European Journal of Operational Research 284 (2020) 906–919

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Production, Manufacturing, Transportation and Logistics

Exact algorithms for the multi-pickup and delivery problem with time windows

Imadeddine Aziez[a], Jean-François Côté[a,*], Leandro C. Coelho[a,b]

[a] CIRRELT and Université Laval, Québec, Canada
[b] Canada Research Chair in Integrated Logistics, Canada

ARTICLE INFO

Article history:
Received 30 May 2019
Accepted 20 January 2020
Available online 25 January 2020

Keywords:
Vehicle routing problem
Multi-pickup and delivery problem
Sequential ordering problem
Branch-and-cut

ABSTRACT

In the multi-pickup and delivery problem with time windows (MPDPTW) a set of vehicles must be routed to satisfy a set of client requests between given origins and destinations. A request is composed of several pickups of different items, followed by a single delivery at the client location. This paper introduces two new formulations for the MPDPTW, the 2-index formulation, and the asymmetric representatives formulation. In addition, we also present an existing 3-index formulation for this problem and improve it by means of several preprocessing and valid inequalities. We solve the problem exactly via a branch-and-cut algorithm. We introduce several families of valid inequalities to strengthen the LP relaxations of the proposed formulations. Computational results are reported on different types of instances to firstly highlight the advantage of adding different families of valid inequalities then to compare the performance of the different formulations presented in this paper. While the heuristic and exact algorithms of the literature prove optimality for 16 instances containing up to 50 nodes, we prove optimality for 41 instances for cases containing up to 100 nodes from the existing benchmark set.

1. Introduction

In the multi-pickup and delivery problem, a request is composed of several pickups to be delivered to one location. Vehicles can perform pickups of different requests in any sequence, as long as all pickups of a single request are performed prior to its delivery. This problem finds many practical applications, for example, in food delivery. Online food delivery platforms such as Uber eats, Seamless and JUST EAT allow customers to order their favorite dishes are multiplying, and the service they provide to customers consists of allowing them to order food from different restaurants; customers call a company and request several dishes from different restaurants. The company must then pick up all dishes before delivering them to the customer's home. Obviously, the company can combine orders of different customers in the same vehicle trip. Some of these locations (either pickups or deliveries) can have time windows (TWs). Since these requests consist of small orders, their size does not impose capacity restrictions at the vehicle. Practical examples and a review of distribution problems at large can be found in Coelho, Renaud, and Laporte (2016).

The only work dealing with the multi-pickup and delivery problem with time windows (MPDPTW) is that of Naccache, Côté, and Coelho (2018) who developed a hybrid adaptive large neighborhood search (ALNS) with improvement operations and proposed a mixed-integer formulation for the problem which was then used to solve the problem via branch-and-bound. The MPDPTW shares many characteristics with problems previously studied in the literature, namely the pickup and delivery problem with time windows (PDPTW) and the sequential ordering problem (SOP). These are briefly reviewed next.

The PDPTW is a generalization of the vehicle routing problem with time windows (VRPTW) in which a customer request is composed of an origin-destination pair. The origin is the location where a given item is picked up and the destination is where this item must be delivered. The PDPTW can be seen as a special case of the MPDPTW. The PDPTW has various practical applications such as door-to-door passenger transportation, freight transportation, urban courier services, and school bus routing. The PDPTW can be modeled using different mathematical formulations such as the classical three-index formulation (see, e.g, Cordeau, 2006, Ropke & Cordeau, 2009), the classical two-index formulation (see, e.g, Ropke, Cordeau, & Laporte, 2007), the set partitionning formulation (see, e.g, Baldacci, Bartolini, & Mingozzi, 2011, Ropke & Cordeau, 2009). Furtado, Munari, and Morabito (2017) proposed a new two-index formulation for the PDPTW by extending the classical two-index formulation of the VRPTW. Dahle, Andersson, Christiansen, and Speranza (2019) introduced the PDPTW

* Corresponding author.
E-mail address: jean-francois.cote@fsa.ulaval.ca (J.-F. Côté).

https://doi.org/10.1016/j.ejor.2020.01.040
0377-2217/Crown Copyright © 2020 Published by Elsevier B.V. All rights reserved.

and occasional drivers, and proposed a load and a flow formulation for this problem. Heuristic algorithms for the PDPTW are the ALNS (Pisinger & Ropke, 2007), the parallel neighborhood descent (Subramanian, Drummond, Bentes, Ochi, & Farias, 2010), the particle swarm optimization (Ai & Kachitvichyanukul, 2009; Goksal, Karaoglan, & Altiparmak, 2013), the population based metaheuristic (Cherkesly, Desaulniers, & Laporte, 2015), the parallel simulated annealing (Wang, Mu, Zhao, & Sutherland, 2015), the granular tabu search (Goeke, 2019). Exact algorithms include the branch-and-cut of Ropke et al. (2007), the branch-and-cut-and-price (Bettinelli, Cacchiani, Crainic, & Vigo, 2019; Ropke & Cordeau, 2009), the set partitioning-based algorithm of Baldacci et al. (2011), the branch-price-and-cut of Cherkesly, Desaulniers, and Laporte (2014) and the branch-price-and-cut with an ad hoc dominance criterion of Veenstra, Cherkesly, Desaulniers, and Laporte (2017).

The SOP is a related problem which aims at solving an asymmetric traveling salesman problem with precedence constraints, meaning that nodes have a partial order imposed to their visits (Escudero, 1988). This problem also models real-world applications in manufacturing and in transportation (Ezzat, Abdelbar, & Wunsch, 2014). Several methods exist to solve the SOP, including local searches (Savelsbergh, 1990), branch-and-cut (Ascheuer, Jünger, & Reinelt, 2000), parallel sequential algorithm (Guerriero & Mancini, 2003), genetic algorithm (Seo & Moon, 2003), and other mathematical programming tools (Letchford & Salazar-González, 2016). Dual bounds were obtained by lagrangian relaxation by Alonso-Ayuso, Detti, Escudero, and Ortuño (2003).

In transportation problems, when multiple homogeneous vehicles are considered, symmetric solutions can be obtained by reassigning the same set of customers to different vehicles. This is known to yield weak relaxations and to a repetitive branch-and-bound tree (Jans & Desrosiers, 2013). To overcome this problem, we develop a new formulation for distribution problems in which this kind of symmetry is no longer present. This new model is called the asymmetric representatives formulation (ARF).

The goal of this paper is to provide the first exact algorithm for the MPDPTW, providing the first dual bounds for the problem and obtaining tight solutions for large instances. To this end, we propose three formulations for the problem and design a state-of-the-art branch-and-cut algorithm to solve them. Several families of cuts are adapted from the literature and improved to tackle this difficult problem. We compare the formulations in terms of efficiency, and we show the value of introducing valid inequalities and preprocessing in strengthening their relaxations.

The remainder of the paper is organized as follows. Section 2 provides a formal description of MPDPTW, and introduces three mixed integer formulations in Section 3. Section 4 introduces several families of valid inequalities used in the branch-and-cut algorithm, which is then described in Section 5, along with preprocessing techniques and separation procedures. Computational experiments are presented in Section 6 and conclusions follow in Section 7.

2. Problem description

The MPDPTW is defined on a graph $G = (V, A)$ in which the set of vertices $V = P \cup D \cup \{0, p+n+1\}$. The set $P = \{1, \dots, p\}$ defines the pickup nodes, and $D = \{p+1, \dots, p+n\}$ is the set of delivery nodes where $|D| = n$ and $p \geq n$. The starting and ending depots are the same and they are defined by the nodes 0 and $p+n+1$ in G. Let $R = \{r_1, \dots, r_n\}$ be the set of requests to be routed. Each request $r \in R$ is represented by a set of pickup nodes $P_r \subseteq P$ and one delivery node $d_r \in D$. All the pickup nodes must be visited before the delivery node for each request $r \in R$. For each pair of requests $r_i, r_j \in R$ such that $r_i \neq r_j$, we have that $P_{r_i} \cap P_{r_j} = \emptyset$. Let $N = P \cup D$ be the set of customer nodes. Let $r(i)$ be the request associated with node $i \in N$. An uncapacitated fleet of m identical vehicles in the set K is available to serve the requests. Each vertex $i \in V$ has a service time s_i and a time window $[a_i, b_i]$ allowing the vehicle to arrive at i prior to a_i but waiting until a_i to service the node, and service must start not later than b_i. Vehicles depart from the depot at a_0 and must return not later than b_0.

The set of arcs is $A = V \times V$ minus arcs that lead to infeasible solutions. A cost $c_{ij} \geq 0$ and a travel time $t_{ij} \geq 0$ are associated with each arc $(i, j) \in A$, with $c_{ij} = t_{ij}$. Travel times satisfy the triangle inequality, meaning that $t_{ij} \leq t_{ik} + t_{kj}$ $\forall i, j, k \in V$. The set of omitted arcs may include each arc (i, j) if: (i) i is a delivery node and j is one of its pickup nodes,(ii) $a_i + s_i + t_{ij} > b_j$, (iii) i is the start depot and j is a delivery node, (iv) i is a pickup node and j is the end depot, (v) $i \in V \backslash \{0\}$ and j is the start depot, (vi) i is the end depot and $j \in V \backslash \{p+n+1\}$, (vii) $i = j$. Let $A^+(i)$ be the set of nodes j such that there is an arc from i to j, that is, $A^+(i) = \{j \in V | (i, j) \in A\}$. Similarly, $A^-(i) = \{j \in V | (j, i) \in A\}$.

A solution to the problem minimizes the routing cost and assigns all requests to the vehicles, guaranteeing that a request is served by a single vehicle.

It is important to define the PDPTW in order to highlight the similarities and the differences with the MPDPTW. In the PDPTW, a set of vehicles must be routed to service a set of origin-destination requests. Vehicles must start and end at the same depot and respect pairing and precedence constraints. The PDPTW is defined on a graph $G' = (V', A')$ in which the set of vertices is $V' = P' \cup D' \cup \{0, 2n+1\}$ and the set of arcs is $A' = V' \times V'$. Nodes 0 and $2n+1$ represent the start and end depots, the set $P' = \{1, \dots, n\}$ defines the pickup nodes, and $D' = \{n+1, \dots, 2n\}$ is the set of delivery nodes. Unlike the MPDPTW, in the PDPTW each request i is associated with only one pickup node i and a delivery node $n+i$. Furthermore, restrictions on the capacity of the vehicles can also encountered. In Section 2.1, we prove that it is possible to transform a MPDPTW into a PDPTW, which can then be solved by existing algorithms.

