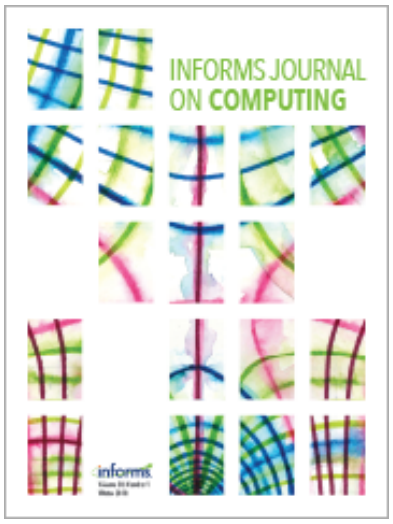


INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Addressing Orientation Symmetry in the Time Window Assignment Vehicle Routing Problem

Kevin Dalmeijer, Guy Desaulniers

To cite this article:
Kevin Dalmeijer, Guy Desaulniers (2021) Addressing Orientation Symmetry in the Time Window Assignment Vehicle Routing Problem. INFORMS Journal on Computing 33(2):495-510. https://doi.org/10.1287/ijoc.2020.0974

http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Vol. 33, No. 2, Spring 2021, pp. 495–510
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

Addressing Orientation Symmetry in the Time Window Assignment Vehicle Routing Problem

Kevin Dalmeijer,[a,b] Guy Desaulniers[c,d]

[a] H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332; [b] Econometric Institute, Erasmus School of Economics, Erasmus University Rotterdam, 3062 PA Rotterdam, Netherlands; [c] Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal, Québec H3T 1J4, Canada; [d] Group for Research in Decision Analysis (GERAD), Montréal, Québec H3T 1J4, Canada

Contact: dalmeijer@gatech.edu, https://orcid.org/0000-0002-4304-7517 (KD); guy.desaulniers@gerad.ca, https://orcid.org/0000-0003-4469-9813 (GD)

Received: July 4, 2018
Revised: March 22, 2019; November 7, 2019; February 14, 2020; March 4, 2020
Accepted: March 9, 2020
Published Online in Articles in Advance: September 9, 2020

https://doi.org/10.1287/ijoc.2020.0974

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 relatively difficult to solve, we observe many similar solutions in which one or more routes have a different orientation, that is, the clients are visited in the reverse order. We introduce an edge-based branching method combined with additional components to eliminate 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 this branching method. Our computational experiments show that addressing orientation symmetry significantly improves our algorithm: The number of nodes in the search tree is reduced by 92.6% on average, and 25 additional benchmark instances are solved to optimality. Furthermore, the resulting algorithm is competitive with the state of the art. 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.

History: Accepted by Antonio Frangioni, Area Editor for Design & Analysis of Algorithms—Continuous.
Supplemental Material: The online supplement is available at https://doi.org/10.1287/ijoc.2020.0974.

Keywords: vehicle routing • time window assignment • symmetry • synchronization • 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 in which 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).

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.

Subramanyam et al. (2018) present a scenario decomposition algorithm to solve strategic time window assignment vehicle routing problems, including the TWAVRP and the DTWAVRP. The decomposition results in multiple vehicle routing problems that can be solved in a "black box" fashion. When applied to the TWAVRP, their algorithm outperforms the BC algorithm by Dalmeijer and Spliet (2018), and instances with up to 50 clients and three demand scenarios are solved to optimality. Additionally, instances with 25 clients are solved to optimality for up to 15 demand scenarios.

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 and driver consistency, that is, 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 a decomposition algorithm.

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 the algorithms by both Spliet and Gabor (2015) and Dalmeijer and Spliet (2018).

We then propose an edge-based branching method combined with additional components to eliminate orientation symmetry from the search tree. The first additional component is an algorithm to find 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. The second additional component is a novel constraint that is used to cut off groups of orientation-symmetric solutions.

For the TWAVRP, branching only on the edge flows is not sufficient to explore the whole search tree as opposed to several other vehicle routing problems in which this is the case (e.g., the symmetric capacitated vehicle routing problem). By combining edge-based branching with the additional components, we are able to solve the TWAVRP while addressing orientation symmetry.

Based on our method to address orientation symmetry, we construct a BPC algorithm to solve the TWAVRP to optimality. Our computational experiments show that addressing orientation symmetry significantly improves our algorithm: the number of nodes in the search tree is reduced by 92.6% on average (from 145.0 to 10.7), and 25 additional benchmark instances are solved to optimality. Furthermore, the resulting algorithm is competitive with the scenario decomposition algorithm by Subramanyam et al. (2018).

Our experiments also show that addressing orientation symmetry improves the BC algorithm by Dalmeijer and Spliet (2018). Finally, we conduct experiments on instances with additional clients and instances with additional demand scenarios.

