots,m\}} t^j_{r_l} - u_j$. Combining the three lower bounds shows that $\underline{t}^j_{r_k}$ as defined in Step 8 is a lower bound on the final value of $t^j_{r_k}$.

Finally, we need to show that Algorithm 2 produces a feasible solution if the TWAVRP is feasible for the given arc flows. If the algorithm does not return "Problem infeasible" we know that $s_j \leq t^j_{r_k} \leq e_j$ for all $k \in \{1,\ldots,m\}$ and all j contained in route r_k. The requirement that $t^j_{r_k} \geq t^i_{r_k} + t_{ij}$ if $(i,j) \in P_{r_k}$ is enforced in Step 8. It follows that all $r_k = (P_{r_k}, t_{r_k})$ are valid routes.

It remains to show that Constraints (2)–(8) are satisfied. By assumption, Constraints (2), (6), (7) and (8) are already satisfied. In Step 18 we assign values to the y_j variables. It follows immediately that $y_j \geq s_j$ for all $j \in V'$. Because all routes are valid, we have $t^j_{r_k} \leq e_j$ for all $k \in \{1,\ldots,m\}$. It then follows that $y_j = \max\{s_j, \max_{l\in\{1,\ldots,m\}} t^j_{r_l} - u_j\} \leq \max\{s_j, e_j - u_j\}$. By definition of the parameters, $e_j - u_j \geq s_j$ and thus $y_j \leq e_j - u_j$ for all $j \in V'$, which shows that Constraints (5) are satisfied.

The while-loop only terminates when $t^j_{r_k} \geq \underline{t}^j_{r_k}$ for all $k \in \{1,\ldots,m\}$ and all j contained in route k. From Steps 8 and 18 it follows that $\underline{t}^j_{r_k} \geq y_j$, and thus $t^j_{r_k} \geq \underline{t}^j_{r_k} \geq y_j$, if j is contained in route k. It follows that Constraints (3) are satisfied.

By Step 18 we have that $y_j + u_j = \max\{s_j + u_j, \max_{l\in\{1,\ldots,m\}} t^j_{r_l}\} \geq \max_{l\in\{1,\ldots,m\}} t^j_{r_l} \geq t^j_{r_l}$ for all $l \in \{1,\ldots,m\}$ and all j contained in route l. It follows immediately that Constraints (4) are satisfied.

We conclude that Algorithm 2 terminates with a feasible solution if one exists and states that the problem is infeasible otherwise.

B Proof Proposition 3

Proposition. *Let the current integral edge flows be given by $\bar{z}^\omega_{ij}$ for all $(i,j) \in E$ and $\omega \in \Omega$. For each scenario $\omega \in \Omega$, let $E^\omega \subset E$ be a selection of edges with non-zero flow in scenario ω. Furthermore, let each E^ω satisfy a technical condition: if $(0,j) \in E^\omega$ and $\bar{z}^\omega_{0j} = 1$ then E^ω contains an edge adjacent to $(0,j)$. Then, the constraint*

$$\sum_{\omega\in\Omega}\left(\sum_{(i,j)\in E^\omega} z^\omega_{ij} + \sum_{(i,j)\in E^\omega | i,j\in V'} \lambda^{E^\omega}_{ij} z^\omega_{ij}\right) \leq \sum_{\omega\in\Omega}\left(\sum_{(i,j)\in E^\omega} \bar{z}^\omega_{ij} + \sum_{(i,j)\in E^\omega | i,j\in V'} \lambda^{E^\omega}_{ij} \bar{z}^\omega_{ij}\right) - 1 \qquad (11)$$

is violated by an integral edge flow if and only if $z^\omega_{ij} = \bar{z}^\omega_{ij}$ for all $(i,j) \in E^\omega$ and $\omega \in \Omega$.