Fig. 1 illustrates an example of the problem with three requests. In this example, Route 1 serves Request 1 and the items located in nodes p_1 and p_2 must be collected before delevering them to node d_1, guaranteeing that precedence constraints are respected. The same rule applies to Requests 2 and 3 which are served by Route 2. Note that the pickup nodes of each request can be visited in any order as long as they are visited prior to the delivery node.

2.1. The transformation to a PDPTW

The MPDPTW can be easily transformed into a PDPTW, and in this case, it can be solved by existing algorithms for the PDPTW. This transformation is possible by considering each request $r \in R$ with the set of pickup nodes $P_r \subseteq P$ and the delivery node $d_r \in D$. We start by making $|P_r| - 1$ copies of d_r to create the set $D_r = \{d_r^1, d_r^2, \dots, d_r^{|P_r|}\}$ of identical delivery nodes for each request $r \in R$. Travel time between all the nodes in D_r equals zero. Furthermore, each one of the pickup nodes in the set P_r is assigned to one delivery node in D_r, resulting in several origin-destination requests.

If the requests created using the nodes of the request r are visited by the same vehicle and the delivery nodes in the set D_r are visited afterwards, the solution of the PDPTW corresponds to a solution of the MPDPTW. In order to ensure that the delivery nodes in the set D_r are visited consecutively in the solution of the PDPTW, we set a high cost on the arcs $(i, j) \in A$ such that $i = d_r^1$ and $j \in V \backslash D_r$, and on the arcs $(i, j) \in A$ such that $i \in V \backslash D_r$ and $j \in D_r \backslash \{d_r^1\}$. In addition we set a cost of zero on the arcs $(i, j) \in A$ such that i, $j \in D_r$.

Fig. 2 illustrates an example of transforming the MPDPTW to a PDPTW. This example is based on that of Fig. 1. For each request in Fig. 1, we create a copy of the delivery node, then each one of the

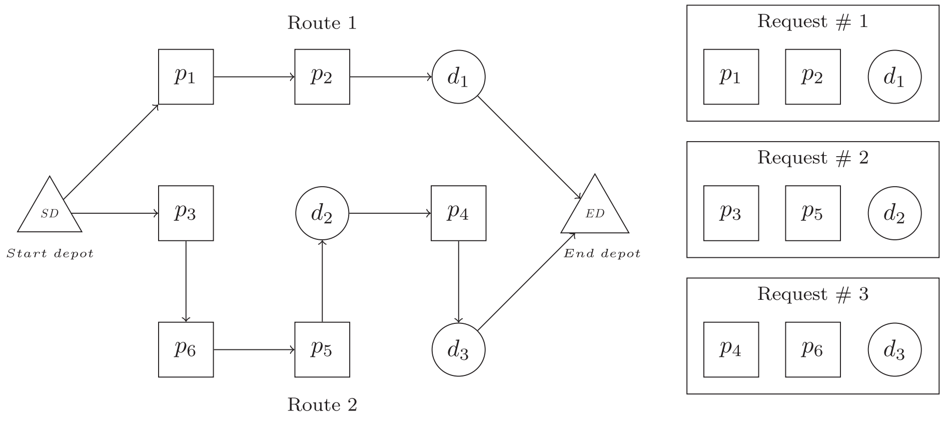


Fig. 1. An example of MPDPTW.

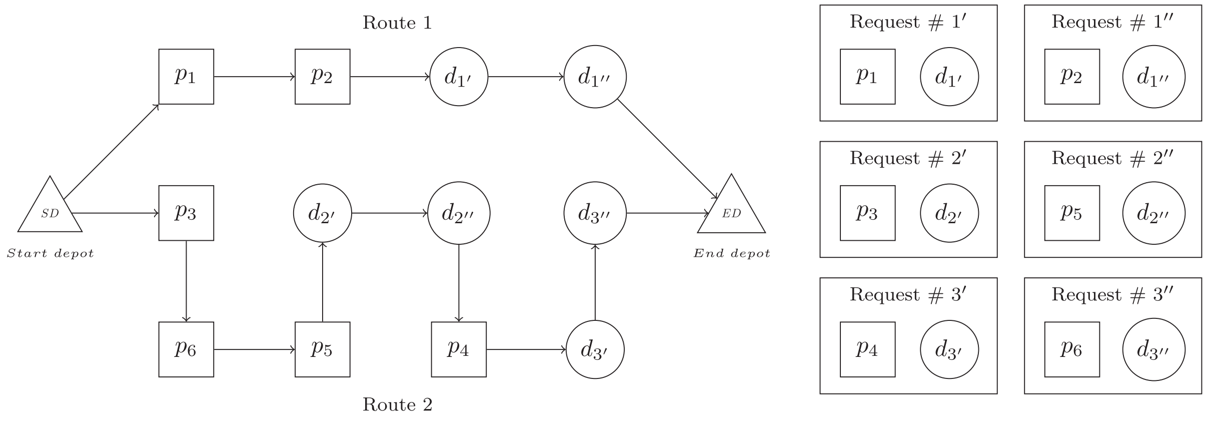


Fig. 2. An example of the transformation to a PDPTW.

pickup nodes is assigned to one delivery, resulting in six PDPTW requests, where each request has exactly two nodes (one pickup and one delivery). For instance, for Request 1 in Fig. 1, we create a copy of the delivery node d_1 resulting in two delivery nodes $d_{1'}$ and $d_{1''}$, then we assign p_1 to $d_{1'}$ and p_2 to $d_{1''}$ resulting in two PDPTW requests, Request 1′ and Request 1″. Note that the cost of the arc $(d_{1'}, d_{1''})$ equals to zero. The solution of the new problem in Fig. 2 is similar to the solution of the problem in Fig. 1 and they have the same cost.

3. Mathematical models

In this section, we propose three mathematical formulations to the problem. The first one is a three-index formulation proposed by Naccache et al. (2018) in which an index in the variables identifies the vehicles, presented in Section 3.1. The second formulation is a classical two-index formulation adapted to the special structure of the problem, shown in Section 3.2. The third model is based on the Asymmetric Representatives Formulation and is introduced in Section 3.3.

3.1. Three-index formulation

The problem can be mathematically formulated with the following decision variables:

- x_{ij}^k equal to 1 if arc (i, j) is traversed by vehicle k, 0 otherwise;
- y_{rk} equal to 1 if request r is satisfied by vehicle k, 0 otherwise;
- S_i indicating the starting time of service at node $i \in V$.

The MPDPTW can then be formulated as follows:

$$\min \sum_{k\in K} \sum_{(i,j)\in A} c_{ij} x_{ij}^k \tag{1}$$

$$\text{s.t.} \sum_{j\in A^+(i)} x_{ij}^k = y_{r(i)k} \qquad k \in K, i \in N \tag{2}$$

$$\sum_{j\in A^-(i)} x_{ji}^k = y_{r(i)k} \qquad k \in K, i \in N \tag{3}$$

$$\sum_{j\in A^+(0)} x_{0j}^k \le 1 \qquad k \in K \tag{4}$$

$$\sum_{k\in K} y_{rk} = 1 \qquad r \in R \tag{5}$$

$$S_j \ge S_i + \left(s_i + t_{ij} + M\right) \sum_{k\in K} x_{ij}^k - M \qquad (i, j) \in A \tag{6}$$

$$a_i \le S_i \le b_i \qquad i \in V \tag{7}$$

$$S_{d_r} \ge S_i + s_i + t_{id_r} \qquad i \in P_r, r \in R \tag{8}$$

$$x_{ij}^k, y_{rk} \in \{0, 1\} \qquad (i, j) \in A, r \in R, k \in K \tag{9}$$

$$S_i \ge 0 \qquad i \in V. \tag{10}$$

The objective function (1) minimizes the overall transportation cost. Constraints (2) and (3) are degree constraints associated with nodes visited by vehicle k. They also ensure that all the nodes of a request are visited by the same vehicle. Constraints (4) ensure that at most $|K|$ vehicles are used in the solution. Constraints (5) force a request to be served by exactly one vehicle. Constraints (6) and (7) guarantee schedule feasibility with respect to time windows. Note that constraints (6) also eliminate subtours. The precedence order is preserved via constraints (8). Constraints (9) and (10) impose the nature and the domain of the variables. M is a big enough number and can be set to the maximum return time to the depot b_{p+n+1}.

3.2. Two-index formulation

A known problem with the three-index formulation is that the number of variables is large because for each arc there are $|K|$ variables. In this section, we formulate the problem using the well-known two-index formulation. Pairing and precedence inequalities need to be added to ensure that all nodes of a request are served by the same vehicle. In other words, the same unit of flow must pass through the nodes of a request to ensure feasibility. To impose these inequalities, it is convenient to define set $\mathcal{S}$ of all node subsets $S \subseteq V$ such that $0 \notin S$, $P_r \subset S$, $d_r \notin S$, and $p+n+1 \in S$ for at least one request $r(i)$. The problem can be mathematically formulated with the following decision variables:

- x_{ij} equal to 1 if arc (i, j) is traversed by a vehicle, 0 otherwise;
- S_i indicating the starting time of service at node $i \in V$.

The MPDPTW can then be formulated as follows:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{11}$$

$$\text{s.t.} \sum_{i\in V} x_{ij} = 1 \qquad j \in N \tag{12}$$

$$\sum_{j\in V} x_{ij} = 1 \qquad i \in N \tag{13}$$

$$\sum_{j\in N} x_{0j} \le |K| \tag{14}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \le |S| - 1 \qquad S \subseteq N \tag{15}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \le |S| - 2 \qquad S \in \mathcal{S} \tag{16}$$

$$S_j \ge S_i + s_i + t_{ij} - M\big(1 - x_{ij}\big) \qquad (i, j) \in A \tag{17}$$

$$a_i \le S_i \le b_i \qquad i \in V \tag{18}$$

$$S_{d_r} \ge S_i + s_i + t_{id_r} \qquad i \in P_r, r \in R \tag{19}$$

$$x_{ij} \in \{0, 1\} \qquad (i, j) \in A \tag{20}$$

$$S_i \ge 0 \qquad i \in V. \tag{21}$$

The objective function (11) minimizes the overall transportation cost. Constraints (12) and (13) are degree constraints requiring each node to be visited exactly once. Constraint (14) ensures that the maximum number of vehicles used in the solution is respected. Constraints (15) are subtour elimination constraints. Constraints (16) are pairing and precedence constraints, which state for each request $r \in R$ represented by a set of pickup nodes $P_r \subseteq P$ and a delivery node $d_r \in D$, for each node $p \in P_r$, node d_r is visited after node p, and both are visited by the same vehicle. Constraints (17) and (18) guarantee schedule feasibility with respect to time windows. The precedence order is preserved via constraints (19). Constraints (20) and (21) impose the nature and the domain of the variables. Again, M is a big number and can be set to the maximum return time to the depot b_{p+n+1}.