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 are used in a later section of the paper. In Section 3, we define orientation symmetry and present an edge-based branching method combined with additional components 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 τ_{ij}. Both the costs and the travel times are nonnegative 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 $d_i^\omega \geq 0$. This is a slight generalization of the assumption by Spliet and Gabor (2015), who assume that demand is strictly positive. Let $V'_\omega \subseteq V'$ be the set of clients with nonzero demand, that is, the clients that have to be visited in scenario 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 $d_i^\omega \leq Q$.

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 nonnegative width u_i, which has to be within the exogenous time window. Service must start, but not necessarily complete, within the assigned time window.

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. Route r is feasible in scenario ω if the exogenous time windows and the vehicle capacity are respected, and only clients with nonzero demand are visited (i.e., clients in V'_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 $r \in \mathcal{R}(\omega)$ 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:

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

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

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

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

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

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

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

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

The objective Function (1) is to minimize the expected cost of distribution. Constraints (2) ensure that, in each scenario, every client with nonzero demand is visited exactly once. Note that clients with zero demand are never visited because of our definition of $\mathcal{R}(\omega)$, the set of feasible routes in scenario ω. Constraints (3) and (4) guarantee that, in each scenario, the time of service for each visited 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 this formulation are 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

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

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, $\sum_{(k,l)\in A_p} \tau_{kl} \leq y_j + u_j - y_i$. Analogously, route r' implies that $\sum_{(k,l)\in A_{p'}} \tau_{kl} \leq y_i + u_i - y_j$. 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 in which 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 earlier 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

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

As p and p' are chosen such that Inequality (9) does not hold, it must be that $\sum_{(k,l)\in A_p} x_{kl}^{\omega} < |A_p|$ or $\sum_{(k,l)\in A_{p'}} x_{kl}^{\omega'} < |A_{p'}|$. 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. For example, 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.

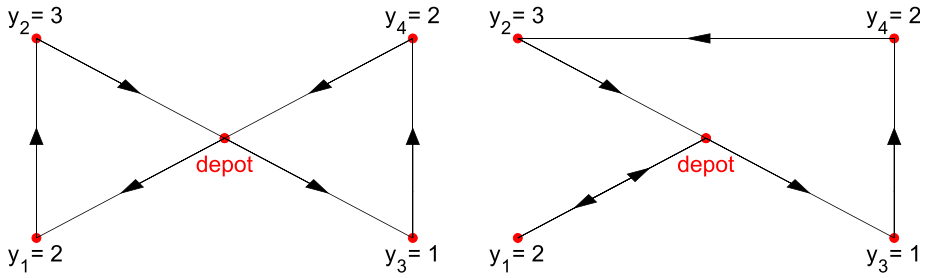
We now present an example of two orientation-symmetric solutions. Consider an instance of the TWAVRP with four clients and two demand scenarios. Every arc has a travel time of one, and all endogenous time window widths are set to zero. Recall that the start of the endogenous time window of client $i \in V'$ is given by y_i.

It can be seen that solution 1 (Figure 1) and solution 2 (Figure 2) are orientation-symmetric solutions. Solution 1 can be turned into solution 2 by reversing the route visiting clients 3 and 4 in scenario 1; reversing the route visiting clients 3, 4, and 2 in scenario 2; and by reassigning the endogenous time windows.

Out of the 40 benchmark instances introduced by Spliet and Gabor (2015), 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 an edge-based branching method, combined with additional components, that ensures that orientation-symmetric solutions are always in the same branch, thereby eliminating orientation symmetry in the

Figure 1. (Color online) Example Solution 1



search tree. In Section 5, we show computationally that addressing orientation symmetry improves algorithmic performance.

3.1. Eliminating Orientation Symmetry

We make the distinction between (directed) arcs and (undirected) edges. An edge between i and j with $i < j$ and $i, j \in \{0\} \cup V'$ is denoted by $[i, j]$. Note that we use square brackets for edges and parentheses for arcs.

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_{ij}^{ω} 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 $z_{13}^{\omega} = x_{13}^{\omega} + x_{31}^{\omega}$, 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_{ij}^{ω}, 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_{ij}^{ω} instead of z_{ij}^{ω}.

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 nondepot edge that has flow closest to 0.5. It can be shown that, if the nondepot 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.