Proof. *It is trivial that if $z^\omega_{ij} = \bar{z}^\omega_{ij}$ for all $(i,j) \in E^\omega$ and $\omega \in \Omega$, then Constraint (11) is violated. It remains to prove the converse implication: if Constraint (11) is violated for an integral edge flow then it must be that $z^\omega_{ij} = \bar{z}^\omega_{ij}$ for all $(i,j) \in E^\omega$ and $\omega \in \Omega$.*

Partition each E^ω into $u < \infty$ sets $E^\omega_1, E^\omega_2, \ldots, E^\omega_u$ such that every non-depot edge is in the same partition as its adjacent depot edges. By definition, $\lambda^{E^\omega}_{ij}$ is the number of depot edges in E^ω that is adjacent to i or j. Hence, by the choice of partition, $\lambda^{E^\omega_k}_{ij} = \lambda^{E^\omega}_{ij}$ if $(i,j) \in E^\omega_k$ and $\lambda^{E^\omega_k}_{ij} = 0$ otherwise. It follows that Constraint (11) is equivalent to

$$\sum_{\omega\in\Omega}\sum_{k\in\{1,\ldots,u\}}\left(\sum_{(i,j)\in E^\omega_k} z^\omega_{ij} + \sum_{(i,j)\in E^\omega_k | i,j\in V'} \lambda^{E^\omega_k}_{ij} z^\omega_{ij}\right) \leq$$
$$\sum_{\omega\in\Omega}\sum_{k\in\{1,\ldots,u\}}\left(\sum_{(i,j)\in E^\omega_k} \bar{z}^\omega_{ij} + \sum_{(i,j)\in E^\omega_k | i,j\in V'} \lambda^{E^\omega_k}_{ij} \bar{z}^\omega_{ij}\right) - 1. \qquad (22)$$

The partition of E^ω can be chosen such that every E^ω_k belongs to one of the following five classes:

1. $E^\omega_k = \emptyset$.
2. $E^\omega_k = \{(0, j)\}$ *for some* $j \in V'$ *and* $\bar{z}^\omega_{0j} = 2$.
3. $E^\omega_k = \{(i, j)\}$ *for some* $i, j \in V'$ *and* $\bar{z}^\omega_{ij} = 1$.
4. E^ω_k *consists of the edge* (i, j) *and one adjacent depot edge. That is, either* $E^\omega_k = \{(0, i), (i, j)\}$ *and* $\bar{z}^\omega_{0i} = \bar{z}^\omega_{ij} = 1$, *or* $E^\omega_k = \{(0, j), (i, j)\}$ *and* $\bar{z}^\omega_{0j} = \bar{z}^\omega_{ij} = 1$.
5. E^ω_k *consists of the edge* (i, j) *and two adjacent depot edges. That is,* $E^\omega_k = \{(0, i), (0, j), (i, j)\}$ *and* $\bar{z}^\omega_{0i} = \bar{z}^\omega_{0j} = \bar{z}^\omega_{ij} = 1$.

That such a partition is possible follows directly from Observation 1 and the conditions stated in Proposition 3.

Lemma 4. *If E^ω_k belongs to one of the five classes, then*

$$\sum_{(i,j)\in E^\omega_k} z^\omega_{ij} + \sum_{(i,j)\in E^\omega_k} \lambda^{E^\omega_k}_{ij} z^\omega_{ij} \leq \sum_{(i,j)\in E^\omega_k} \bar{z}^\omega_{ij} + \sum_{(i,j)\in E^\omega_k} \lambda^{E^\omega_k}_{ij} \bar{z}^\omega_{ij}. \tag{23}$$

Furthermore, equality holds if and only if $z^\omega_{ij} = \bar{z}^\omega_{ij}$ for all $(i, j) \in E^\omega_k$.

Proof. *For the first class, $E^\omega_k = \emptyset$, the result is trivial. For the second class, Inequality (23) reduces to $z^\omega_{0j} \leq 2$ for which the lemma trivially holds. The third class is also trivial: in this case, Inequality (23) reduces to $z^\omega_{ij} \leq 1$.*

For the fourth class, we assume $E^\omega_k = \{(0, i), (i, j)\}$ and $\bar{z}^\omega_{0i} = \bar{z}^\omega_{ij} = 1$. The proof for $E^\omega_k = \{(0, j), (i, j)\}$ and $\bar{z}^\omega_{0j} = \bar{z}^\omega_{ij} = 1$ is analogous. Inequality (23) now reduces to $z^\omega_{0i} + 2z^\omega_{ij} \leq 3$. If we have $z^\omega_{ij} = 0$, then the left-hand side of this inequality is maximized by choosing $z^\omega_{0i} = 2$, which gives value 2. If we have $z^\omega_{ij} = 1$, then client i is not contained in a single client path in scenario ω, which implies that $z^\omega_{0i} \neq 2$. It follows that the left-hand side is maximized by choosing $z_{0i} = 1$, which gives the value 3. Hence, Inequality (23) is satisfied for all integral edge flows and we have equality if and only if $z_{0i} = 1 = \bar{z}_{0i}$ and $z_{ij} = 1 = \bar{z}_{ij}$.