Furtado et al. (2017) have proposed a new compact formulation for the PDPTW, introducing a new way to represent pairing relations. They defined new additional variables to identify the routes by storing the index of the first node visited by each route. We define the continuous decision variable v_i that is equal to the index of the first node in the route that visits node $i \in N$. For example, consider that a node $j \in N$ is the first node visited in the route that visits i; as a result we have $v_i = j$. Therefore, the nodes of a given request r with i, $k \in P_r$ and $d_r \in D$ are visited in the same route if and only if $v_i = v_k = v_{d_r}$. The following constraints can be used to reinforce the pairing constraints:

$$v_{d(r)} = v_i \qquad i \in P_r, r \in R \tag{22}$$

$$v_j \ge jx_{0j} \qquad j \in N \tag{23}$$

$$v_j \le jx_{0j} - |N|(x_{0j} - 1) \qquad j \in N \tag{24}$$

$$v_j \ge v_i + |N|(x_{ij} - 1) \qquad i, j \in N \tag{25}$$

$$v_j \le v_i + |N|(1 - x_{ij}) \qquad i, j \in N. \tag{26}$$

Constraints (22) guarantee that the pickup nodes and the delivery node of a given request belong to the same route. Constraints (23) and (24) ensure that the route identifier is taken as the index of the first visited node. Constraints (25) and (26) guarantee that the index of the first node is forwarded to the next nodes in the route. The value $|N|$ represents the number of customer nodes and it is given by $|N| = (n + p)$.

3.3. Asymmetric representatives formulation

The ARF for the MPDPTW is inspired by the work of Jans and Desrosiers (2013) for the job grouping problem. It is based on the idea of identifying a cluster of requests by its lowest indexed request. This formulation is mainly used to eliminate the symmetry between identical vehicles in the case of a homogeneous transportation fleet. It also yields a smaller problem due to the reduction of the number of arcs in the graph G. Symmetry exists when for each feasible solution, other alternative feasible solutions are obtained with the same value of the objective function by just reassigning the same clusters of requests to different vehicles. Symmetry breaking is possible by imposing a request assignment rule in the form of symmetry breaking constraint. This constraint ensures that the assignment of a request i to a higher indexed cluster $j > i$ is not allowed. The number of clusters in this formulation equals the number of requests to be served.

The illustration in Fig. 3 shows a comparison between a symmetric formulation (SF) and ARF. In this example, we have three requests that must serviced. In SF, there is no restrictions on the assignment of requests to the clusters, as each one of the requests can be assigned to any of the three clusters. In ARF, each cluster

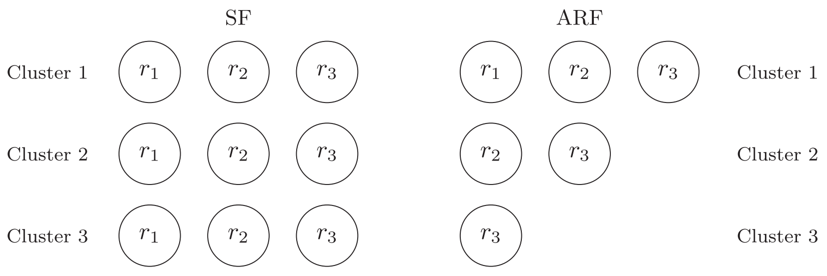


Fig. 3. A comparaison between SF and ARF.

is identified by its lowest indexed request, for instance, *Cluster* 2 is identified by r_2, and in a given solution of this problem with three requests, if r_2 is not assigned to *Cluster* 2, this cluster will be closed and it will not be assigned to a vehicle. Furthermore, the ARF breaks the symmetry by imposing the request assignment rule. For instance, the first request r_1 must be assigned to the first cluster. For the second request r_2, we impose that it must belong to either the first or the second clusters. Finally, the third request r_3 must be assigned to one of the first three clusters. Consider a given feasible solution, in which requests r_1 and r_2 are assigned to *Cluster* 1 and r_3 is assigned to *Cluster* 3; Then, in SF, it is possible to find another alternative feasible solution with the same value of the objective function by just swapping the requests in clusters 1 and 3, meaning that r_1 and r_2 will then be assigned to *Cluster* 3 and r_3 will be assigned to *Cluster* 1. This kind of swapping is not possible in ARF due to the restrictions on request assignment, meaning that r_1 and r_2 are not allowed to be assigned to *Cluster* 3.

In this formulation, the binary variable y_{rk} is defined to indicate whether or not request r is assigned to cluster k. It is also used to indicate whether or not a cluster of requests is assigned to one of the vehicles; for instance, if $y_{kk}=1$, it means that the cluster of requests with the identifier k is assigned to one of the identical vehicles. Otherwise, $y_{kk}=0$, the cluster is closed and no request can be assigned. In this formulation, we also add a new constraint to prevent pairs of infeasible requests from being assigned to the same cluster. To this end, we define $W=\{(i,j)\in R\times R|\ i$ and j cannot be feasibly served together by one vehicle$\}$ as the set of infeasible pairs of requests. It represents pairs of requests that cannot be in the same route because they lead to an infeasible path.

The problem can be mathematically formulated with the following decision variables:

- x_{ij}^k equal to 1 if arc (i,j) is present in cluster k, 0 otherwise;
- y_{rk} equal to 1 if request r is present in cluster k, 0 otherwise;
- S_i indicating the starting time of service at node $i\in V$.

To ease the readability, variable $y_{rk}=0$ if $r<k$, and $x_{ij}^k=0$ if $r(i)<k$ or $r(j)<k$. The ARF for the MPDPTW is then as follows:

$$\min \sum_{k\in R}\sum_{(i,j)\in A} c_{ij}x_{ij}^k \qquad (27)$$

$$\text{s.t.} \sum_{j\in A^+(i)} x_{ij}^k = y_{r(i)k} \qquad k\in R, i\in N \qquad (28)$$

$$\sum_{j\in A^-(i)} x_{ji}^k = y_{r(i)k} \qquad k\in R, i\in N \qquad (29)$$

$$\sum_{j\in A^+(0)} x_{0j}^k \le 1 \qquad k\in R \qquad (30)$$

$$\sum_{k\in R} y_{kk} \le |K| \qquad (31)$$

$$\sum_{k\in R} y_{rk} = 1 \qquad r\in R \qquad (32)$$

$$\sum_{r\in R: r>k} y_{rk} \le y_{kk}(|R|-k) \qquad k\in R \qquad (33)$$

$$y_{rk}+y_{r'k}\le 1 \qquad k,r',r\in R, (r,r')\in W \qquad (34)$$

$$S_j \ge S_i + \left(s_i+t_{ij}+M\right)\sum_{k\in K} x_{ij}^k - M \qquad (i,j)\in A \qquad (35)$$

$$a_i \le S_i \le b_i \qquad i\in V \qquad (36)$$

$$S_{d_r} \ge S_i + s_i + t_{id_r} \qquad i\in P_r, r\in R \qquad (37)$$

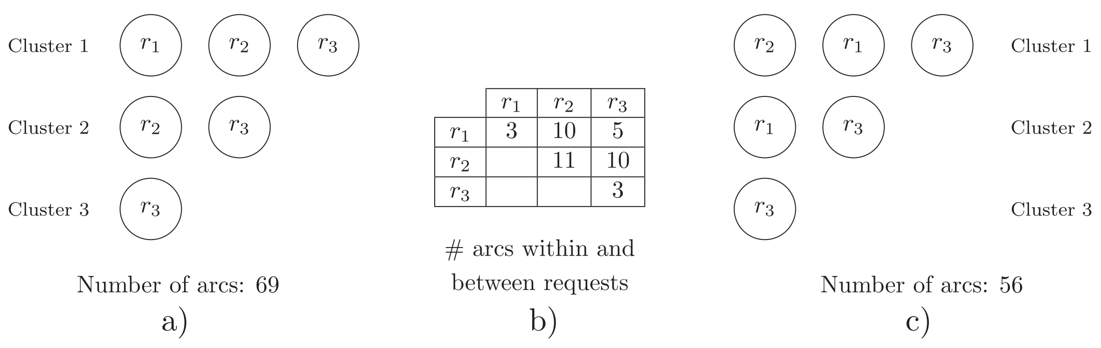
$$x_{ij}^k, y_{rk} \in \{0,1\} \qquad (i,j)\in A, r,k\in R \qquad (38)$$

$$S_i \ge 0 \qquad i\in V. \qquad (39)$$

The objective function (27) minimizes the overall transportation cost. Constraints (28) and (29) are degree constraints associated with nodes in cluster k. They also ensure that all the nodes of a request are in the same cluster. Constraints (30) ensure that at most one vehicle departs from the depot for each cluster. Constraint (31) ensures that at most $|K|$ vehicles are used in the solution. Constraints (32) force a request to be present in exactly one cluster. Constraints (33) ensure that the cluster k is open only if the request k is assigned to this cluster. Constraints (34) ensure that pairs of infeasible requests cannot be in the same cluster. Constraints (35) and (36) guarantee schedule feasibility with respect to time windows. The precedence order is preserved via constraints (37). Constraints (38) and (39) impose the nature and the domain of the variables. M is a big enough number and can be set to the maximum return time to the depot b_{p+n+1}.

To further improve the ARF, it is possible to rearrange the indices of the requests such that fewer arcs are considered in the formulation. This is demonstrated in Fig. 4.b which shows how many feasible arcs exist between the three requests of the example. For the clusters represented in Fig. 4.a, we should consider all arcs between requests r_1, r_2 and r_3 for *Cluster* 1 (totaling 3+10+11+10+3+5=42), plus all arcs between requests r_2 and r_3 for *Cluster* 2 (totaling 11+10+3=24), plus all arcs within request r_3 for *Cluster* 3 (totaling 3 arcs), for a total number of arcs in the formulation equal to 42+24+3=69.

By rearranging the requests within the clusters, as shown in Fig. 4.c, *Cluster* 1 continues with 42 arcs, *Cluster* 2 now only contains the arcs between requests r_1 and r_3 (totaling 3+5+3=11), and *Cluster* 3 only the arcs within request r_3, i.e., 3 arcs. The



	r_1	r_2	r_3
r_1	3	10	5
r_2		11	10
r_3			3

Fig. 4. An example of the reduction of the number of arcs in ARF.

formulation for the clusters represented by Fig. 4.c contains only 42+11+3=56 arcs.

