Publié par :
Published by:
Publicación de la:

Faculté des sciences de l'administration
2325, rue de la Terrasse
Pavillon Palasis-Prince, Université Laval
Québec (Québec) Canada G1V 0A6
Tél. Ph. Tel. : (418) 656-3644
Télec. Fax : (418) 656-7047

Disponible sur Internet :
Available on Internet
Disponible por Internet :

http://www4.fsa.ulaval.ca/la-recherche/publications/documents-de-travail/

DOCUMENT DE TRAVAIL 2019-003

A Branch-and-Cut Algorithm for the Multi-Pickup and Delivery Problem with Time Windows

Imadeddine AZIEZ
Jean-François CÔTÉ
Leandro C. COELHO

Document de travail également publié par le Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, sous le numéro CIRRELT-2019-04

Février 2019

Dépôt legal – Bibliothèque et Archives nationales du Québec, 2019
Bibliothèque et Archives Canada, 2019

ISBN 978-2-89524-482-0 (PDF)

A Branch-and-Cut Algorithm for the Multi-Pickup and Delivery Problem with Time Windows

Imadeddine Aziez, Jean-François Côté*, Leandro C. Coelho

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Operations and Decision Systems, 2325, rue de la Terrasse, Université Laval, Québec, Canada, G1V 0A6

**Corresponding author: jean-francois.cote@cirrelt.ca*

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.

Keywords: Multi-pickup and delivery problem, vehicle routing problem, sequential ordering problem, branch-and-cut.

Acknowledgments: This work was partly supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grants 2015-04893 and 2014-05764. We thank Calcul Québec for providing high performance parallel computing facilities.

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 was recently introduced in the literature [Naccache et al., 2018] and finds many practical applications, the most notable in food delivery: customers call a company and request several dishes from different restaurants. The company must then pickup all dishes prior to 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). Practical examples and a review of distribution problems at large can be found in Coelho et al. [2016].

The only work dealing with the multi-pickup and delivery problem with time windows (MPDPTW) is that of Naccache et al. [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 (PDP) and the sequential ordering problem (SOP). These are briefly reviewed next.

Heuristic algorithms for the PDP are the ALNS [Pisinger and Ropke, 2007], the parallel neighborhood descent [Subramanian et al., 2010], and the particle swarm optimization [Ai and Kachitvichyanukul, 2009, Goksal et al., 2013]. Exact algorithms include the branch-and-cut of Ropke et al. [2007], the branch-cut-and-price of Ropke and Cordeau [2009] and the set partitioning-based algorithm of Baldacci et al. [2011]. 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 et al., 2014]. Several methods exist to solve the SOP, including local searches [Savelsbergh, 1990], branch-and-cut [Ascheuer et al., 2000], parallel sequential algorithm [Guerriero and Mancini, 2003], genetic algorithm [Seo and Moon, 2003], and other mathematical programming tools [Letchford and Salazar-González, 2016]. Dual bounds were obtained by lagrangian relaxation by Alonso-Ayuso et al. [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 and 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 $P = \{1, ..., p\}$ defines the pickup nodes, and $D = \{p+1, ..., p+n\}$ is the set of delivery nodes where $|D| = n$ and $p \geq n$. Nodes 0 and $p+n+1$ are the starting and ending depot. Let $R = \{r_1, \ldots, r_n\}$ 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 $d_r \in D$. 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 $A = V \times V$ minus arcs that lead to infeasible solutions. For instance, we omit arc (i, j) if: (i) i is a delivery node and j is one of its pickup nodes,(ii) if $a_i + s_i + t_{ij} > b_j$, (iii) i is the start depot and j is a delivery node, or (iv) i is a pickup node and j is the end depot. A distance $d_{ij} \geq 0$ and a travel time $t_{ij} \geq 0$ are associated with each arc $(i, j) \in A$. Let $A^+(i)$ and $A^-(i)$ be the sets of incoming and outgoing arcs from node $i \in V$.

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.

