A branch-and-cut algorithm for the generalized traveling salesman problem with time windows

Yuan Yuan, Diego Cattaruzza, Maxime Ogier, Frédéric Semet

A branch-and-cut algorithm for the generalized traveling salesman problem with time windows

Yuan Yuan, Diego Cattaruzza, Maxime Ogier, Frédéric Semet
Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL Lille, France
yyuaninria@gmail.com, {diego.cattaruzza, maxime.ogier, frederic.semet}@centralelille.fr

Abstract: The generalized traveling salesman problem with time windows (GTSPTW) is defined on a directed graph where the vertex set is partitioned into clusters. One cluster contains only the depot. Each vertex is associated with a time interval, the time window, during which the visit must take place if the vertex is visited. The objective is to find a minimum cost tour starting and ending at the depot such that each cluster is visited exactly once and time constraints are respected, i.e., for each cluster, one vertex is visited during its time window. In this paper, two integer linear programming formulations for GTSPTW are provided as well as several problem-specific valid inequalities. A branch-and-cut algorithm is developed in which the inequalities are separated dynamically. To reduce the computation times, an initial upper bound is provided by a simple and fast heuristic. We propose different sets of instances characterized by their time window structures. Experimental results show that our algorithm is effective and instances including up to 30 clusters can be solved to optimality within one hour.

Keywords: generalized traveling salesman problem; time-windows; branch-and-cut; trunk delivery.

1 Introduction

E-commerce is a thriving market around the world and is very well suited to the busy lifestyle of today's customers. An annual survey conducted by the analytics firm comScore and UPS revealed that American consumers purchased more things online than in stores in 2016 (Farber, 2016). eMarketer estimated that e-commerce sales would reach $4 trillion in 2020 (eMarketer, 2018). It is clear that this growing e-commerce poses a major challenge to transportation companies, especially with regard to last mile delivery. Nowadays, the most common delivery service is home/workplace delivery (Lowe and Rigby, 2014). Customers wait at home or at work to get their online orders. Besides, companies like Amazon and FedEx are developing locker delivery. When customers shop online, they can choose a nearby locker as their pickup location. In the past two years, a new concept called *trunk delivery* has been introduced. Here, customers' orders can be delivered to the trunks of their cars. Volvo launched its world-first in-car delivery service in Sweden in 2016. The courier has a one-time digital code to get access to the trunk of the car. Trunk delivery is different from home delivery and locker delivery since the car moves during the day and may be in different locations during different periods of time. As a consequence, synchronization

between the car and the courier is required to perform the delivery. In this article, we provide two mathematical programming models, and we develop an efficient exact method for the last mile delivery problem that combines all these delivery services: home/workplace, locker, and car trunk. We focus on the one vehicle case, i.e., we assume that a single vehicle can deliver all the customers on the same route. In Figure 1, we give an example of the real-life case. Four customers are represented with their associated locations into a dotted circle. Every possible delivery location has a time interval that represents the time window (TW). In the case of a home or trunk delivery, the TW represents when the customer or his/her car would be present at that location. In the case of a locker, the TW represents the period the courier can deliver the parcel before the customer picks it up. The problem consists in determining jointly the location visited for each customer and the sequence of visits while satisfying the TW restrictions.