The fifth class is given by $E^\omega_k = \{(0, i), (0, j), (i, j)\}$ and $\bar{z}^\omega_{0i} = \bar{z}^\omega_{0j} = \bar{z}^\omega_{ij} = 1$. Inequality (23) then reduces to $z^\omega_{0i} + z^\omega_{0j} + 3z^\omega_{ij} \leq 5$. If $z^\omega_{ij} = 0$ then we maximize the left-hand side by choosing $z^\omega_{0i} = z^\omega_{0j} = 2$, which results in the value 4. If $z^\omega_{ij} = 1$ then $z^\omega_{0i} \neq 2$ and $z^\omega_{0j} \neq 2$. Hence, the left-hand side is maximized by choosing $z^\omega_{0i} = z^\omega_{0j} = 1$, which results in the value 5. It follows that Inequality (23) is satisfied for E^ω_k and we have equality if and only if $z_{0i} = 1 = \bar{z}_{0i}$, $z_{0j} = 1 = \bar{z}_{0j}$ and $z_{ij} = 1 = \bar{z}_{ij}$.

By assumption, Constraint (11) is violated. Equivalently, Constraint (22) is violated. By integrality of the edge flow variables this implies

$$\sum_{\omega\in\Omega} \sum_{k\in\{1,\dots,u\}} \left(\sum_{(i,j)\in E^\omega_k} z^\omega_{ij} + \sum_{(i,j)\in E^\omega_k | i,j\in V'} \lambda^{E^\omega_k}_{ij} z^\omega_{ij} \right) \geq \sum_{\omega\in\Omega} \sum_{k\in\{1,\dots,u\}} \left(\sum_{(i,j)\in E^\omega_k} \bar{z}^\omega_{ij} + \sum_{(i,j)\in E^\omega_k | i,j\in V'} \lambda^{E^\omega_k}_{ij} \bar{z}^\omega_{ij} \right). \tag{24}$$

Lemma 4 then implies that for every $\omega \in \Omega$ and $k \in \{1, \dots, u\}$ Inequality (23) holds with equality. By the same lemma this implies that $z^\omega_{ij} = \bar{z}^\omega_{ij}$ for all $(i, j) \in E^\omega_k$. As $E^\omega_1, \dots, E^\omega_u$ partition E^ω, it follows that $z^\omega_{ij} = \bar{z}^\omega_{ij}$ for all $(i, j) \in E^\omega$ and $\omega \in \Omega$.

C Additional tables

Table 5: Detailed comparison between BC and BPC+OS on the Dalmeijer and Spliet (2018) benchmark instances.