A MIP was formulated to determine the best order of input requests that yields the lowest number of arcs in the input graph G. The idea of this MIP is based on the classical assignment problem: on one side we have the list of requests and on the other side we have the possible assignment positions. The goal is to minimize the overall number of arcs in the input graph G by assigning the requests with high overall number of arcs to the first positions. Therefore, these requests will be prevented from being assigned to a higher indexed cluster which results in reducing the number of arcs in G. An example of this case is the request r_2 in Fig. 4 which was assigned to *Cluster* 1 in order to reduce the overall number of arcs in the graph as it is shown in Fig. 4.c. In this MIP, the binary variable u_i^k is used to assign a request to a cluster, the binary variable v_i^k is used to identify whether the arcs of request i are in cluster k or not, finally, the binary variable z_{ij}^k is used to identify whether the arcs between the nodes of requests i and j are in cluster k or not. The MIP is formulated with the following parameters and decision variables:

Parameters:

- e_{ij} equal to the number of feasible arcs between requests i and j;
- e_i equal to the number of feasible arcs from the depot to all pickups of request i, plus the arc from the drop to the depot, plus feasible arcs between each pair of nodes of request i.

Variables:

- u_i^k equal to 1 if request i is the first in cluster k;
- v_i^k equal to 1 if request i is in cluster k;
- z_{ij}^k equal to 1 if requests i and j are both in cluster k.

The MIP can then be formulated as follows:

$$\min \sum_{k\in R}\sum_{j\in R}\sum_{i\in R, i<j} e_{ij}z_{ij}^k + \sum_{k\in R}\sum_{j\in R} e_j v_j^k \quad (40)$$

$$\text{s.t.} \sum_{k\in R} u_i^k = 1 \qquad i \in R \quad (41)$$

$$\sum_{i\in R} u_i^k = 1 \qquad k \in R \quad (42)$$

$$v_i^k \geq u_i^k \qquad i, k \in R \quad (43)$$

$$z_{ij}^k \geq v_i^k + v_j^k - 1 \qquad i, j, k \in R, i < j \quad (44)$$

$$v_i^l \geq u_i^k - \sum_{j\in W_i} u_j^l \qquad i, l, k \in R, l < k \quad (45)$$

$$u_i^k \in \{0, 1\} \qquad i, k \in R \quad (46)$$

$$v_i^k \geq 0 \qquad i, k \in R \quad (47)$$

$$z_{ij}^k \geq 0 \qquad i, j, k \in R. \quad (48)$$

The objective function (40) minimizes the overall number of arcs. Constraints (41) and (42) ensure that a request can be assigned to the first position in exactly one cluster. Constraints (43) force variables v_i^k to be dependent on variables u_i^k to ensure the feasibility of the model, guaranteeing the existence of the request in the same cluster where it appears in the first position. Constraints (44) link variables z_{ij}^k to v_i^k. Constraints (45) ensure that pairs of infeasible requests cannot be in the same cluster where $W_i = \{j \in R | (i, j) \in W\}$ is the set of requests which are in conflict with request i. Constraints (46)–(48) impose the nature and the domain of the variables.

4. Valid inequalities

This section describes several families of valid inequalities for the MPDPTW. The usefulness of these inequalities is demonstrated through computational experiments in Section 6.2. To describe them, an additional notation is needed. For any node subset $S \subseteq V$, we define $x(S) = \sum_{i,j\in S} x_{ij}$, $\bar{S} = \{i \in V | i \notin S\}$. And for any node subsets S and T, we define $x(S, T) = \sum_{i\in S}\sum_{j\in T} x_{ij}$. Note that the inequalities in Sections 4.1 and 4.2 are only valid for the 2-index formulation.

4.1. Lifted subtour elimination constraints

Lifted subtour elimination constraints (LSEC) were used for solving the dial-a-ride problem (Cordeau, 2006), and PDPTW (Ropke et al., 2007). Consider the classical subtour elimination constraint (15). It can also be lifted in the case of MPDPTW by taking into account the fact that for each request $r(i)$, the set of pickup nodes P_r must be visited before the delivery node d_r. For any set $S \subseteq N$, let $\pi(S) = \{i \in P_r | d_r \in S\}$ and $\sigma(S) = \{d_r \in D | i \in P_r \cap S\}$ denote the sets of predecessors and successors of S. Two families of inequalities were proposed by Balas, Fischetti, and Pulleyblank (1995) for the precedence constrained asymmetric TSP which also apply to the MPDPTW by considering the fact that each node $i \in N$ is either the predecessor or the successor of exactly one node (Ropke et al., 2007). For $S \subseteq N$, the following inequality, called a successor inequality (or σ-inequality) is valid for the MPDPTW:

$$x(S) + \sum_{i\in\bar{S}\cap\sigma(S)}\sum_{j\in S} x_{ij} + \sum_{i\in\bar{S}\setminus\sigma(S)}\sum_{j\in S\cap\sigma(S)} x_{ij} \leq |S| - 1. \quad (49)$$

Similarly, the following predecessor inequality (or π-inequality) is valid for the MPDPTW:

$$x(S) + \sum_{i\in S} \sum_{j\in \bar{S}\cap\pi(S)} x_{ij} + \sum_{i\in S\cap\pi(S)} \sum_{j\in \bar{S}\setminus\pi(S)} x_{ij} \leq |S| - 1. \tag{50}$$

4.2. Infeasible path constraints

An infeasible path can occur in different ways. For instance, the violations of time windows may give rise to a path that is infeasible. In some other cases, a given path could satisfy time windows, however, precedence relationships are not respected, i.e., a pickup node of a given request visited after the delivery node of the same request. Forbidding such paths can be accomplished through different valid inequalities. In general, let $F = (k_1, \dots, k_l)$ be an infeasible path and let $A(F)$ be the arc set of F. The following inequality is valid:

$$\sum_{i=1}^{l-1} x_{k_i,k_{i+1}} \leq |A(F)| - 1. \tag{51}$$

Infeasible path constraints can be strengthened in different ways. In this paper, we consider three different ways to strengthen inequality (51). The first one is inspired by the existence of pairing and precedence relations between nodes of the same request, the second one uses the concept of infeasible pairs of requests defined in Section 3.3, and the third one is based on the idea of eliminating infeasible paths by considering groups of infeasible paths sharing some common arcs which was introduced in Ropke et al. (2007) and known as fork constraints. In what follows, we discuss in detail the strengthening of the infeasible path inequality.

4.2.1. Infeasible drop/pickup paths

This section introduces valid inequalities for the MPDPTW that aim to forbid infeasible paths violating a special case of the pairing and precedence constraints (PC). For a given request $r\in R$, let $F = (0, k_1, k_2, \dots, k_l, d_r)$ be an infeasible path in G such that not all pickups of r are on the path from 0 to d_r. A way to strengthen inequality (51) for this case is to consider that any ordering of the nodes k_1 to k_l and d_r will lead to an infeasible solution. Let $S = \{k_1, k_2, \dots, k_l\}$ be a node subset. Inequality (51) can then be replaced by the following stronger valid inequality:

$$x(0, S) + x(S \cup \{d_r\}) \leq |S|. \tag{52}$$

The same idea can be used in the case of paths from a pickup to the depot. For a given request $r\in R$, let $F = (p, k_1, k_2, \dots, k_l, p + n + 1)$ be an infeasible path in G such that $p\in P_r$ and $k_i \neq d_r$ for $i = 1, \dots, l$. This infeasible path can be forbidden using inequality (51), however, a similar way as in the case of depot to drop paths is used to strengthen it in the case of pickup to depot paths. We consider that any ordering of the nodes k_1 to k_l and p will lead to an infeasible solution. Let $S = \{k_1, k_2, \dots, k_l\}$ be a node subset. Inequality (51) can then be replaced by the following stronger valid inequality:

$$x(S \cup \{p\}) + x(S, p + n + 1) \leq |S|. \tag{53}$$

The same idea can also be used in the case of paths from the drop to a pickup node of the same request. For a given request $r\in R$, let $F = (d_r, k_1, k_2, \dots, k_l, p)$ be an infeasible path in G such that $p\in P_r$ and d_r is the drop node of r. Let $S = \{k_1, k_2, \dots, k_l\}$ be a node subset. The following valid inequality can replace inequality (51) to forbid an infeasible drop to pickup path:

$$x(d_r, S) + x(S) + x(S, p) \leq |S|. \tag{54}$$

4.2.2. Infeasible requests paths

Paths can be infeasible if they contain nodes belonging to pairs of infeasible requests. Although inequality (51) can be used to forbid them, a stronger inequality is used in this case named infeasible request constraint (IRC). For a given pair of nodes $i, j\in N$ such that $(r(i), r(j))\in W$, the path $F = (i, k_1, k_2, \dots, k_l, j)$ is infeasible in G. Let $S = \{k_1, k_2, \dots, k_l\}$ be a node subset. The following inequality is valid for the MPDPTW:

$$x(\{i, j\} \cup S) \leq |S|. \tag{55}$$

4.2.3. Fork constraints

As stated before, the main idea of the fork constraints (FC) is to eliminate infeasible paths by considering groups of infeasible paths sharing some common arcs. Inequalities (56) and (57) are called infork and outfork inequalities, respectively.

Let $F = (k_1, \dots, k_l)$ be a feasible path in G, and $S_1, \dots, S_l, T \subset N$, such that $k_j \notin S_{j+1}$ for $j = 2, \dots, l-1$. If for any integer $h \leq l$ and any node pair $i\in S_h$, $j\in T$, the path $(i, k_1, \dots, k_h, j)$ is infeasible, then the following inequality is valid for the MPDPTW:

$$\sum_{h=1}^{l} \sum_{i\in R_h} x_{i,k_h} + \sum_{h=1}^{l-1} x_{k_h,k_{h+1}} + \sum_{j\in T} x_{k_l,j} \leq l. \tag{56}$$

Let $F = (k_1, \dots, k_l)$ be a feasible path in G, and $S, T_1, \dots, T_l \subset N$, such that $k_j \notin T_{j-1}$ for $j = 2, \dots, l$. If for any integer $h \leq l$ and any node pair $i\in S$, $j\in T_h$, the path $(i, k_1, \dots, k_h, j)$ is infeasible, then the following inequality is valid for the MPDPTW:

$$\sum_{i\in R} x_{i,k_1} + \sum_{h=1}^{l-1} x_{k_h,k_{h+1}} + \sum_{h=1}^{l} \sum_{j\in T_h} x_{k_h,j} \leq l. \tag{57}$$

4.3. New valid inequalities

This section describes a set of new valid inequalities for the MPDPTW. Additional constraints can be added to the ARF by considering the fact that some requests might be alone in their cluster. Let $F_k = (k_1, \dots, k_l)$ be the optimal path to service the nodes of the request $k\in R$ in a single route for $l > 1$, and let $A(F_k)$ be the arc set of F_k. Then the following inequality is valid for the ARF:

$$\sum_{(i,j)\in F_k} x^k_{ij} \geq |A(F_k)|(y_{kk} - \sum_{r\in R, r\neq k} y_{rk}). \tag{58}$$

In case request k is the only one assigned to cluster k, then the right hand side of inequality (58) will be equal to $|A(F_k)|$ which represents exactly the number of arcs in the path F_k. However, if at least one other request is assigned to cluster k, then the right hand side of inequality (58) will be equal to 0 or a negative value.