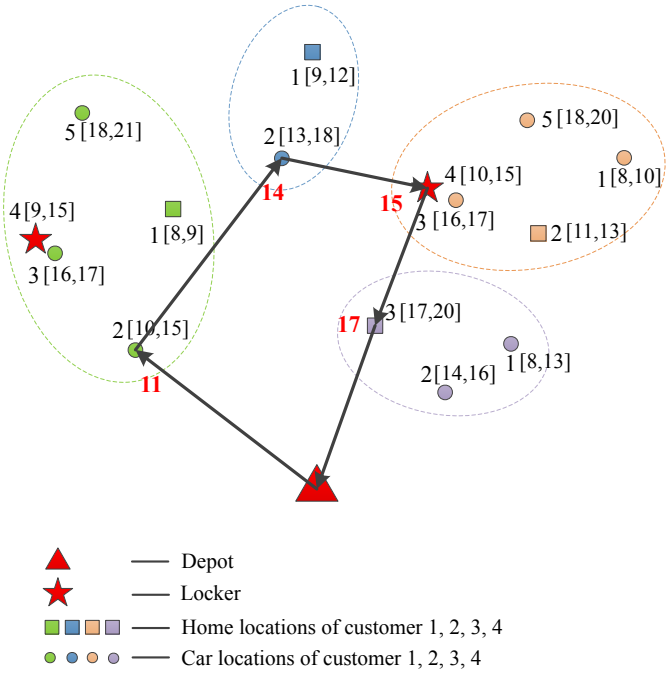


Figure 1: An example of GTSPTW instance.

The problem addressed in this paper is the generalized traveling salesman problem with time windows (GTSPTW). To the best of our knowledge, this problem has not been studied yet. It is related to the generalized traveling salesman problem (GTSP) (Fischetti et al., 1997) where TWs are not present, and to the traveling salesman problem with time windows (TSPTW) where all clusters contain a single vertex.

This article aims to provide an efficient exact solution method for the GTSPTW. The main contributions of the paper are as follows: 1) we study a new problem and present two formulations for the GTSPTW. This problem is of great interest in the context of last mile delivery, 2) we propose several valid inequalities for GTSPTW, 3) we develop procedures to separate these inequalities within a branch-and-cut algorithm, 4) we present a simple and fast heuristic for GTSPTW to get an initial solution, 5) we assess the efficiency of our algorithm on different sets of instances that we generated for the GTSPTW.

The remainder of this paper is organized as follows. A formal description of the problem and two mathematical models are provided in Section 2. Section 3 presents the related literature. Section 4 de-

scribes some valid inequalities for the GTSPTW. A general framework of the branch-and-cut algorithm is given in Section 5, including preprocessing, an initial heuristic to compute an upper bound, and separation procedures for the proposed valid inequalities. Section 6 gives details about the generation of three groups of instances and reports the computational results. Finally, conclusions are drawn in Section 7.

2 Problem definition and mathematical modeling

The GTSPTW can be formally defined as follows: given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, the set of vertices $\mathcal{V} = \{0, 1, ..., N\}$ is partitioned into $\mathcal{C}_0 = \{0\}, \mathcal{C}_1, ..., \mathcal{C}_K$ clusters. $\mathcal{K} = \{0, 1, ..., K\}$ denotes the cluster index set. Cluster $\mathcal{C}_0$ contains only the starting and ending vertex, i.e. the depot. A TW $[E_i, L_i]$, is associated with each vertex $i \in \{0, 1, ..., N\}$ with $[E_0, L_0] = [0, T]$ representing the optimization time horizon. A visit can only be made to a vertex during its TW, and an early arrival leads to waiting time while a late arrival causes infeasibility. There is no assumption on the TWs for a given cluster, i.e., TWs can overlap or be disjointed. Arcs are only defined between vertices belonging to different clusters, that is, $\mathcal{A} \subseteq \{(i,j) | i \in \mathcal{C}_k, j \in \mathcal{C}_l, k \neq l, k, l \in \mathcal{K}\}$. A traveling cost C_{ij} and a traveling time T_{ij} are associated with each arc $(i,j) \in \mathcal{A}$. We call an arc (i,j) *feasible* if $E_i + T_{ij} \leq L_j$, which means that vertex j can be reached from vertex i through arc (i,j). $\mathcal{A}$ is defined as the set of *feasible* arcs, which is a subset of $\{(i,j) | i \in \mathcal{C}_k, j \in \mathcal{C}_l, k \neq l, k, l \in \mathcal{K}\}$.

The objective of the GTSPTW is to find a minimum cost tour starting and ending at the depot such that each cluster is visited exactly once and the TW constraints are respected, i.e., one vertex of each cluster is visited during its time window.