3 Mathematical models

In this section we propose three mathematical formulations to the problem. The first one is a three-index formulation in which vehicles are identified by an index in the variables, 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:

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

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

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

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

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

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

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

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

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

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

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:

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

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

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

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

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

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

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

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

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

$$x_{ij} \in \{0, 1\} \qquad i \in N, j \in N. \tag{20}$$

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

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 two-index formulation can be improved by adding the new pairing constraints in addition to constraints (16). The new pairing constraints adapted to the MPDPTW are defined as follows:

$$v_{d(r)} = v_i \qquad i \in P_r, r \in R \tag{22}$$
$$v_j = j x_{0j} \qquad j \in N \tag{23}$$
$$v_j \leq j x_{0j} - |N|(x_{0j} - 1) \qquad j \in N \tag{24}$$
$$v_j \geq v_i + |N|(x_{ij} - 1) \qquad i, j \in N \tag{25}$$
$$v_j \leq v_i + |N|(1 - x_{ij}) \qquad i, j \in N. \tag{26}$$

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 Figure 1.a shows an example of the logic behind the ARF with three requests. In this example we show how the ARF breaks the symmetry by imposing the request assignment rule. 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.

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 $W = \{(i, j) \in R \times R |\ i \text{ and } j \text{ cannot be feasibly served together by one vehicle}\}$ 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:

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

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

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

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

$$\sum_{k \in R} y_{kk} \leq |K| \tag{31}$$

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

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

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

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

$$a_i \leq S_i \leq b_i \qquad i \in V \tag{36}$$

$$S_{d_r} \geq S_i + s_i + t_{id_r} \qquad i \in P_r, r \in R \tag{37}$$

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

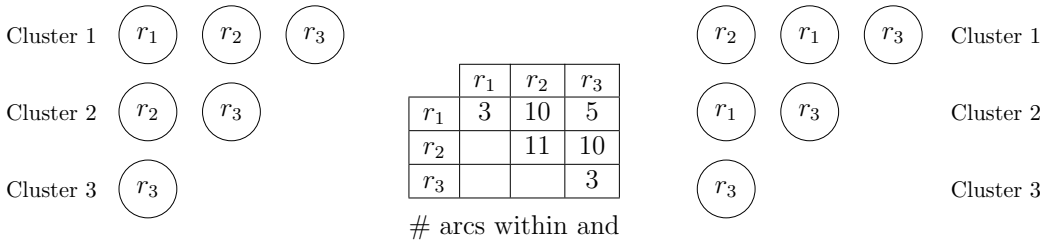
$$S_i \geq 0 \qquad i \in V. \tag{39}$$

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 Figure 1.b which shows how many feasible arcs exist between the three requests of the example.

For the clusters represented in Figure 1.a, we should consider all arcs between requests 1, 2 and 3 for cluster 1 (totaling 3+10+11+10+3+5=42), plus all arcs between requests 2 and 3 for cluster 2 (totaling 11+10+3=24), plus all arcs within request 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 Figure 1.c, cluster 1 continues with 42 arcs, cluster 2 now only contains the arcs between requests 1 and 3 (totaling 3+5+3=11), and cluster 3 only the arcs within request 3, i.e., 3 arcs. The formulation for the clusters represented by Figure 1.c contains only 42+11+3=56 arcs.



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

Figure 1: An example of ARF

An MIP was formulated to determine the best order of input requests that yields the lowest number of arcs in the input graph G. 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:

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

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

$$\sum_{i\in R} u_i^k = 1 \qquad k \in R \tag{42}$$

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

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

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

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

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

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

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. 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 $x(S) = \sum_{i,j \in S} x_{ij}$, and for any node subsets S and T, we define $x(S,T) = \sum_{i \in S} \sum_{j \in T} x_{ij}$.