The violation of precedence constraints can be prevented by generating different types of valid inequalities considering special cases of the violation of these constraints such as inequalities (52)–(54). In what follows, other special cases of the violation of precedence constraints are prevented through inequalities (59)–(62). For each pair of requests r_1, $r_2\in R$ such that d_{r_1} and d_{r_2} are the drop nodes of r_1 and r_2, respectively, the following inequalities are valid for MPDPTW:

$$\sum_{i\in P_{r_1}} x_{id_{r_2}} + \sum_{i\in P_{r_2}} x_{id_{r_1}} \leq 1 \tag{59}$$

$$\sum_{i\in P_{r_1}} x_{id_{r_2}} + \sum_{j\in P_{r_2}} x_{d_{r_1}j} \leq 1 \tag{60}$$

$$\sum_{j \in P_{r_1}} x_{d_{r_2} j} + \sum_{i \in P_{r_2}} x_{i d_{r_1}} \leq 1 \qquad (61)$$

$$\sum_{j \in P_{r_1}} x_{d_{r_2} j} + \sum_{j \in P_{r_2}} x_{d_{r_1} j} \leq 1. \qquad (62)$$

In any solution of the MPDPTW, the vehicles must depart from the start depot. Therefore, for each node $i \in N$, the following inequality is valid for MPDPTW:

$$S_i \geq \sum_{j \in N} x_{ji}(t_{0j} + s_j + t_{ji}). \qquad (63)$$

Inequality (63) is a special case of the general schedule feasibility inqualities ((6), (17) and (35)). The starting time of service at node i must be at least greater than or equal to the arrival time at node i from its predecessor node j, assuming that node j is the first node visited after departing from the depot.

Inequalities (59)–(63) are valid for the 2-index formulation, and they can easily be adapted to the ARF and the 3-index formulation by only taking into consideration the index k in the decision variable.

5. Branch-and-cut algorithm

In this section, we describe the branch-and-cut algorithm developed to solve the MPDPTW. After applying the preprocessing techniques presented in Section 5.1, the algorithm starts by solving the LP-relaxation of the problem. The models are solved using a commercial MIP solver that is in charge of solving the LP relaxations and branching. The description of the separation procedures used to generate the valid inequalities from Section 4 is presented in Section 5.2.

5.1. Preprocessing

This section describes the preprocessing procedures that are performed prior to applying the branch-and-cut algorithm. The goal is to modify the input graph G in order to reduce it, and to tighten time windows associated with each node $i \in N$.

5.1.1. Infeasible request pairs

Section 3.3 defined the set W of pairs of requests that cannot be in the same route because they lead to an infeasible path. In order to build such a set, we perform three main steps for each pair of requests $r_1, r_2 \in R$:

Step 1: we try to find an initial feasible solution through the insertion of the requests in a route. If the solution is feasible, we move to the next pair of requests. Otherwise, we continue to step 2.

Step 2: few iterations of the ALNS of Naccache et al. (2018) are performed to find a feasible solution. Again, if the solution is feasible, we move to the next pair of requests. If the solution of the ALNS is infeasible, we continue to step 3.

Step 3: we enumerate all possible feasible paths built using nodes belonging to both requests. If no feasible path is found, then the pair (r_1, r_2) is added to the set W.

5.1.2. Time-windows tightening

The time-windows are tightened following the procedure described in Ascheuer, Fischetti, and Grötschel (2001). This procedure is implemented through four steps for each node $k \in N$. We cycle through these steps until no more changes can be made to the time windows:

Step 1: $a_k \leftarrow \max\{a_k, min_{i \in A^-(k)}\{a_i + s_i + t_{ik}\}\}\ \forall k \in N \mid A^-(k) \neq \emptyset$.

Step 2: $a_k \leftarrow \max\{a_k, min\{b_k, min_{j \in A^+(k)}\{a_j - s_k - t_{kj}\}\}\ \forall k \in N \mid A^+(k) \neq \emptyset$.

Step 3: $b_k \leftarrow \min\{b_k, max\{a_k, max_{i \in A^-(k)}\{b_i + s_i + t_{ik}\}\}\ \forall k \in N \mid A^-(k) \neq \emptyset$.

Step 4: $b_k \leftarrow \min\{b_k, max_{j \in A^+(k)}\{b_j - s_k - t_{kj}\}\}\ \forall k \in N \mid A^+(k) \neq \emptyset$.

In this paper, we propose a second time-windows tightening procedure that is based on enumerating all possible paths formed by nodes of each request $r \in R$. Let $F(r)$ be the set of all possible feasible paths created using only the nodes of request r. Let h_k^F be the arrival time at node k in path F. For the drop node d_r we have:

$$a_{d_r} \leftarrow \max\{a_{d_r}, \min_{k \in P_r}\{\min_{F \in F(r)}\{h_k^F + s_k + t_{kd_r}\}\}\}$$

and for each pickup node $k \in P_r$ such that k is never the first node in any path $F \in F(r)$, we have:

$$a_k \leftarrow \max\{a_k, \min_{i \in P_r \setminus \{k\}}\{\min_{F \in F(r)}\{h_i^F + s_i + t_{ik}\}\}\}$$

5.1.3. Arc elimination

Each one of the three formulations presented in this paper is defined on a complete graph G. However, due to time windows, infeasible pairs of requests, and pairing and precedence constraints, many arcs cannot belong to a feasible solution. Therefore they can be removed from the graph. The following are the arcs eliminated from the graph:

1. Arcs $(0, d_r)$, $(p, 0)$, and (d_r, p) are infeasible for each request $r \in R$ such that $p \in P_r$ and d_r is the drop node of the request r.
2. Arc $(i, j) \in A$ is infeasible if $a_i + s_i + t_{ij} > b_j$.
3. Arc $(i, j) \in A$ is infeasible if $(r(i), r(j)) \in W$.
4. For each request $r \in R$, we enumerate all possible feasible paths built using nodes of r. Arc $(i, j) \in A$ such that $r(i) = r(j) = r$ is infeasible if it does not belong to any of the feasible paths.
5. For each pair of requests $r_1, r_2 \in R$ such that $(r_1, r_2) \notin W$, we enumerate all possible feasible paths built using nodes belonging to both requests. Arc $(i, j) \in A$ such that $r(i) = r_1$ and $r(j) = r_2$ is infeasible if it does not belong to any of the feasible paths.

5.2. Separation procedures

We now describe the separation procedures used to identify violated inequalities. In addition to the valid inequalities in Section 4, we have also separated the pairing and precedence constraints (constraints (16) in the 2-index formulation), the infeasible drop to pickup constraints (54) and the generalized order constraints (GOC) adapted to the MPDPTW. However, these inequalities are not used in this paper because of their negative impact on the performance of the 2-index formulation in terms of CPU time. The valid inequalities introduced in this paper are only generated for the 2-index formulation. This is due to their negative impact on the CPU time when they are adapted and generated for the ARF and the 3-index formulation.

5.2.1. Lifted subtour elimination constraints

The separation procedure of the lifted subtour elimination constraints starts by separating the subtour elimination inequalities represented by the inequality (15) in the 2-index formulation. The CVRPSEP package is used in this case. It is a package of separation routines for the capacitated vehicle routing problem developed by Lysgaard, Letchford, and Eglese (2004). We have adapted to the case of the MPDPTW. When a subtour S is identified, inequalities (49) and (50) are generated by adding all other arcs in the left-hand side of the cut.

5.2.2. *Infeasible drop/pickup paths*

This section describes the separation procedure used to generate inequalities (52) and (53) for a fractional solution at a given node of the tree. In the case of the infeasible drop paths, in the first step of the heuristic, for each request r, we try to find a path from the depot 0 to the drop d_r by depth search on graph G. If a path exists, we check the feasibility of the path: all the pickups of request r must be in the path; otherwise, a pairing and precedence constraint is violated. The same logic is used in the case of the infeasible pickup paths: for each request r, we try to find a path from each pickup node p such that $p \in P_r$ to the depot 0 using a depth search in G. If a path exists, we check the feasibility of the path: the drop d_r of request r must be in the path; otherwise, a pairing and precedence constraint is violated.

5.2.3. *Infeasible request paths*

Inequalities (55) are separated using a heuristic similar to the one described in Section 5.2.2. For each pair of nodes $i, j \in N$ such that $(r(i), r(j)) \in W$, we try to find a path from node i to node j by depth search on graph G. If a path exists, inequality (55) is generated to forbid this path.

5.2.4. *Fork constraints*

We separate infork constraints (56) in three different ways. The first one searches for a path from the depot 0 to the drop d_r for each request $r \in R$ using a depth-first search on graph G. If a path exists, we check the feasibility of the path: all pickups of request r must be in the path, otherwise the path is infeasible. If the path is infeasible, we then add the depot node 0 to set S_1 and the drop node d_r to the set T. The second way to separate constraints (56) is similar to the previous one, however in this case for each request $r \in R$ and for each pickup node $p \in P_r$ we check for an infeasible path from p to the depot 0. We then add the pickup node p to set S_1 and the depot node 0 to set T. Lastly, we also separate infork constraints (56) by checking for a path between each pair of nodes $i, j \in N$ such that $(r(i), r(j)) \in W$; if such a path exists, we add node i to set S_1 and node j to set T. In the next step we add as many nodes as possible to the sets $S_1, \ldots, S_l, T$.

Knowing that the outfork constraints (57) are similar to the infork constraints as they are only obtained by reversing the orientation of the arcs reaching path the F (see Section 4.2.3), their separation is similar to the separation of the infork constraints.

6. Computational experiments

This section describes the computational experiments. Section 6.1 introduces the characteristics of the MPDPTW instances. The results of detailed and extensive computational experiments are then presented in Section 6.2.

6.1. *Test instances*

The test instances of the MPDPTW used in this paper were proposed by Naccache et al. (2018), and they originate from the Li and Lim (2001) instances for the PDPTW. As explained by Naccache et al. (2018), the generation of a feasible instance starts by making two distinct lists of pickup and delivery nodes for each PDPTW instance. In the next step, the requests are created, firstly by randomly generating k pickup nodes for the request then creating an empty request with a dummy drop node at the depot. The pickup nodes are then randomly added to the request for $k-1$ iterations. Then, the request is inserted into an empty solution using an insertion operator. A random insertion order is used 20 times to insert the request. If the insertion is successful, the pickup node is removed from the list and the next iteration follows; otherwise, the process is restarted by randomly selecting another pickup node. The same process is applied to the drop node of a request once all $k-1$ pickup nodes are selected. In the case where a feasible solution cannot be found, the whole process is restarted. For an instance with n nodes, the request generation process stops when all the requests contain at least n nodes.