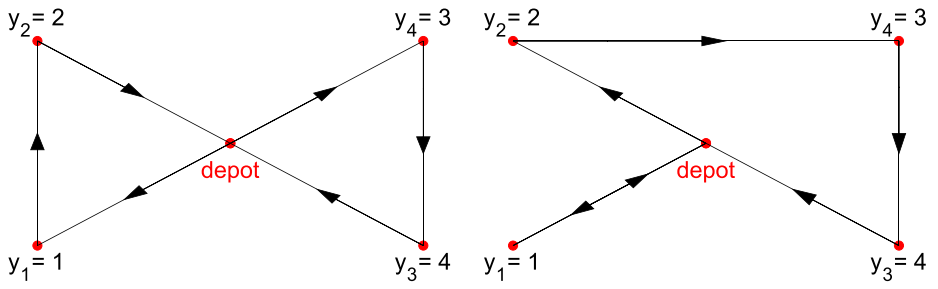
Procedure 1. Processing a Node with Integral Edge Flows and Fractional Arc Flows

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

Figure 2. (Color online) Example Solution 2



Scenario 1 Scenario 2

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 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 any algorithm for the TWAVRP. 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 $k \in \{1,\dots,m\}$, 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 $(o_1,o_2,\dots,o_m) \in \{-1,1\}^m$ 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 Online Supplement 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 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.

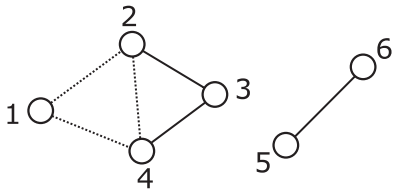
We introduce a *conflict graph* to represent conflicts between the o_k variables resulting from violated path-precedence inequalities. The vertex set of the conflict graph is given by $\{1,\dots,m\}$, in which vertex k corresponds to o_k.

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 one. 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 3 presents an example of a conflict graph that 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 3. Example of a Conflict Graph



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

Observation 2 follows directly from the definition of a conflict-free assignment: If an assignment is *not* conflict-free, then at least one path-precedence inequality is violated. 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.

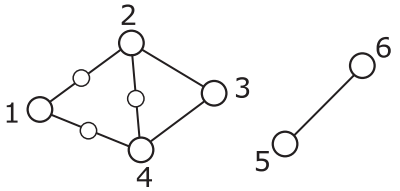
The conflict graph contains a positive edge if both $(o_k, o_l) = (1, -1)$ and $(o_k, o_l) = (-1, 1)$ result in a violated path-precedence inequality. Note that, if the travel times are symmetric, it follows from (9) and (10) that $(o_k, o_l) = (1, -1)$ leads to a violation if and only if $(o_k, o_l) = (-1, 1)$ 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 because of 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 Observation 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 4 shows the result of modifying the conflict graph presented in Figure 3. 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 4. 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.

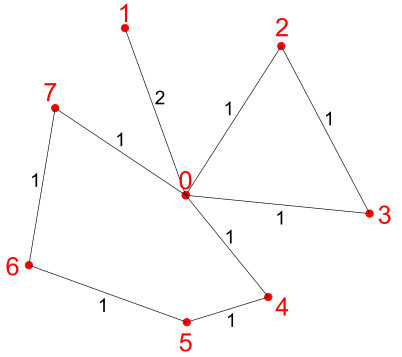
In conclusion, we can now enumerate all two-colorings of the modified conflict graph or, equivalently, enumerate all conflict-free assignments of the original conflict graph. The number of conflict-free assignments is equal to two to the power of the number of connected components of the conflict graph. 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.

Figure 5. (Color online) Example Current Edge Flows $\bar{z}_{ij}^{\omega}$ for a Given Scenario ω



Let the current edge flows be given by $z_{ij}^{\omega} = \bar{z}_{ij}^{\omega}$ for all $[i,j] \in E$ and $\omega \in \Omega$.

Figure 5 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 $z_{01}^{\omega} = x_{01}^{\omega} + x_{1,n+1}^{\omega}$, and hence, in the example, we have $\bar{z}_{01}^{\omega} = 2$. 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 1. For a given subset of edges with nonzero flow, Proposition 1 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 nondepot 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 $\lambda_{ij}^{\hat{E}}$ to be the number of depot edges in $\hat{E}$ that are incident to i or j. That is, $\lambda_{ij}^{\hat{E}} = |\{[0,i], [0,j]\} \cap \hat{E}|$. Given this notation, Proposition 1 can be stated as follows.

Proposition 1. *Let the current integral edge flows be given by $\bar{z}_{ij}^{\omega}$ for all $[i,j] \in E$ and $\omega \in \Omega$. For each scenario $\omega \in \Omega$, let $E^{\omega} \subset E$ be any subset of edges with nonzero flow in scenario ω, satisfying the following technical condition: If $[0,j] \in E^{\omega}$ and $\bar{z}_{0j}^{\omega} = 1$, then E^{ω} contains an edge adjacent to $[0,j]$. Then, the constraint*

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

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

Proof. See Online Supplement B.

To illustrate Proposition 1, we construct an inequality for the example in Figure 5. Note that $\lambda_{ij}^{E^{\omega}}$ can be seen as an additional weight that is assigned to nondepot 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 $(z_{02}^{\omega} + z_{23}^{w} + z_{03}^{w}) + 2z_{23}^{\omega} \leq 4$. The edge $[2,3]$ gets additional weight $(\lambda_{23}^{E^{\omega}} = 2)$ because it is adjacent to two depot edges in E^{ω}. This additional weight is necessary: The constraint $z_{02}^{\omega} + z_{23}^{w} + z_{03}^{w} \leq 2$ (obtained by setting $\lambda_{ij}^{E^{\omega}} = 0$) would cut off the current solution but is not valid as it would also cut off the solution $z_{02}^{\omega} = z_{03}^{\omega} = 2$.