4.1 Lifted subtour elimination constraints

Lifted subtour elimination constraints 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 et al. [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:

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

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 $F = (k_1, ..., k_l)$ 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 two 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, and the second 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. For a given request $r \in R$, let $F = (0, k_1, k_2, ..., k_l, d_r)$ 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 $S = \{k_1, k_2, ..., k_l\}$ 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 $F = (p, k_1, k_2, ..., k_l, 0)$ be an infeasible path in G such that $p \in P_r$ and $k_i \neq d_r$ for $i = 1, ..., l$. 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 $S = \{k_1, k_2, ..., k_l\}$ be a node subset. Inequality (51) can then be replaced by the following stronger valid inequality:

$$x(S \cup \{p\}) + x(S, 0) \leq |S|. \tag{53}$$

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 $F = (d_r, k_1, k_2, ..., k_l, p)$ be an infeasible path in G such that $p \in P_r$ and d_r is the drop node of r. Let $S = \{k_1, k_2, ..., k_l\}$ 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. For a given pair of nodes $i, j \in N$ such that $(r(i), r(j)) \in W$, the path $F = (i, k_1, k_2, ..., k_l, j)$ is infeasible in G. Let $S = \{k_1, k_2, ..., k_l\}$ 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 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 $F = (k_1, ..., k_l)$ be a feasible path in G, and $S_1, ..., S_l, T \subset N$, such that $k_j \notin S_{j+1}$ for $j = 2, ..., k-1$. If for any integer $h \leq l$ and any node pair $i \in S_h, j \in T$, the path $(i, k_1, ..., k_h, j)$ 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 $F = (k_1, ..., k_l)$ be a feasible path in G, and $S, T_1, ..., T_l \subset N$, such that $k_j \notin T_{j-1}$ for $j = 2, ..., l$. If for any integer $h \leq l$ and any node pair $i \in S, j \in T_h$, the path $(i, k_1, ..., k_h, j)$ 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 $F_k = (k_1, ..., k_l)$ be the optimal path to service the nodes of the request $k \in R$ in a single route for $l > 1$. Then the following inequality is valid for the ARF:

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

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:

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

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

$$\sum_{j\in P_{r_1}} x_{d_{r_2}j} + \sum_{i\in P_{r_2}} x_{id_{r_1}} \leq 1 \tag{61}$$

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

For each node $i \in N$, the following inequality is valid for MPDPTW:

$$t_i \geq \sum_{j\in N} x_{ij}(t_{0j} + t_{ji} + s_j). \tag{63}$$

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 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. If the solution is not feasible 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 et al. [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: $b_k \leftarrow min\{b_k, max\{a_k, max_{i \in A^+(k)}\{b_i + t_{ik}\}\}\ \forall k \in N \mid A^+(k) \neq \emptyset$.

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 $F(r)$ represent all possible paths 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 $k \in P_r$ 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 et al. [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 way starts by checking 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 the set R_1 and the drop node d_r to the set T. In the next step we add as many nodes as possible to the sets $R_1, ..., R_l, T$. 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 the set R_1 and the depot node 0 to the set T. Finally, we add as many nodes as possible to the sets $R_1, ..., R_l, T$. 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 the set R_1 and node j to the set T, and we add as many nodes as possible to sets $R_1, ..., R_l, T$.

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 used in this paper were proposed by Naccache et al. [2018], and they originate from the Li and Lim [2001] instances for the PDPTW. 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 $L\_8\_25\_5$.

Table 1: Instance types

	Without TW		Normal TW		Large TW	
Instance size	Short requests	Long requests	Short requests	Long requests	Short requests	Long 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

6.2 Computational results

The experiments reported here have been performed on a desktop computer equipped with a 2.67 GHz Intel processor operating under the Scientific Linux 6.3. Each test was allowed to run for a maximum of 1 hour and to use up to 28 GB of RAM. The branch-and-cut algorithm is coded in C++ and CPLEX 12.8 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. Finally, we compare the results of the three formulations proposed in this paper. The optimality gap presented in the next sections is calculated as $100 * \left(\frac{BestUpperBound - LowerBound}{BestUpperBound}\right)$. The best upper bound is obtained by checking for each instance the minimum value between the solution obtained by ALNS in Naccache et al. [2018] and any of our solutions selection. This way, we focus on improving the dual bounds for this difficult problem.