The characteristics of the instances are presented in Table 1. Each instance type is defined by three main elements: the TW type, the maximum length of the requests, and the number of nodes (instance size). An instance is defined as *Without* when the TW of the original node is deleted, as *Normal* when the TW for each node is slightly enlarged by opening it 150 units earlier and closing it 150 units later, or as *Large* when the TW for each node is enlarged by opening it 300 units earlier and closing it 300 units later. For each instance, the size of a request must be more than or equal to two, meaning that a request must at least contain one pickup and one delivery node. Requests can contain at most 4 (*Short* requests) or 8 (*Long* requests) pick-up and delivery nodes depending on the type of the instance. Instances contain 25, 35, 50 or 100 nodes, meaning we do not use instances with 400 nodes from Naccache et al. (2018), but we created new instances with 35 nodes to better evaluate the exact algorithm. At the end, we have 24 instance types, and for each type, we generate five instances which make a total of 120 instances. For better understanding, we give an example about how to read the instance's name: the instances of type *L_8_25* represent *Large* TW, *Long* requests, 25 nodes and are denoted *L_8_25_1* to *L_8_25_5*. The instances are publicly available at: https://www.leandro-coelho.com/multi-pickup-and-delivery/.

In this paper, we also consider PDPTW instances. This set of instances correspond to a subset of the instances derived by Li and Lim (2001). These instances involve approximately 50 requests and are divided in six subclasses called LC1, LC2, LR1, LR2, LRC1, LRC2. These instances are publicly available at https://www.sintef.no/.

6.2. *Computational results*

The experiments reported here have been performed on a computer equipped with a 2.1 Gigahertz Intel processor. Each test was allowed to run for a maximum of 1 hour and to use up to 40 Gigabyte of RAM. The branch-and-cut algorithm is coded in C++ and CPLEX 12.8 with default parameters is used to perform the computational experiments. The algorithm is only applied to the 2-index formulation, as adapting and adding the valid inequalities to the 3-index formulation and to the ARF did not improve their effectiveness, and for some instances the runtime increases. We first start by assessing the effectiveness of the ARF with and without imposing the order of the input requests. We then report the results of the branch-and-cut algorithm applied to the 2-index formulation under different scenarios. After that, we report the results of assessing the performance of the three formulations with and without the preprocessing techniques described in Section 5.1, followed by reporting the results of the problem transformation to a PDPTW. Finally, we compare the results of the three formulations proposed in this paper. The optimality gap presented in the next sections is calculated as $100 * \left(\frac{BestUpperBound - LowerBound}{BestUpperBound}\right)$. For MPDPTW instances, the best upper bound is obtained by checking for each instance the minimum value between the solution obtained by ALNS of Naccache et al. (2018) and any of ours. This way, we focus on improving the dual bounds for this difficult problem. For PDPTW instances, the best upper bound for each instance correspond to the best known solution reported at https://www.sintef.no/.

6.2.1. *ARF with and without imposing the order of the input requests*

Section 3.3 presented a MIP to improve the effectiveness of the ARF by imposing the order of the input requests. The model

Table 1
Instance types.

Instance	Without TW		Normal TW		Large TW	
	Short	Long	Short	Long	Short	Long
size	requests	requests	requests	requests	requests	requests
25	W_4_25	W_8_25	N_4_25	N_8_25	L_4_25	L_4_25
35	W_4_35	W_8_35	N_4_35	N_8_35	L_4_35	L_8_35
50	W_4_50	W_8_50	N_4_50	N_8_50	L_4_50	L_8_50
100	W_4_100	W_8_100	N_4_100	N_8_100	L_4_100	L_8_100

Table 2
ARF with different input requests order.

	3-index	Worst case ARF	Random case ARF	Best case ARF
Average # of arcs	15036.9	15668.0	12693.7	9779.5
Average runtime (s)	213.9	33.5	34.0	20.5
Avg gap root (%)	27.7	24.9	24.3	22.6
Avg gap final (%)	39.0	36.3	35.5	34.0
Solved/# inst.	49/120	60/120	60/120	60/120
Solved by all/# inst.			49/120	

was then solved by branch-and-bound using CPLEX 12.8. Since the lower bound improves very slowly and the optimality gap is very large, we developed a quick heuristic algorithm to determine the best possible order of the input requests which yields the lowest number of arcs in the input graph G. The algorithm is composed of three main parts: it starts by sorting the requests according to the number of arcs per request represented by the parameter e_i in the MIP (see Section 3.3) from the request with the largest e_i to the smallest. We then use a simulated annealing (SA) algorithm for a limited number of iterations to improve the solution. Finally, in order to further improve the solution obtained by the SA, we developed a local search algorithm where we swap all possible pairs of requests searching for the best improvement of the solution.

Table 2 reports the results obtained by the ARF under different scenarios and the results obtained by the 3-index formulation. Three different scenarios of the ARF are described in this table: the first scenario represents the worst order of the input requests, the second scenario represents the case where the requests are randomly classified as they appear in the instances, and the third case represents the best case where the algorithm is used. The first column in Table 2 reports the results of the 3-index formulation, then each one of the next three columns represents one possible scenario of the ARF. The rows report the following information respectively: the average number of arcs in each scenario (including the 3-index), the average runtime for instances solved to optimality by all the scenarios simultaneously, the average gap at the root node of the tree for all instances, the average optimality gap for instances unsolved by all the scenarios simultaneously, the number of instances solved to optimality in each scenario out of the total number of instances, and finally the number of instances solved to optimality by all scenarios out of the total number of instances. Note that the 3-index formulation and the ARF (in the three scenarios) use the preprocessing techniques described in Section 5.1, and the runtimes reported in Table 2 include preprocessing times.

Results in Table 2 show that the algorithm has proven to be efficient in improving the performance of the ARF. The average number of arcs and the average runtime for instances solved to optimality under the three scenarios of the ARF have significantly decreased in the best case compared to the worst and random cases. Moreover, the LP relaxation of the ARF has been strengthened as the average gap at the root node of the tree for all instances decreases from 24.9% in the worst case and 24.3% in the random case to 22.6% in the best case. Finally, a considerable improvement of the lower bound is achieved through the application of the algorithm: the average gap for instances unsolved by all the three scenarios of the ARF simultaneously was improved by 2.3% in the best case compared to the worst case, and by 1.5% compared to the random case. Overall, the algorithm was able to improve the effectiveness of the ARF through a considerable reduction of the problem size by reducing the number of arcs in the input graph. The average time needed to find the best order of input requests is 0.01 seconds.

Table 3
Summary of the results for all instances tested under different scenarios.

	LP	LSEC	PC	IRC	FC	Full
Average time (s)	2429.1	2408.6	2397.5	2408.9	2379.5	2310.2
Avg time by all (s)	342.8	295.6	242.2	321.9	206.8	111.9
Average gap (%)	33.8	31.7	33.3	31.8	32.5	29.9
Solved/# inst.	44/120	44/120	43/120	44/120	44/120	47/120
Solved by all/# inst.				43/120		

The comparison between the results obtained by the 3-index formulation and the results obtained by the best case of the ARF shows that the ARF yields a smaller and easier problem than the 3-index formulation and this can be seen through the results that report a significant decrease of the average number of arcs from 15036.9 to 9779.5, the average runtime for instances solved to optimality by the 3-index and the ARF simultaneously from 213.9 seconds to 20.5 seconds, and the average gap for instances unsolved by both formulations from 39.0% to 34.0%. In addition, the best case of the ARF solved to optimality 11 more instances than the 3-index formulation.

6.2.2. The impact of valid inequalities on the 2-index formulation

In this section, we measure the strength of each family of valid inequalities introduced in Section 4 on the 2-index formulation. In Table 3 we report a summary of the results for all instances tested under different scenarios. Column LP reports the results obtained by solving the LP relaxation of the 2-index formulation, where a separation procedure of the classical infeasible path constraints is used in this case to ensure the feasibility of the solutions. The next 4 columns report the results obtained by generating violated inequalities of one of the following families: lifted subtour elimination constraints (LSEC), infeasible drop/pickup path constraints named pairing and precedence constraints (PC), infeasible request path constraints (IRC), and fork constraints (FC). Each one of the scenarios represented by the previous 4 columns uses only one type of valid inequalities. The last column (Full) reports the results obtained when using all valid inequalities described in Section 4 as

Table 4
Results of testing the three formulations with and without preprocessing.

	2-index		3-index		ARF	
	Without	With	Without	With	Without	With
Average time (s)	3071.2	2314.1	2819.3	2217.3	2709.1	1826.6
Avg time by all (s)	446.6	236.9	115.4	93.9	13.9	10.2
Avg gap root (%)	47.1	28.8	48.1	27.7	46.7	22.6
Avg gap final (%)	44.5	36.7	47.6	39.0	46.6	34.0
Solved/# inst.	20/120	47/120	29/120	49/120	33/120	60/120

Table 5
Comparison between the transformed version of the problem to a PDPTW and the original version.

	2-index		3-index		ARF	
	PDPTW	MPDPTW	PDPTW	MPDPTW	PDPTW	MPDPTW
Average time (s)	3145.8	2314.1	2841.2	2217.3	2445.6	1826.6
Avg time by all (s)	745.8	236.9	165.0	93.9	5.6	10.2
Avg gap root (%)	46.5	28.8	40.3	27.7	36.7	22.6
Avg gap final (%)	43.7	36.7	43.6	39.0	39.9	34.0
Solved/# inst.	21/120	47/120	28/120	49/120	41 /120	60/120

well as the preprocessing techniques of Section 5.1. The rows report the following information respectively: the average runtime for all instances, the average runtime for instances solved to optimality by all the scenarios simultaneously, the average optimality gap for instances unsolved by all the scenarios simultaneously, the number of instances solved in each scenario out of the total number of instances, and finally the number of instances solved by all scenarios out of the total number of instances. Note that the different scenarios of the 2-index formulation use the preprocessing techniques described in Section 5.1, and the runtimes reported in Table 3 include preprocessing times.

Results in Table 3 show that using any of the four families of valid inequalities separately yields a limited impact on the results of the 2-index formulation. The most significant improvement in the average runtime for instances solved by all is obtained with the generation of FC, while the PC have limited impact. However, combining all the valid inequalities yields significant improvements as the number of instances increases from 44 in column LP to 47 in column Full. More importantly, the runtime and the optimality gap are also improved by separating all the valid inequalities simultaneously.