		Seconds		Nodes		Optimality gap		Root gap	
Inst.	**Clients**	**BC**	**BPC+OS**	**BC**	**BPC+OS**	**BC**	**BPC+OS**	**BC**	**BPC+OS**
1	10	0	0	1	1	0	0	0	0
2	10	0	0	18	1	0	0	0.28	0
3	10	0	0	1	1	0	0	0	0
4	10	0	0	6	1	0	0	0.14	0
5	10	0	0	42	1	0	0	0.34	0
6	10	0	0	1	1	0	0	0	0
7	10	0	0	1	1	0	0	0	0
8	10	0	0	21	3	0	0	0.96	0.41
9	10	0	0	1	1	0	0	0	0
10	10	0	0	1	1	0	0	0	0
11	15	0	1	1	1	0	0	0	0
12	15	39	13	14,037	93	0	0	2.36	0.07
13	15	3	3	587	7	0	0	1.78	0.88
14	15	0	1	1	1	0	0	0.03	0
15	15	1	0	11	1	0	0	0.17	0
16	15	0	1	1	1	0	0	0.17	0
17	15	0	0	1	1	0	0	0	0
18	15	1	1	124	1	0	0	0.47	0
19	15	1	2	210	1	0	0	0.97	0
20	15	0	1	11	1	0	0	0	0
21	20	1	3	48	3	0	0	1.11	0.15
22	20	9	2	658	11	0	0	0.19	0
23	20	0	1	1	1	0	0	0.12	0
24	20	2	2	58	1	0	0	0.86	0
25	20	7	8	389	5	0	0	1.09	0.34
26	20	0	2	1	1	0	0	0	0
27	20	0	4	1	1	0	0	0	0
28	20	1	1	11	1	0	0	0.08	0
29	20	1	2	1	1	0	0	0.05	0
30	20	0	2	1	1	0	0	0	0
31	25	2	5	20	1	0	0	0.57	0
32	25	1	6	4	1	0	0	0	0
33	25	9	16	395	5	0	0	1.03	0.04
34	25	11	6	391	3	0	0	0.33	0
35	25	6	3	201	5	0	0	0.85	0
36	25	39	5	1,733	3	0	0	1.49	0
37	25	22	51	1,472	31	0	0	0.43	0.25
38	25	10	7	337	3	0	0	0.59	0.05
39	25	7	9	219	1	0	0	0.94	0
40	25	15	8	471	1	0	0	0.62	0
41	30	137	12	3,281	1	0	0	1.99	0
42	30	3600	291	40,701	45	1.51	0	3.34	0.54
43	30	188	194	8,317	23	0	0	1.91	0.30
44	30	61	17	1,296	4	0	0	1.46	0
45	30	111	33	4,158	13	0	0	1.78	0.04
46	30	8	8	165	1	0	0	0.58	0
47	30	18	13	292	1	0	0	0.70	0
48	30	358	55	9,657	27	0	0	1.84	0.19
49	30	930	23	24,789	3	0	0	2.34	0.11
50	30	31	60	710	7	0	0	0.72	0.07
51	35	18	20	114	1	0	0	0.78	0
52	35	14	11	69	1	0	0	0.06	0
53	35	3600	185	32,201	11	1.08	0	2.63	0.36
54	35	3600	55	50,601	7	0.81	0	3.00	0.11
55	35	69	19	983	4	0	0	0.68	0
56	35	3600	40	42,289	7	0.92	0	2.58	0.05
57	35	3600	3600	31,201	170	0.44	0.24	1.85	0.39
58	35	128	127	1,871	15	0	0	0.89	0.16
59	35	245	33	2,485	2	0	0	0.93	0
60	35	443	124	6,655	15	0	0	1.32	0.11
61	40	3600	118	17,282	7	1.61	0	2.81	0.04
62	40	550	34	8,479	1	0	0	1.08	0
63	40	3600	208	36,554	5	0.23	0	1.83	0.09
64	40	3170	129	18,069	4	0	0	1.88	0.01

Table 5 . . .

		Seconds		Nodes		Optimality gap		Root gap	
Inst.	**Clients**	**BC**	**BPC+OS**	**BC**	**BPC+OS**	**BC**	**BPC+OS**	**BC**	**BPC+OS**
65	40	3600	199	37,915	5	0.62	0	2.07	0
66	40	3600	304	32,601	21	1.21	0	2.79	0.20
67	40	3600	354	27,201	19	0.51	0	1.73	0.15
68	40	3600	3600	16,210	78	1.76	0.28	2.88	0.49
69	40	3600	126	17,822	8	1.58	0	2.95	0.17
70	40	3600	356	16,901	5	0.68	0	1.76	0.03
71	45	3600	36	5,741	2	2.23	0	2.97	0
72	45	3600	173	9,201	9	1.95	0	2.66	0.03
73	45	3600	529	14,001	5	0.48	0	1.70	0.02
74	45	3600	113	4,789	7	1.08	0	2.03	0.01
75	45	3600	231	9,501	7	1.25	0	1.97	0.06
76	45	3600	342	6,401	5	2.91	0	3.41	0.06
77	45	3600	3600	12,064	25	2.13	0.29	2.87	0.58
78	45	3600	177	8,501	3	2.47	0	3.08	0
79	45	3600	1719	20,601	27	1.34	0	2.47	0.10
80	45	3600	139	10,097	3	1.38	0	2.15	0
81	50	3600	419	5,898	9	2.24	0	2.87	0.29
82	50	3600	1971	5,701	21	2.10	0	2.76	0.15
83	50	3600	1176	4,001	30	1.94	0	2.25	0.10
84	50	3600	331	4,401	5	2.98	0	3.49	0.07
85	50	3600	1182	3,146	17	2.23	0	2.99	0.06
86	50	3600	620	3,585	14	2.18	0	2.70	0.01
87	50	3600	222	6,611	2	2.31	0	2.87	0
88	50	3600	3600	6,801	44	3.68	1.43	4.39	1.56
89	50	3600	574	4,001	9	1.66	0	2.02	0.05
90	50	3600	193	4,901	9	2.13	0	2.68	0.01

Table 6: Detailed comparison between BC, BC+OS, BPC and BPC+OS on the Dalmeijer and Spliet (2018) benchmark instances.