To obtain a constraint that cuts off the current integral edge flows, we can now apply Proposition 1. For each scenario $\omega \in \Omega$, choose E^{ω} to be the set of *all* edges with nonzero flow in scenario ω. Proposition 1 then provides a constraint that can be used in step 5 of procedure 1. For example, the scenario shown in Figure 5 contributes $z_{01}^{\omega} + (z_{02}^{\omega} + z_{23}^{\omega} + z_{03}^{\omega}) + 2z_{23}^{\omega} + (z_{04}^{\omega} + z_{45}^{\omega} + z_{56}^{\omega} + z_{67}^{\omega} + z_{07}^{\omega}) + z_{45}^{\omega} + z_{67}^{\omega}$ to the left-hand side of this constraint and 14 to the right-hand side. The technical condition in Proposition 1 is satisfied because the edges with nonzero 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 because of 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 5, 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 1 provides a valid inequality.

The constraint that we construct based on the conflict is stronger than the constraint we obtain in the case that no conflict is found. This follows from the fact that the latter only cuts off the current integral edge flows although 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, Models (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 $\omega \in \Omega$ and $i \in V'_\omega$, let β_i^ω, $\gamma_i^\omega \geq 0$, and $\delta_i^\omega \leq 0$ be the dual variables corresponding to constraints (2)–(4), respectively. For convenience, let $\pi_i^\omega = \gamma_i^\omega + \delta_i^\omega$. Note that both β_i^ω and π_i^ω are unrestricted. The reduced cost $\bar{c}_r^\omega$ corresponding to the route variable θ_r^ω can now be expressed as

$$\bar{c}_r^\omega = p_w c_r - \sum_{i \in V'_\omega} \beta_i^\omega a_r^i - \sum_{i \in V'_\omega} \pi_i^\omega t_r^i. \tag{12}$$

Recall that $a_r^i = 0$ and $t_r^i = 0$ if client i is not visited by route r. Hence, in Equation (12), we can sum over all clients in V'_ω. The pricing problem is to find a route variable θ_r^ω for which $\bar{c}_r^\omega$ is negative.

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 and linear node costs.

Let $G_\omega = (V_w, A_w)$ be the subgraph of G that is obtained by removing all clients with zero demand and all adjacent arcs. The goal is to find a minimum cost elementary path in G_ω from the starting depot to the ending depot such that both vehicle capacity and the exogenous time windows are respected. Note that we may reduce the size of the arc set A_ω by removing arcs that are infeasible because of capacity or time constraints without affecting any of the feasible paths in G_ω.

To each arc $(i,j) \in A_\omega$, we assign the cost $\bar{c}_{ij}(t^j) = p_w c_{ij} - \beta_j^\omega - \pi_j^\omega t^j$ if $j \in V'_\omega$ and $\bar{c}_{ij}(t^j) = p_w c_{ij}$ otherwise with t^j the time at which vertex j is visited. It follows that a path with negative costs corresponds to a route with negative reduced cost. Note that $\bar{c}_{ij}(t^j)$ depends linearly on t^j, 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 bidirectional labeling algorithm for solving it. To solve the pricing problem of the TWAVRP, we implement a monodirectional variant of their algorithm with a simplified dominance rule.

Labeling algorithms maintain a set of *labels*, each of which 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 $L = (i, \bar{c}(T), d, \tau, U, p)$, where $i \in V_\omega$ 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, τ 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 because of capacity or time constraints, and p is the predecessor label of L—that is, the label that has been extended to obtain L. The components of label L are denoted by $i(L)$, $\bar{c}(L,T)$, $d(L)$, $\tau(L)$, $U(L)$, and $p(L)$, respectively.

The function $\bar{c}(L,T)$ is defined on the domain $T \in [\tau(L), e_{i(L)}]$, where $e_{i(L)}$ is the ending time of the exogenous time window of vertex $i(L)$, the end vertex of the path associated with L. It is shown by Ioachim et al. (1998) that $\bar{c}(L,T)$ is convex, nonincreasing, 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 $(i,j) \in A_\omega$ if $j \notin U(L)$. For the extended label, $i(L')$, $d(L')$, $\tau(L')$, $U(L')$, and $p(L')$ are given by the following straightforward extension functions:

$$i(L') = j, \tag{13}$$

$$d(L') = d(L) + d_j^\omega, \tag{14}$$

$$\tau(L') = \max\{s_j, \tau(L) + \tau_{ij}\}, \tag{15}$$

$$U(L') = U(L) \cup \{j\} \cup \{k \in V'_\omega \mid d(L') + d_k^\omega > Q \vee \tau(L') + \tau_{jk} > e_k\} \tag{16}$$

$$p(L') = L. \tag{17}$$

The label L' is feasible if $d(L') \leq Q$ and $\tau(L') \leq e_j$.