6.2.3. *Performance of the formulations with and without preprocessing*

In this section, we present the results of testing the three formulations presented in this paper with and without the application of the preprocessing techniques described in Section 5.1. Table 4 reports the results of testing the three formulations under two scenarios named: *Without* representing the case where the preprocessing techniques are not used and *With* representing the case where the preprocessing techniques are used. For each formulation and for each scenario, Table 4 reports the following results: the average runtime for all instances, the average runtime for instances solved to optimality by all the formulations, the average gap at the root node of the tree for all instances, the average optimality gap for instances unsolved by all the formulations, and the number of instances solved to optimality out of the total number of instances. Note that the ARF is tested with the best order of input requests. The average preprocessing time is 0.6 seconds; however, this can be up to 17.9 seconds for some instances with 100 nodes. Note that the 3-index formulation without preprocessing techniques is equivalent to the model proposed by Naccache et al. (2018).

The results show that the preprocessing techniques have a significant impact on improving the effectiveness of the three formulations. A considerable reduction in problem size and improvement in *Average time*, *Avg time by all*, *Avg gap root* and *Avg gap final* are achieved through the application of these techniques in all the formulations. Moreover, the number of instances solved to optimality per formulation has significantly increased due to the use of the preprocessing techniques. The 2-index formulation solves 27 more instances, the 3-index formulation solves 20 more instances, and the ARF solves 27 more instances.

Note that in the following sections, the preprocessing techniques described in Section 5.1 are used in testing the three formulations, and the reported runtimes include preprocessing times. In addition, the ARF is tested with the best order of input requests.

6.2.4. *Results of the transformation to a PDPTW*

In Section 2.1 we presented how to transform the MPDPTW into a PDPTW, in a series of steps to follow in order to achieve this transformation through the modification of the original graph. In this section, we compare the performance of the three formulations using the transformed and the original versions of the problem. Table 5 reports the results of testing the three formulations under two scenarios named: *PDPTW* representing the case where the problem is transformed to a PDPTW and *MPDPTW* representing the case of the original version of the problem. For each formulation and for each scenario, Table 5 reports the following results: the average runtime for all instances, the average runtime for instances solved to optimality by all the formulations, the average gap at the root node of the tree for all instances, the average optimality gap for instances unsolved by all the formulations, and the number of instances solved to optimality out of the total number of instances.

The results show that the MPDPTW version of the problem outperforms the transformed PDPTW one. The number of instances solved to optimality in the case of PDPTW is 21 for the 2-index formulation, 28 for the 3-index formulation, and 41 for the ARF, while in the case of MPDPTW the number of instances solved to optimality are 47, 49, and 60 for the three formulations, consecutively. The underperformance of three formulations under the PDPTW scenario is mainly due to an increase in the size of the instance by creating the new delivery nodes, and to the difficulty encountered when handling the pairing constraints.

6.2.5. *Comparison between the three formulations*

In this section, we compare the strength of the three proposed formulations and their performance in solving the MPDPTW. We also compare the performance of these formulations in solving a special case of the MPDPTW which is the PDPTW where each request is composed of one pickup and one delivery nodes. Table 6 reports the results for all instances tested on the following formulations: 2-index formulation, 3-index formulation, and ARF. For each formulation, we report the number of instances solved to optimality per instance type, the average gap at the root node of the tree, the average optimality gap after 1 hour of runtime for unsolved instances per instance type, and finally the average runtime for instances solved to optimality per instance type. At the bottom of the table, the last three rows report the following information: *Average* reports the average value for all instances per column type, *Avg by all* reports for columns *Gap root* and *Gap final* the average value per each column for instances unsolved by all formulations, and for column *Time* it reports the average runtime for instances solved to optimality by all formulations. Finally *Sum* reports the overall number of instances solved to optimality by each formulation out of the total number of instances.

The results in Table 6 show that the ARF outperforms the two other formulations as it solves 60 instances to optimality out of

Table 6
Summary of the results for all instances tested on three formulations.

Instance	2-index				3-index				ARF			
	Solved	Gap root (%)	Gap final (%)	Time (s)	Solved	Gap root (%)	Gap final (%)	Time (s)	Solved	Gap root (%)	Gap final (%)	Time (s)
L_4_25	4	27.6	10.2	836.0	5	26.6	–	278.9	5	20.2	–	88.2
L_4_35	0	34.2	23.4	–	2	33.7	18.1	826.4	3	26.5	12.6	370.6
L_4_50	0	44.0	36.8	–	0	47.8	37.7	–	0	41.3	27.9	–
L_4_100	0	54.6	53.0	–	0	58.0	56.2	–	0	52.5	50.1	–
L_8_25	5	9.5	–	3.1	5	7.3	–	1.3	5	3.4	–	0.2
L_8_35	5	15.5	–	32.5	5	10.2	–	6.7	5	3.9	–	1.1
L_8_50	2	22.5	12.3	2126.5	3	22.0	11.9	1469.6	4	11.8	15.1	34.1
L_8_100	0	36.0	32.9	–	0	36.1	32.6	–	0	20.4	17.8	–
N_4_25	5	8.6	–	1.8	5	4.8	–	0.5	5	1.6	–	0.2
N_4_35	5	15.3	–	119.7	5	7.9	–	3.0	5	2.1	–	0.4
N_4_50	2	23.2	16.2	550.6	2	20.4	10.4	538.1	5	9.7	–	56.9
N_4_100	0	38.4	35.7	–	0	38.8	35.5	–	0	22.3	19.0	–
N_8_25	5	1.4	–	0.2	5	1.8	–	0.3	5	0.8	–	0.1
N_8_35	5	2.8	–	0.9	5	1.0	–	0.3	5	0.0	–	0.1
N_8_50	5	4.7	–	21.8	5	4.4	–	19.2	5	1.3	–	0.3
N_8_100	2	10.3	1.9	435.0	0	9.8	7.6	–	5	3.0	–	40.1
W_4_25	0	33.4	20.5	–	0	33.4	25.7	–	0	33.5	25.5	–
W_4_35	0	39.7	29.8	–	0	39.9	37.9	–	0	40.0	37.2	–
W_4_50	0	46.3	39.9	–	0	46.5	45.4	–	0	46.4	45.2	–
W_4_100	0	54.9	53.1	–	0	55.4	54.7	–	0	55.5	54.6	–
W_8_25	2	27.2	15.6	823.9	1	25.6	15.9	805.8	2	23.1	19.4	416.7
W_8_35	0	35.9	24.2	–	1	32.5	27.8	984.5	1	27.2	26.2	175.9
W_8_50	0	47.5	38.2	–	0	47.4	43.4	–	0	44.6	41.6	–
W_8_100	0	56.6	53.8	–	0	54.4	51.6	–	0	51.7	48.9	–
Average	–	28.8	19.5	2314.1	–	27.7	20.2	2217.3	–	22.6	17.0	1826.6
Avg by all	–	43.4	36.7	236.9	–	43.8	39.0	93.9	–	39.0	34.0	10.2
Sum/# inst.	47/120	–	–	–	49/120	–	–	–	60/120	–	–	–

Table 7
Number of instances solved to optimality classified according to the number of nodes per instance.

Size	# inst.	2-index	3-index	ARF
25	30	21	21	22
35	30	15	18	19
50	30	9	10	14
100	30	2	0	5
Sum	120	47	49	60

Table 8
Number of instances solved to optimality classified according to the type of TW.

TW type	# inst.	2-index	3-index	ARF
Large TW	40	16	20	22
Normal TW	40	29	27	35
Without TW	40	2	2	3
Sum	120	47	49	60

120 compared to 49 and 47 instances solved to optimality by the 3-index and the 2-index formulations, respectively. For instances that were solved to optimality by all formulations, ARF required less runtime, the average by all runtime for ARF is 10.2 seconds compared to 93.9 seconds and 236.9 seconds for 3-index and 2-index formulations, and when none of the models could solve to optimality a set of instances per each type, the average by all optimality gap of the ARF was 34.0% which is the best gap among the gaps of all the formulations. The ARF also has the lowest average by all optimality gap at the root node 39.0%, which means that its LP relaxation is the strongest among the proposed formulations. In Table 6, we also notice that the average by all optimality gap at root node for the 2-index and 3-index formulations are somewhat equivalent and slightly different, meaning that the proposed 2-index formulation is competitive. Moreover, it is important to highlight that those results from the ARF are obtained thanks to the developments in preprocessing the input data as shown in Section 6.2.1.

Furthermore, we investigate the impact of instance size, TW type and request length on the results of the three formulations. Table 7 reports the number of instances solved to optimality by each one of the formulations classified according to the number of nodes per instance. The column *# inst.* reports the number of instances per category. Results show that the ARF solves more instances in the four categories presented in the table than the other two formulations, and it solves 5 instances with 100 nodes, confirming that the ARF outperforms the two other formulations. We also notice that although the 2-index formulation solves fewer instances than the ARF and the 3-index formulation, it was able to solve 2 large instances with 100 nodes. This reinforces the statement about the competitiveness of our 2-index formulation and algorithm.

Table 8 reports the number of instances solved to optimality by each formulation classified according to the type of TW (large, normal, without). The column *# inst.* reports the number of instances per category. In general, the results show that the larger the TW the more difficult the instances are to solve by all the formulations. The ARF outperforms the two other formulations in the number of instances solved to optimality per TW type. The comparison between the 2-index and the 3-index formulations shows that the 2-index outperforms the 3-index in the category *Normal TW* as it solves 29 instances to optimality against 27 for the 3-index, while the 3-index solves 4 more instances than the 2-index in the category *Large TW*. Finally, both formulations solve the same number of instances to optimality in the category *Without TW*.

Table 9 reports the number of instances solved to optimality by each one of the formulations classified according to the length of the requests (short requests, long requests). The column *# inst.* reports the number of instances per category. Again, the ARF has shown the best performance among the three formulations in solving instances from both categories reported in Table 9. The comparison between the 2-index and the 3-index formulations shows that the 2-index solves one more instance than the 3-index in the category *Long req*, whereas the 3-index outperforms the 2-index in

Table 9
Number of instances solved to optimality classified according to the maximum length of a request.

Req length	# inst.	2-index	3-index	ARF
Short req	60	16	19	23
Long req	60	31	30	37
Sum	120	47	49	60

Table 10
Summary statistics of the instances solved to optimality.