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 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 in each scenario, and finally the number of instances solved by all scenarios.

Table 2: ARF with different request input order

	3-index	Worst case ARF	Random case ARF	Best case ARF
Average # of arcs	15037.4	15667.8	12693.8	9779.8
Average runtime (s)	264.9	36.5	31.0	21.1
Avg gap root (%)	27.7	24.9	25.1	22.6
Avg gap final (%)	38.8	36.1	35.2	33.9
Solved	50	60	60	60
Solved by all	50			

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 25.1% 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 2.2% in the best case compared to the worst case, and by 1.3% 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 comparison between the results obtained by the 3-index formulation and the results obtained by the best case of the ARF show 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 15037.4 to 9779.8, the average runtime for instances solved to optimality by the 3-index and the ARF simultaneously from 264.9 seconds to 21.1 seconds, and the average gap for instances unsolved by both formulations from 38.8% to 33.9 %. In addition, the best case of the ARF solved to

optimality 10 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 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, and finally the number of instances solved by all scenarios.

Results show that using any of the four families of valid inequalities yields a small improvement in the number of instances solved to optimality per each scenario and in the average optimality gap for instances unsolved by all the scenarios simultaneously. The largest improvement in the average runtime for instances solved by all is obtained with the generation of FC, while the LSEC have limited impact. However, combining all the valid inequalities yields significant improvements.

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

	LP	LSEC	PC	IRC	FC	Full
Average time (s)	2420.7	2419.9	2369.8	2372.3	2346.8	2303.2
Avg time by all (s)	161.1	145.9	135.3	125.4	82.0	56.6
Average gap (%)	35.4	33.5	35.0	33.6	34.2	31.3
Solved	42	41	43	44	44	46
Solved by all	41					

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. The average preprocessing time is 0.6 seconds, however, this can be up to 15.5 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].

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)	3052.8	2303.2	2786.9	2210.4	2711.6	1826.7
Avg time by all (s)	615.1	167.9	155.2	106.6	41.6	7.8
Avg gap root (%)	46.5	28.6	48.1	27.7	46.7	22.6
Avg gap final (%)	43.7	36.4	47.5	38.8	46.3	33.9
Solved	22	46	30	50	32	60

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 optmality per formulation has significantly increased due to the use of the preprocessing techniques. The 2-index formulation solves 24 more instances, the 3-index formulation solves 20 more instances, and the ARF solves 28 more instances.

6.2.4 Comparison between the three formulations

In this section, we compare the strength of the three proposed formulations and their performance in solving the MPDPTW. Table 5 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 by each formulation.

The results in Table 5 show that the ARF outperforms the two other formulations as it solves 60 instances to optimality out of 120 compared to 50 and 46 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 7.8 seconds compared to 106.6 seconds and 167.9 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 33.9% 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 39.1% which means that its LP relaxation is the strongest among the proposed formulations. In Table 5, 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.

Table 6 reports the number of instances solved by each one of the formulations classified according to the number of nodes per instance. 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 5: Summary of the results for all instances tested on three formulations