Next, we give the extension function for $\bar{c}(L', T)$. Let $T^* \in [\tau(L'), e_j]$ 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^j then vertex i must be visited at or before time $t^j - \tau_{ij}$. Furthermore, vertex i must be visited at or before time e_i. It follows that

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

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 $t^j = \min\{T, T^*\}$. It follows that the extension function for $\bar{c}(L', T)$ is given by

$$\bar{c}(L', T) = \begin{cases} \bar{c}\big(L, \min\{T - \tau_{ij}, e_j\}\big) + \bar{c}_{ij}(T) \\ \qquad \text{if } \tau(L') \le T \le T^* \\ \bar{c}\big(L, \min\{T^* - \tau_{ij}, e_j\}\big) + \bar{c}_{ij}(T^*) \\ \qquad \text{if } T^* \le T \le e_j. \end{cases} \tag{19}$$

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. $\bar{c}(L_1, T) \le \bar{c}(L_2, T) \forall T \in [\tau(L_1), e_{i(L_1)}] \cap [\tau(L_2), e_{i(L_2)}]$.
iii. $d(L_1) \le d(L_2)$.
iv. $\tau(L_1) \le \tau(L_2)$.
v. $U(L_1) \subseteq U(L_2)$.

Conditions (i)–(v) imply that L_2 is not Pareto-optimal for any $T \in [\tau(L_1), e_{i(L_1)}] \cap [\tau(L_2), e_{i(L_2)}]$ (Liberatore et al. 2011). By condition (i), we have that $e_{i(L_1)} = e_{i(L_2)}$ and by condition (iv), we have that $\tau(L_1) \le \tau(L_2)$. Hence, conditions (i) and (iv) together imply that $[\tau(L_1), e_{i(L_1)}] \cap [\tau(L_2), e_{i(L_2)}] = [\tau(L_2), e_{i(L_2)}]$. It follows that L_2 is not Pareto-optimal for any $T \in [\tau(L_2), e_{i(L_2)}]$, which is the complete domain of $\bar{c}(L_2, T)$. As such, label L_2 can be eliminated.

With our dominance rule, label L_1 can only dominate label L_2 if $\bar{c}(L_1, T) \le \bar{c}(L_2, T)$ 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 $[\tau(L_1), e_{i(L_1)}]$, 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 because of 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. (2014). Both have been used to solve various vehicle routing problems. For example, 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 vehicle routing problem with time windows by Contardo et al. (2015).

We initialize the *ng*-route relaxation with neighborhoods of size 10. As proposed by Roberti and Mingozzi (2014), we allow for dynamically adding clients to these neighborhoods to further eliminate cycles. The number of times that a client can be added to a neighborhood is limited to five per scenario. If cycles remain after increasing the neighborhoods, we add at most 30 SDCs per scenario as long as they are violated by at least 0.05.

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. The reduced cost of an arc $(i, j) \in A_\omega$ in scenario $\omega \in \Omega$ is given by $\bar{c}_{ij}(t^j) = p_w c_{ij} - \beta_j^\omega - \pi_j^\omega t^j$ if $j \in V'_\omega$ and $\bar{c}_{ij}(t^j) = p_w c_{ij}$ otherwise. For each vertex, we sort the incoming and outgoing arcs by reduced cost, ignoring the time-dependent term. Then, we keep for each vertex the ξ incoming arcs with the least reduced cost and the ξ outgoing arcs with the least reduced cost, where ξ is a preset parameter equal to 10 and increased to 15 if no negative reduced-cost columns can be found.

Second, 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

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

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:

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

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

The 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.

Each time that the master problem is solved to optimality, we first add all capacity cuts with a violation of at least 0.05. After adding the capacity cuts, the master problem is resolved. If no capacity cuts can be added, we try increasing the ng-route relaxation neighborhoods and adding SDCs as discussed in Section 4.2. If neither can be done, we add per iteration at most 10 3SR-inequalities with a minimum violation of 0.1.

We do not add precedence inequalities as valid inequalities. The precedence inequalities are used in Section 3.2.1 to create a conflict graph for the restricted TWAVRP and, in Section 3.3, to derive stronger constraints to cut off integral edge flows. Preliminary experiments showed that if we already address orientation symmetry, 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 use our edge-based branching method combined with additional components as introduced in Section 3.1 to address orientation symmetry. 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.