Let us introduce the following notation. For any set $\mathcal{S} \subset \mathcal{V}$, $\delta^+(\mathcal{S}) = \{(i,j) \in \mathcal{A} | i \in \mathcal{S}, j \notin \mathcal{S}\}$, $\delta^-(\mathcal{S}) = \{(i,j) \in \mathcal{A} | i \notin \mathcal{S}, j \in \mathcal{S}\}$. For simplicity, when $\mathcal{S} = \{i\}$, we use $\delta^+(i)$ and $\delta^-(i)$ as opposed to $\delta^+(\{i\})$ and $\delta^-(\{i\})$.

To model the GTSPTW, we define three set of variables. For all $(i,j) \in \mathcal{A}$, let x_{ij} be a binary variable equal to one if and only if arc $(i,j) \in \mathcal{A}$ belongs to the tour. For all $i \in \mathcal{V}$, let y_i be a binary variable equal to one if $i \in \mathcal{V}$ belongs to the tour, and t_i be the service time at vertex $i \in \mathcal{V}$. For the depot, the service time actually corresponds to the departure time. A first compact mathematical programming formulation $\mathcal{F}1$ is as follows:

$$(\mathcal{F}1) \quad \text{minimize} \quad \sum_{(i,j) \in \mathcal{A}} C_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{C}_k} y_i = 1 \qquad \forall k \in \mathcal{K}, \tag{2}$$

$$\sum_{(i,j) \in \delta^+(i)} x_{ij} = y_i \qquad \forall i \in \mathcal{V}, \tag{3}$$

$$\sum_{(j,i) \in \delta^-(i)} x_{ji} = y_i \qquad \forall i \in \mathcal{V}, \tag{4}$$

$$E_i y_i \leq t_i \leq L_i y_i \qquad \forall i \in \mathcal{V}, \tag{5}$$

$$t_i - t_j + T_{ij} x_{ij} \leq L_i y_i - E_j y_j - (L_i - E_j) x_{ij} \qquad \forall (i,j) \in \mathcal{A}, j \neq 0, \tag{6}$$

$$t_i + T_{i0} x_{i0} \leq L_0 \qquad \forall i \in \mathcal{V} \setminus \{0\}, \tag{7}$$

$$y_i \in \{0,1\} \qquad \forall i \in \mathcal{V}, \tag{8}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}, \tag{9}$$

$$t_i \geq 0 \qquad \forall i \in \mathcal{V}. \tag{10}$$

The objective function (1) minimizes the total traveling cost. Constraints (2) ensure that exactly one vertex from each cluster is visited. Constraints (3) and (4) are flow conservation constraints. Constraints (5) ensure that each vertex is visited during its TW. Constraints (6) ensure that the service times are consistent. If vertex j is visited just after vertex i, then constraint (6) will ensure $t_j \geq t_i + T_{ij}$. In addition, constraints (6) eliminate subtours since they generalize the subtour elimination constraints of Miller, Tucker and Zemlin for the traveling salesman problem (Miller et al., 1960). Constraints (7) ensure that the tour ends at the depot before its TW closes. Constraints (8) $\sim$ (10) are related to variable definitions.

The second formulation is based on the following observation. Since only one vertex is selected in each cluster, we can define one time variable per cluster instead of defining a time variable for every vertex as above. Let $\tau_k \geq 0$, $k \in \mathcal{K}$ be the service time at cluster k.

In the second formulation $\mathcal{F}2$, the objective function and constraints (2)$\sim$(4), (8)$\sim$(9) are as in $\mathcal{F}1$. Constraints (5)$\sim$(7), (10) are replaced by (11)$\sim$(13), (14) respectively. We obtain the following compact model:

$(\mathcal{F}2)$ minimize (1)

s.t. (2) $\sim$ (4)

(8) $\sim$ (9)

$$\sum_{i \in C_k} E_i y_i \leq \tau_k \leq \sum_{i \in C_k} L_i y_i \qquad \forall k \in \mathcal{K}, \tag{11}$$

$$\tau_h - \tau_k + \sum_{\substack{i \in C_h \\ j \in C_k}} T_{ij} x_{ij} \leq \sum_{i \in C_h} L_i y_i - \sum_{j \in C_k} E_j y_j - \sum_{\substack{i \in C_h \\ j \in C_k}} (L_i - E_j) x_{ij} \qquad \forall h \in \mathcal{K}, k \in \mathcal{K} \setminus \{0\}, \tag{12}$$