		Seconds			
Inst.	**Clients**	**BC**	**BC+OS**	**BPC**	**BPC+OS**
1	10	0.0	0.0	0.1	0.1
2	10	0.1	0.0	0.5	0.2
3	10	0.0	0.0	0.3	0.2
4	10	0.1	0.0	0.8	0.1
5	10	0.3	0.0	0.3	0.2
6	10	0.0	0.0	0.1	0.1
7	10	0.0	0.0	0.1	0.1
8	10	0.0	0.0	0.2	0.2
9	10	0.0	0.0	0.3	0.2
10	10	0.0	0.0	0.2	0.1
11	15	0.1	0.1	2.6	1.4
12	15	39.1	9.3	74.5	13.4
13	15	2.6	0.8	139.0	2.9
14	15	0.2	0.1	0.9	0.6
15	15	0.6	0.1	1.1	0.4
16	15	0.3	0.1	0.6	0.6
17	15	0.1	0.1	0.7	0.4
18	15	0.8	0.2	2.3	1.1
19	15	1.3	0.2	1.8	1.7
20	15	0.4	0.1	1.6	0.6
21	20	1.2	0.6	4.4	2.9
22	20	9.0	0.9	4.8	1.5
23	20	0.4	0.2	5.9	1.4
24	20	1.7	0.4	3.7	1.6
25	20	6.9	1.2	3600.0	7.5
26	20	0.2	0.1	4.6	1.8
27	20	0.3	0.2	9.4	3.5
28	20	1.1	0.3	2.5	1.0
29	20	0.5	0.2	4.4	1.9
30	20	0.3	0.1	4.2	2.1
31	25	2.3	1.1	14.0	5.1
32	25	1.3	0.7	28.0	6.1
33	25	9.4	3.5	163.3	15.7

Table 6 ...

		Seconds			
Inst.	**Clients**	**BC**	**BC+OS**	**BPC**	**BPC+OS**
34	25	11.1	1.9	19.0	5.6
35	25	6.1	4.1	9.0	3.3
36	25	39.3	11.7	25.3	4.9
37	25	22.4	4.1	3600.0	50.8
38	25	9.7	1.2	3600.0	7.0
39	25	7.2	3.1	26.6	9.3
40	25	15.2	2.4	29.3	8.3
41	30	137.0	49.4	47.5	12.1
42	30	3600.0	3600.0	3600.0	290.7
43	30	187.6	205.0	3600.0	193.6
44	30	60.6	23.2	49.0	16.7
45	30	110.8	48.2	775.9	33.4
46	30	7.7	2.7	30.0	8.3
47	30	17.8	12.0	44.5	13.0
48	30	357.9	155.8	3600.0	54.6
49	30	930.3	513.1	3600.0	23.4
50	30	30.7	6.8	3600.0	60.3
51	35	18.2	7.7	101.5	20.4
52	35	14.0	1.3	66.3	10.5
53	35	3600.0	3600.0	3600.0	184.5
54	35	3600.0	3600.0	3600.0	55.3
55	35	68.5	14.0	61.8	19.1
56	35	3600.0	3600.0	103.1	39.6
57	35	3600.0	1148.3	3600.0	3600.0
58	35	127.9	15.4	262.7	127.4
59	35	245.1	28.5	147.5	33.3
60	35	443.3	67.9	3600.0	123.5
61	40	3600.0	3600.0	3600.0	117.5
62	40	550.3	153.9	311.9	33.5
63	40	3600.0	3600.0	928.9	207.6
64	40	3169.7	1738.9	993.2	128.5
65	40	3600.0	3600.0	1646.3	199.1
66	40	3600.0	3600.0	3600.0	303.9
67	40	3600.0	3600.0	3600.0	354.2
68	40	3600.0	3600.0	3600.0	3600.0
69	40	3600.0	3600.0	3600.0	125.6
70	40	3600.0	3600.0	3600.0	356.1
71	45	3600.0	3600.0	537.3	36.1
72	45	3600.0	3600.0	2624.6	172.6
73	45	3600.0	3600.0	3600.0	529.2
74	45	3600.0	3600.0	786.6	113.0
75	45	3600.0	3600.0	3600.0	230.8
76	45	3600.0	3600.0	2274.8	342.4
77	45	3600.0	3600.0	3600.0	3600.0
78	45	3600.0	3600.0	198.5	176.5
79	45	3600.0	3600.0	3600.0	1718.7
80	45	3600.0	3600.0	1209.8	138.6
81	50	3600.0	3600.0	3600.0	418.7
82	50	3600.0	3600.0	3600.0	1970.8
83	50	3600.0	3600.0	3600.0	1175.9
84	50	3600.0	3600.0	1127.3	331.2
85	50	3600.0	3600.0	3600.0	1181.5
86	50	3600.0	3600.0	3600.0	619.7
87	50	3600.0	3600.0	3600.0	222.3
88	50	3600.0	3600.0	3600.0	3600.0
89	50	3600.0	3600.0	2604.3	574.3
90	50	3600.0	3600.0	1096.1	192.8