To address orientation symmetry as in 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 3.30 GHz processor and 16 GB 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 10 instances with 10 clients, 10 instances with 15 clients, etc., up to 50 clients. These clients are uniformly distributed over a square with sides of five hours. The starting and ending depots are both located at the center of the square. The travel costs and 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 one sixth of the vehicle capacity. The exogenous time windows have a width of 10.8 hours on average, and 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 is available on VRP-REP.

5.2. Comparison with Other Algorithms

We first test our BPC algorithm on the benchmark instances used by Dalmeijer and Spliet (2018) and we compare with their results. Dalmeijer and Spliet (2018) perform their computational experiments on an Intel i7 3.5 GHz processor.

The results are summarized in Table 1, and the full table is available as Online Supplement 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 3,600 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 (Online Supplement C). Additionally, 29 instances that could not be solved by Dalmeijer and Spliet (2018) are now solved to optimality. Only four of the benchmark instances remain unsolved.

BC and BPC+OS use different strategies to solve the TWAVRP. BC has a relatively weak LP relaxation that is easy to calculate; BPC+OS relies on a strong bound that takes more computational effort to determine. It is, thus, no surprise that BPC+OS explores fewer 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

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	1,531.7	421.4	16,846.9	23.3	0.33	0.02	1.47	0.12	6	9
40	3,252.0	542.6	22,903.4	15.3	0.82	0.03	2.18	0.12	2	9
45	3,600.0	705.8	10,089.7	9.3	1.72	0.03	2.53	0.09	0	9
50	3,600.0	1,028.7	4,904.6	16.0	2.35	0.14	2.90	0.23	0	9
Average	1,394.1	309.5	7,358.9	10.7	0.60	0.02	1.39	0.10	57/90	86/90

with the 1.39% average root gap for BC, which requires 7,358.9 nodes on average to close the gap.

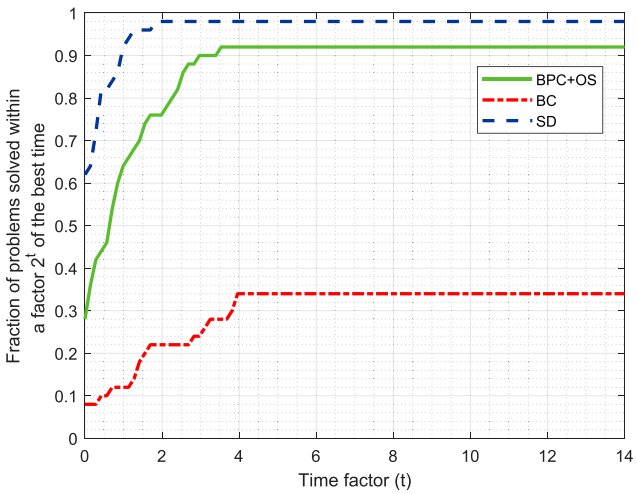
While this paper was under review, Subramanyam et al. (2018) published a scenario decomposition algorithm for the TWAVRP that outperforms the algorithm by Dalmeijer and Spliet (2018). We briefly compare the results for BPC+OS with the results for the scenario decomposition algorithm, which we refer to as SD. The computational experiments reported by Subramanyam et al. (2018) are based on the same benchmark instances and are performed on an Intel Xeon E5-2687W 3.1 GHz processor using a single thread. For convenience, we have included the reported solution times in Online Supplement D.

For SD, it takes 207.2 seconds on average to solve the benchmark instances to optimality. In total, 89 out of the 90 instances are solved to optimality within one hour of computation time. For BPC+OS, the average solution time is 309.5 seconds, and 86 instances can be solved to optimality. SD has the best solution time for 69 instances, and BPC+OS has the best solution time for 24 instances. Note that the numbers do not sum to 90 because of draws.

Following Subramanyam et al. (2018), we also compare BPC+OS, BC, and SD using performance profiles (Dolan and Moré 2002). Figure 6 presents performance profiles for the benchmark instances with at least 30 clients. The performance profile of an algorithm gives the percentage of problems for which this algorithm would be the fastest given that it were 2^t times faster than it currently is. The instances with less than 30 clients are left out to prevent that short solution times obscure the figure.

It is important to note that, even if one performance profile is completely above another, this does not imply that the corresponding algorithm is strictly better on all instances. For $t = 0$, we observe that SD is currently the fastest algorithm for 62% of the instances, and BPC+OS is the fastest algorithm for 28% of the instances. For $t \to \infty$, the curves converge to the percentage of instances that could be solved within one hour of computation time. Here we see a major difference in performance between BC (34%) on one hand and BPC+OS (92%) and SD (98%) on the other hand.

Figure 6. (Color online) Performance Profiles Comparing BPC+OS, BC, and SD on the Dalmeijer and Spliet (2018) Benchmark Instances with at Least 30 Clients



We conclude that our BPC algorithm is competitive with the state of the art when orientation symmetry is properly addressed. Combining BPC+OS and SD to get the best of both worlds is not trivial and may be an interesting topic for future research.