Instance	Avg # req	Avg # nodes	Avg # vehicles	Avg # req per route	Avg # nodes per route
L_4_25	8.6	26	3.2	2.7	8.3
L_4_35	12.4	36.6	4.0	3.0	9.0
L_4_50	17.2	50.6	–	–	–
L_4_100	34.2	101.2	–	–	–
L_8_25	6	26.8	3.4	1.8	8.2
L_8_35	7.2	37.8	4.2	1.7	9.1
L_8_50	10.8	51	5.3	2.0	9.8
L_8_100	21.6	103.2	–	–	–
N_4_25	8.6	26	4.2	2.1	6.3
N_4_35	11.6	35.8	5.4	2.2	6.7
N_4_50	17.2	50.6	7.2	2.4	7.1
N_4_100	34.2	101.2	–	–	–
N_8_25	6	26.8	3.8	1.6	7.3
N_8_35	7.6	37.2	5.2	1.5	7.4
N_8_50	10.8	51	6.8	1.6	7.5
N_8_100	21.6	103.2	12.8	1.7	8.1
W_4_25	8.6	26	–	–	–
W_4_35	12.6	35.6	–	–	–
W_4_50	17.2	50.6	–	–	–
W_4_100	34.2	101.2	–	–	–
W_8_25	6	26.8	2.0	2.8	13.3
W_8_35	7	37.6	3.0	2.0	12.7
W_8_50	10.8	51	–	–	–
W_8_100	21.6	103.2	–	–	–

the category *Short req*. Results in Table 9 also show that all the formulations solve more *Long req* instances than *Short req* instances; this is mainly due to the increase of the number of requests in *Short req* instances which effects the size and the complexity of the problem.

Table 10 reports some additional information related to each instance type and summary statistics of the instances solved to optimality. For each instance type, the following details are reported: the average number of requests, the average number of nodes, the average number of vehicles used in the solution for instances solved to optimality, the average of the average number of requests per route for instances solved to optimality, and finally the average of the average number of nodes per route for instances solved to optimality.

Table 11 reports the results of testing PDPTW instances on the three formulations presented in this paper. For each formulation, we report the number of instances solved to optimality per instance subclass, the average optimality gap after 1 hour of runtime for unsolved instances by all formulations per instance subclass, and finally the average runtime for instances solved to optimality by all formulations per instance subclass. In column *# inst.*, we report the number of instances used in the tests per subclass. At the bottom of the table, the last three rows report the following information: *Average* reports the average value for all instances per column type, *Avg by all* reports for column *Gap final* the average optimality gap for all instances unsolved by all formulations, and for column *Time* it reports the average runtime for all instances solved to optimality by all formulations. Finally *Sum* reports the total number of instances used in the tests and the overall number of instances solved to optimality by each formulation.

Results in Table 11 show that the 2-index formulation outperforms the two other formulations as it solves 15 instances to optimality out of 56 compared to 10 and 7 instances solved to optimality by the ARF and the 3-index formulation, respectively. When looking at the results by subclass, the 2-index formulation also outperforms the two other formulations in the number of instances solved to optimality, the average optimality gap for unsolved instances and the average runtime for instances solved to optimality per instance subclass. Except for the subclass LR1 where the ARF solves to optimality one more instance compared to the number of instances solved to optimality by 2-index and 3-index formulations. The average by all optimality gap of the 2-index formulation is the lowest with 22.6%, compared to 27.6% and 29.6% for the ARF and the 3-index formulation, respectively. While the average by all runtime of the 2-index formulation is approximately equivalent to the one of the ARF, the 2-index formulation outperforms the two other formulations in the total average runtime for all the instances which is 2688.4 seconds compared to 3030.7 seconds for the ARF and 3166.7 for the 3-index formulation.

7. Conclusion

In this paper, we have designed and implemented a new branch-and-cut algorithm for the MPDPTW. We introduced two new formulations for MPDPTW in addition to the existing 3-index formulation. We have also proposed and tested a transformation of the MPDPTW into a PDPTW. The first new formulation named 2-index formulation does not require the use of a vehicle index to impose pairing and precedence constraints, as in the case of the 3-index formulation. In addition, we introduced valid inequalities to strengthen the LP-relaxation of this formulation. The second new model named asymmetric representatives formulation (ARF) is based on the idea of identifying a cluster of requests by its lowest indexed request. We have first shown how the MPDPTW formulations and algorithms outperform the PDPTW ones. We have then conducted extensive computational experiments on the two new formulations as well as on the 3-index formulation. We have also evaluated the strength of the valid inequalities which were generated only for the 2-index formulation. Finally, we have

Table 11
Summary of the results for PDPTW instances tested on three formulations.

Subclass	# inst.	3-index			ARF			2-index		
		Solved	Gap final (%)	Time (s)	Solved	Gap final (%)	Time (s)	Solved	Gap final (%)	Time (s)
LC1	9	3	38.2	6.4	3	35.2	9.1	5	25.5	1.8
LC2	8	2	10.9	8.4	3	9.6	7.8	7	2.3	0.9
LR1	12	2	28.9	450.2	3	25.8	5.9	2	22.9	26.8
LR2	11	0	22.9	–	0	22.9	–	0	18.6	–
LRC1	8	0	37.9	–	1	32.4	–	1	28.6	–
LRC2	8	0	30.2	–	0	30.4	–	0	23.3	–
Average	–	–	22.4	3166.7	–	20.7	3030.7	–	16.2	2688.4
Avg by all	–	–	29.6	133.8	–	27.6	7.8	–	22.6	8.7
Sum	56	7	–	–	10	–	–	15	–	–

developed and tested an exact and a heuristic algorithm to determine the best input order for the ARF, yielding a significantly smaller and easier problem.

Results show that the ARF outperforms the 2-index and 3-index formulations in solving MPDPTW instances, as it solves more instances to optimality, it has lower runtime on instances solved by all formulations, it yields better lower bounds, and it is capable of solving many large instances with 100 nodes. However, the 2-index formulation outperforms the two other formulations in solving PDPTW instances. We hope these developments will attract more research for other distribution problems, further extending our methods.

Acknowledgments

This work was partly supported by the Canadian Natural Sciences and Engineering Research Council (NSERC) under grants 2015-04893 and 2019-00094. We thank Compute Canada for providing high-performance parallel computing facilities. We thank the editor and two anonymous referees for their valuable suggestions on an earlier version of this paper.

References

Ai, T. J., & Kachitvichyanukul, V. (2009). A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research, 36*(5), 1693–1702.

Alonso-Ayuso, A., Detti, P., Escudero, L. F., & Ortuño, M. T. (2003). On dual based lower bounds for the sequential ordering problem with precedences and due dates. *Annals of Operations Research, 124*(1-4), 111–131.

Ascheuer, N., Fischetti, M., & Grötschel, M. (2001). Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Mathematical Programming, 90*(3), 475–506.

Ascheuer, N., Jünger, M., & Reinelt, G. (2000). A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications, 17*(1), 61–84.

Balas, E., Fischetti, M., & Pulleyblank, W. R. (1995). The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming, 68*(1-3), 241–265.

Baldacci, R., Bartolini, E., & Mingozzi, A. (2011). An exact algorithm for the pickup and delivery problem with time windows. *Operations Research, 59*(2), 414–426.

Bettinelli, A., Cacchiani, V., Crainic, T. G., & Vigo, D. (2019). A branch-and-cut-and-price algorithm for the multi-trip separate pickup and delivery problem with time windows at customers and facilities. *European Journal of Operational Research, 279*(3), 824–839.

Cherkesly, M., Desaulniers, G., & Laporte, G. (2014). Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and last-in-first-out loading. *Transportation Science, 49*(4), 752–766.

Cherkesly, M., Desaulniers, G., & Laporte, G. (2015). A population-based metaheuristic for the pickup and delivery problem with time windows and LIFO loading. *Computers & Operations Research, 62*, 23–35.

Coelho, L., Renaud, J., & Laporte, G. (2016). Road-based goods transportation: a survey of real-world logistics applications from 2000 to 2015. *INFOR: Information Systems and Operational Research, 54*(2), 79–96.

Cordeau, J.-F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research, 54*(3), 573–586.

Dahle, L., Andersson, H., Christiansen, M., & Speranza, M. G. (2019). The pickup and delivery problem with time windows and occasional drivers. *Computers & Operations Research, 109*, 122–133.

Escudero, L. F. (1988). An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research, 37*(2), 236–249.

Ezzat, A., Abdelbar, A. M., & Wunsch, D. C. (2014). An extended EigenAnt colony system applied to the sequential ordering problem. In *Proceedings of the IEEE symposium on swarm intelligence* (pp. 1–7). Orlando, US.

Furtado, M., Munari, P., & Morabito, R. (2017). Pickup and delivery problem with time windows: a new compact two-index formulation. *Operations Research Letters, 45*(4), 334–341.

Goeke, D. (2019). Granular tabu search for the pickup and delivery problem with time windows and electric vehicles. *European Journal of Operational Research, 278*(3), 821–836.

Goksal, F. P., Karaoglan, I., & Altiparmak, F. (2013). A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery. *Computers & Industrial Engineering, 65*(1), 39–53.

Guerriero, F., & Mancini, M. (2003). A cooperative parallel rollout algorithm for the sequential ordering problem. *Parallel Computing, 29*(5), 663–677.

Jans, R., & Desrosiers, J. (2013). Efficient symmetry breaking formulations for the job grouping problem. *Computers & Operations Research, 40*(4), 1132–1142.

Letchford, A. N., & Salazar-González, J.-J. (2016). Stronger multi-commodity flow formulations of the (capacitated) sequential ordering problem. *European Journal of Operational Research, 251*(1), 74–84.

Li, H., & Lim, A. (2001). A metaheuristic for the pickup and delivery problem with time windows. *International Journal of Artificial Intelligence Tools, 12*(2), 160–170.

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

Naccache, S., Côté, J.-F., & Coelho, L. (2018). The multi-pickup and delivery problem with time windows. *European Journal of Operational Research, 269*(1), 353–362.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research, 34*(8), 2403–2435.

Ropke, S., & Cordeau, J.-F. (2009). Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science, 43*(3), 267–286.

Ropke, S., Cordeau, J.-F., & Laporte, G. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks, 49*(4), 258–272.

Savelsbergh, M. W. P. (1990). An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research, 47*(1), 75–85.

Seo, D., & Moon, B. (2003). A hybrid genetic algorithm based on complete graph representation for the sequential ordering problem. In *Proceedings of the Genetic and evolutionary computation conference* (pp. 669–680).

Subramanian, A., Drummond, L. M. A., Bentes, C., Ochi, L. S., & Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research, 37*(11), 1899–1911.

Veenstra, M., Cherkesly, M., Desaulniers, G., & Laporte, G. (2017). The pickup and delivery problem with time windows and handling operations. *Computers & Operations Research, 77*, 127–140.

Wang, C., Mu, D., Zhao, F., & Sutherland, J. W. (2015). A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows. *Computers & Industrial Engineering, 83*, 111–122.