References

Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. Operations Research, 59(5):1269–1283, 2011.

Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. European Journal of Operational Research, 218(1):1–6, 2012.

Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. Operations Research, 46(3):316–329, 1998.

Claudio Contardo, Jean-François Cordeau, and Bernard Gendron. An exact algorithm based on cut-and-column generation for the capacitated location-routing problem. INFORMS Journal on Computing, 26(1):88–102, 2013.

Claudio Contardo, Guy Desaulniers, and François Lessard. Reaching the elementary lower bound in the vehicle routing problem with time windows. Networks, 65(1):88–99, 2015.

Kevin Dalmeijer and Remy Spliet. A branch-and-cut algorithm for the time window assignment vehicle routing problem. Computers & Operations Research, 89(Supplement C):140–152, 2018.

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

Guy Desaulniers, Jacques Desrosiers, and Simon Spoorendonk. Cutting planes for branch-and-price algorithms. Networks, 58(4):301–310, 2011.

Michael Drexl. Synchronization in vehicle routing-a survey of VRPs with multiple synchronization constraints. Transportation Science, 46(3):297–316, 2012.

Chris Groër, Bruce Golden, and Edward Wasil. The Consistent Vehicle Routing Problem. Manufacturing & Service Operations Management, 11(4):630–643, 2009.

Irina Ioachim, Sylvie Gelinas, Francois Soumis, and Jacques Desrosiers. A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. Networks, 31(3):193–204, 1998.

Mads Jepsen, Bjørn Petersen, Simon Spoorendonk, and David Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research, 56(2):497–511, 2008.

Jon Kleinberg and Eva Tardos. Algorithm Design. Pearson Education India, 2006.

Attila A Kovacs, Bruce L Golden, Richard F Hartl, and Sophie N Parragh. Vehicle Routing Problems in Which Consistency Considerations are Important: A Survey. Networks, 64(3):192–213, 2014.

Kunlei Lian. Service Consistency in Vehicle Routing. PhD thesis, University of Arkansas, 2017.

Federico Liberatore, Giovanni Righini, and Matteo Salani. A column generation algorithm for the vehicle routing problem with soft time windows. 4OR, 9(1):49–82, 2011.

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

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

Jorge E Mendoza, Christelle Guéret, Maxim Hoskins, Hubert Lobit, Victor Pillac, Thibaut Vidal, and Daniele Vigo. VRP-REP: the vehicle routing community repository. In Third Meeting of the EURO Working Group on Vehicle Routing and Logistics Optimization (VeRoLog). Oslo, Norway, 2014. URL http://www.vrp-rep.org/.

Artur Pessoa, Marcus Poggi De Aragão, and Eduardo Uchoa. Robust branch-cut-and-price algorithms for vehicle routing problems. In Bruce Golden, S. Raghavan, and Edward Wasil, editors, The vehicle routing problem: Latest advances and new challenges, pages 297–325. Springer, 2008.

Remy Spliet and Guy Desaulniers. The discrete time window assignment vehicle routing problem. European Journal of Operational Research, 244(2):379–391, 2015.

Remy Spliet and Adriana F Gabor. The Time Window Assignment Vehicle Routing Problem. Transportation Science, 49(4):721–731, 2015.

Remy Spliet, Said Dabia, and Tom van Woensel. The Time Window Assignment Vehicle Routing Problem with Time-Dependent Travel Times. Transportation Science, 52(2):261–276, 2018.

Anirudh Subramanyam and Chrysanthos E Gounaris. A Decomposition Algorithm for the Consistent Traveling Salesman Problem with Vehicle Idling. Transportation Science, 52(2):386–401, 2018.