Finally, note that there are still opportunities to improve the performance of BPC+OS. Subramanyam et al. (2018) use a branch-price-and-cut algorithm to solve the scenario-specific VRPTW subproblems. To speed up this part of the algorithm, several elements as described in Pecin et al. (2017) are introduced, including *bidirectional labeling*, *variable fixing*, *route enumeration*, and *limited-memory subset row cuts*. The same elements can be included in BPC+OS, which may improve performance. Other recent enhancements are discussed by Pessoa et al. (2019).

5.3. Effect of Addressing Orientation Symmetry

In this section, we consider the isolated effect of addressing orientation symmetry. To this end, we also adapt BC to address orientation symmetry (denoted by BC+OS), and we test BPC+OS without addressing orientation symmetry (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 addressing orientation symmetry. The four algorithms are compared in Table 2, which is a summary of Online Supplement D.

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 1,394.1 seconds to 1,327.1 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 (Online Supplement D).

For BPC, addressing orientation symmetry clearly has a bigger effect: The average solution time decreases from 1,366.8 seconds to 309.5 seconds. Furthermore, we see that the number of instances that can be solved to optimality increases from 61 to 86.

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	1,894.7	70.6	9	9	5	10
35	1,531.7	1,208.3	1,514.3	421.4	6	7	6	9
40	3,252.0	3,069.3	2,548.0	542.6	2	2	4	9
45	3,600.0	3,600.0	2,203.2	705.8	0	0	6	9
50	3,600.0	3,600.0	3,002.8	1,028.7	0	0	3	9
Average	1,394.1	1,327.1	1,366.8	309.5	57/90	58/90	61/90	86/90

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 addressing orientation symmetry 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, and 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 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.

Based on Table 3, we conclude that our algorithm cannot consistently solve the benchmark instances with more than 50 clients. We manage to solve 4 out of the 10 instances with 55 clients, 2 out of the 10 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 and also because of 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.

Table 3. Computational Results for BPC+OS on Instances with 55 up to 65 Clients and Three Demand Scenarios

Instance	Clients	Seconds	Nodes	Optimality gap	Root gap
91	55	3,600.0	17	—	—
92	55	3,600.0	17	—	—
93	55	3,600.0	38	—	—
94	55	3,600.0	7	—	—
95	55	306.0	10	0	0.01
96	55	2,082.7	9	0	0.06
97	55	1,938.0	21	0	0.09
98	55	3,600.0	5	—	—
99	55	3,216.5	25	0	0.14
100	55	3,600.0	8	—	—
101	60	3,600.0	16	—	—
102	60	2,042.6	7	0	0.04
103	60	3,600.0	5	—	—
104	60	3,600.0	15	—	—
105	60	3,600.0	19	—	—
106	60	358.4	1	0	0
107	60	3,600.0	5	—	—
108	60	3,600.0	16	—	—
109	60	3,600.0	5	—	—
110	60	3,600.0	14	—	—
111	65	3,600.0	2	—	—
112	65	3,600.0	13	—	—
113	65	3,600.0	3	—	—
114	65	3,600.0	7	—	—
115	65	2,742.1	7	0	0.02
116	65	3,600.0	7	—	—
117	65	3,600.0	12	—	—
118	65	3,600.0	3	—	—
119	65	3,600.0	13	—	—
120	65	3,600.0	7	—	—

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	Three scenarios	Five scenarios	Seven scenarios	Three scenarios	Five scenarios	Seven scenarios
10	0.2	0.5	0.9	10	10	10
15	2.3	364.2	1,033.4	10	9	8
20	2.5	14.7	56.0	10	10	10
25	11.6	398.2	1,294.7	10	9	7
30	70.6	514.5	1,281.1	10	9	8
35	421.4	899.5	2,531.8	9	9	5
40	542.6	1,547.0	2,926.9	9	7	5
45	705.8	2,507.2	3,308.5	9	5	1
50	1,028.7	3,427.2	3,600.0	9	1	0
Average	309.5	1,074.8	1,781.5	86/90	69/90	54/90

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, and the full table is available as Online Supplement E. For two of the instances with five scenarios, CPLEX reported that one of the linear programs could not be solved because of numerical issues. These instances have been reported as unsolved with a solution time of 3,600 seconds.

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 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 an edge-based branching method combined with additional components that eliminates orientation symmetry from the search tree. We then present a branch-price-and-cut algorithm to solve the TWAVRP while addressing orientation symmetry.

Our computational experiments suggest that addressing orientation symmetry significantly improves the BPC algorithm. On the benchmark set, the average solution time decreases from 1,366.8 seconds to 309.5 per instance, and 25 additional instances are solved to optimality. Addressing orientation symmetry also greatly reduces the number of nodes in the search tree: The average number of nodes per instance decreases from 145.0 to 10.7 or by 92.6%. The resulting algorithm is competitive with the scenario decomposition algorithm by Subramanyam et al. (2018).