	2-index				3-index				ARF			
Instance	Solved	Gap root (%)	Gap final (%)	Time (s)	Solved	Gap root (%)	Gap final (%)	Time (s)	Solved	Gap root (%)	Gap final (%)	Time (s)
l_4_25	3	27.6	10.8	152.8	5	26.7	-	262.5	5	19.8	-	126.0
l_4_35	1	33.4	22.6	3312.7	2	33.7	18.3	711.7	3	28.3	13.2	398.5
l_4_50	0	44.0	36.0	-	0	47.4	37.6	-	0	41.1	27.8	-
l_4_100	0	54.4	53.3	-	0	58.0	56.3	-	0	52.9	50.5	-
l_8_25	5	9.3	-	3.2	5	8.8	-	2.0	5	3.3	-	0.2
l_8_35	5	14.9	-	25.9	5	10.0	-	4.8	5	4.7	-	1.0
l_8_50	2	22.4	12.5	966.7	4	21.8	20.2	1865.5	4	12.1	15.2	33.0
l_8_100	0	36.4	32.5	-	0	35.9	32.7	-	0	19.8	17.4	-
n_4_25	5	8.3	-	2.3	5	4.7	-	0.5	5	1.8	-	0.2
n_4_35	5	15.6	-	78.0	5	8.0	-	2.6	5	2.2	-	0.4
n_4_50	2	23.4	15.0	390.9	2	20.0	10.0	511.9	5	9.8	-	57.8
n_4_100	0	38.6	35.4	-	0	38.8	35.6	-	0	21.9	19.0	-
n_8_25	5	1.4	-	0.2	5	1.8	-	0.2	5	0.0	-	0.1
n_8_35	5	2.6	-	0.7	5	1.2	-	0.3	5	0.0	-	0.0
n_8_50	5	4.8	-	27.1	5	3.9	-	16.0	5	1.3	-	0.3
n_8_100	2	10.3	1.5	1300.0	0	9.7	6.7	-	5	2.6	-	29.2
w_4_25	0	33.2	19.5	-	0	33.6	25.4	-	0	33.4	25.5	-
w_4_35	0	39.4	29.6	-	0	40.0	37.3	-	0	40.0	37.1	-
w_4_50	0	46.2	39.7	-	0	46.4	45.6	-	0	46.4	45.2	-
w_4_100	0	54.9	53.0	-	0	55.5	54.8	-	0	55.4	54.5	-
w_8_25	1	26.5	11.8	215.6	1	26.0	14.8	709.1	2	22.8	18.5	391.6
w_8_35	0	35.2	23.6	-	1	32.4	26.9	1177.2	1	27.3	25.8	13.2
w_8_50	0	47.1	38.3	-	0	47.0	43.3	-	0	44.8	41.2	-
w_8_100	0	56.3	53.4	-	0	54.2	51.3	-	0	51.7	48.7	-
Average	-	28.6	19.3	2303.2	-	27.7	20.0	2210.4	-	22.6	17.0	1826.7
Avg by all	-	43.2	36.4	167.9	-	43.7	38.8	106.6	-	39.1	33.9	7.8
Sum	46	-	-	-	50	-	-	-	60	-	-	-

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 exist-

Table 6: Number of solved instances classified according to the number of nodes per instance

Size	Solved by 2-index	Solved by 3-index	Solved by ARF
25	19	21	22
35	16	18	19
50	9	11	14
100	2	0	5
Sum	46	50	60

ing 3-index formulation. 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 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 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. The results indicate that the pairing and precedence constraints have the largest impact in decreasing the optimality gap. We have also 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 as it solves more instances to optimality, it has less runtime on instances solved by all formulations, it yields better lower bounds, and it is capable of solving many large instances with 100 customer nodes. We hope these developments will attract more research for other distribution problems further extending our methods.

Acknowledgements

This work was partly supported by the Canadian Natural Sciences and Engineering Research Council (NSERC) under grants 2015-04893 and 2014-05764. We thank Calcul Québec for providing high performance parallel computing facilities.

References

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

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

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

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

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

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

L.C. Coelho, J. Renaud, and G. Laporte. 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, 2016.

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

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

A. Ezzat, A. M. Abdelbar, and D. C. Wunsch. An extended EigenAnt colony system applied to the sequential ordering problem. In *2014 IEEE Symposium on Swarm Intelligence*, pages 1–7, Orlando, US, 2014.

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

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

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

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

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

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

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

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

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

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

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

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

D.I. Seo and B.R. Moon. A hybrid genetic algorithm based on complete graph representation for the sequential ordering problem. In *Genetic and Evolutionary Computation Conference*, pages 669–680, 2003.

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