Our experiments also show that addressing orientation symmetry improves the BC algorithm by Dalmeijer and Spliet (2018). Finally, we report computational results on instances with additional clients and instances with additional demand scenarios.

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. The algorithm that we propose for the restricted TWAVRP (Section 3.2) is based on a conflict graph. By redefining the conflict graph, the same algorithm can be used to solve the restricted version of other problems. Proposition 1 provides a general constraint to cut off integer edge flows, which is not unique to the TWAVRP. Given the benefit of addressing orientation symmetry in the TWAVRP, applying our method to other problems is an interesting direction for further research.

References

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

Baldacci R, Mingozzi A, Roberti R (2012) Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *Eur. J. Oper. Res.* 218(1):1–6.

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):316–329.

Contardo C, Cordeau JF, Gendron B (2014) An exact algorithm based on cut-and-column generation for the capacitated location-routing problem. *INFORMS J. Comput.* 26(1):88–102.

Contardo C, Desaulniers G, Lessard F (2015) Reaching the elementary lower bound in the vehicle routing problem with time windows. *Networks* 65(1):88–99.

Dalmeijer K, Spliet R (2018) A branch-and-cut algorithm for the time window assignment vehicle routing problem. *Comput. Oper. Res.* 89:140–152.

Desaulniers G, Desrosiers J, Spoorendonk S (2011) Cutting planes for branch-and-price algorithms. *Networks* 58(4):301–310.

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 Sci.* 42(3):387–404.

Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. *Math. Programming* 91(2):201–213.

Drexl M (2012) Synchronization in vehicle routing—A survey of VRPs with multiple synchronization constraints. *Transportation Sci.* 46(3):297–316.

Groër C, Golden BL, Wasil E (2009) The consistent vehicle routing problem. *Manufacturing Service Oper. Management* 11(4): 630–643.

Ioachim I, Gélinas S, Soumis F, Desrosiers J (1998) A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks* 31(3):193–204.

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

Kleinberg J, Tardos É (2006) *Algorithm Design* (Addison-Wesley, Boston).

Kovacs AA, Golden BL, Hartl RF, Parragh SN (2014) Vehicle routing problems in which consistency considerations are important: A survey. *Networks* 64(3):192–213.

Lian K (2017) Service consistency in vehicle routing. Unpublished doctoral dissertation, University of Arkansas, Fayetteville.

Liberatore F, Righini G, Salani M (2011) A column generation algorithm for the vehicle routing problem with soft time windows. *4OR* 9(1):49–82.

Lysgaard J (2003) CVRPSEP: A package of separation routines for the capacitated vehicle routing problem. Working paper, Aarhus School of Business, Denmark.

Lysgaard J, Letchford AN, Eglese RW (2004) A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Math. Programming* 100(2):423–445.

Mendoza JE, Guéret C, Hoskins M, Lobit H, Pillac V, Vidal T, Vigo D (2014) VRP-REP: The vehicle routing community repository. *3rd Meeting EURO Working Group Vehicle Routing Logistics Optim. (VeRoLog)* (Oslo, Norway).

Pecin D, Pessoa A, de Aragão MP, Uchoa E (2017) Improved branch-cut-and-price for capacitated vehicle routing. *Math. Programming Comput.* 9(1):61–100.

Pessoa A, de Aragão MP, Uchoa E (2008) Robust branch-cut-and-price algorithms for vehicle routing problems. Golden BL, Raghavan S, Wasil E, eds. *The Vehicle Routing Problem: Latest Advances and New Challenges* (Springer, New York), 297–325.

Pessoa A, Sadykov R, Uchoa E, Vanderbeck F (2019) A generic exact solver for vehicle routing and related problems. Lodi A, Nagarajan V, eds. *Integer Programming and Combinatorial Optimization* (Springer, Cham, Switzerland), 354–369.

Roberti R, Mingozzi A (2014) Dynamic ng-path relaxation for the delivery man problem. *Transportation Sci.* 48(3):413–424.

Spliet R, Desaulniers G (2015) The discrete time window assignment vehicle routing problem. *Eur. J. Oper. Res.* 244(2):379–391.

Spliet R, Gabor AF (2015) The time window assignment vehicle routing problem. *Transportation Sci.* 49(4):721–731.

Spliet R, Dabia S, van Woensel T (2018) The time window assignment vehicle routing problem with time-dependent travel times. *Transportation Sci.* 52(2):261–276.

Subramanyam A, Gounaris CE (2018) A decomposition algorithm for the consistent traveling salesman problem with vehicle idling. *Transportation Sci.* 52(2):386–401.

Subramanyam A, Wang A, Gounaris CE (2018) A scenario decomposition algorithm for strategic time window assignment vehicle routing problems. *Transportation Res. Part B: Methodological* 117:296–317.