$$\tau_k + \sum_{i \in \mathcal{C}_k} T_{i0} x_{i0} \leq L_0 \qquad \forall k \in \mathcal{K} \setminus \{0\}, \tag{13}$$

$$\tau_k \geq 0 \qquad \forall k \in \mathcal{K}. \tag{14}$$

3 Literature review

To the best of our knowledge, there is no existing literature on the GTSPTW. However, there exist works addressing the GTSP. The GTSP is defined on a graph where the vertex set is partitioned into clusters. The problem consists in finding a minimum cost tour which visits exactly one vertex of each cluster. In the GTSP, TWs are not present. In the literature, there are various approaches to solve the GTSP.

One approach is to transform an instance of the GTSP into an instance of the well-studied Traveling Salesman Problem (TSP), and then solve it by applying algorithms for the TSP (Noon and Bean, 1993;

Dimitrijević and Šarić, 1997; Laporte and Semet, 1999). At first glance, this approach seems promising. However, the resulting instances are difficult to solve for the existing TSP solvers since the produced instances have a rather unusual structure, and a near-optimal solution for the TSP instance may correspond to an infeasible solution for the related GTSP instance (Karapetyan and Gutin, 2012).

Another approach consists in developing exact algorithms. However, the existing literature is quite limited. Noon and Bean (1991) presented a branch-and-bound approach for the asymmetric GTSP (AGTSP). They proposed a Lagrangian relaxation to compute a lower bound and a heuristic to compute an upper bound. Non-optimal arcs and nodes were identified and eliminated based on the reduced costs. This method was tested on a set of randomly generated instances, and the results showed that they could solve instances with up to 104 nodes and 8 clusters. Fischetti et al. (1997) proposed an efficient branch-and-cut algorithm to solve the AGTSP. They developed exact and heuristic separation procedures for some classes of facet-defining inequalities. They also generated a library of GTSP instances called GTSP-LIB by taking TSP-LIB instances and performing a clustering procedure on the nodes. Their algorithm could solve instances with up to 89 clusters and 442 nodes.

A different approach to solve the GTSP is to develop heuristics. Gutin and Karapetyan (2010) proposed a memetic algorithm combining genetic and powerful local search algorithms. They reported excellent results on the GTSP-LIB instances, with computation times less than 60 seconds and most of the solutions within 0.2% of the best-known values. Helsgaun (2015) extended the Lin-Kernighan-Helsgaun TSP heuristic (Helsgaun, 2000, 2009) to the GTSP. The resulting algorithm improved the solution quality on GTSP-LIB instances compared with the memetic algorithm proposed in Gutin and Karapetyan (2010), at the expense of more computation time. Smith and Imeson (2017) presented an algorithm solver based on adaptive large neighborhood search. Their results showed that given the same amount of computation time, their algorithm was competitive on instances from the GTSP-LIB and other libraries.

The GTSPTW is also related to the TSPTW. When all the clusters of the GTSPTW are singletons, i.e., they contain only one vertex, the problem reduces to the TSPTW. Ascheuer et al. (2001) proposed several formulations for the asymmetric TSPTW and compared them within a branch-and-cut scheme. They incorporated techniques such as data pre-processing, primal heuristics, local search and variable fixing, in addition to separation algorithms. Dash et al. (2012) presented an extended formulation for the TSPTW based on the partitioning of the TW into sub-windows, which they called buckets. The LP relaxation of this formulation provided strong lower bounds. Strong valid inequalities (bucket sequential ordering polytope inequalities) were generated and incorporated in a branch-and-cut framework. Their results showed that the proposed formulation was effective and solved several previously unsolved benchmark instances.

Although there is no article related to the GTSPTW, one can find related papers in the multi-vehicle case. Ghiani and Improta (2000) extended the GTSP to the generalized vehicle routing problem (GVRP) by introducing quantities to be delivered to customers and considering vehicles with limited capacity. Only a few works address the multi-vehicle case with TWs, but with some restrictions on the TWs. Moccia et al. (2012) proposed a tabu search method for what they called the Generalized-VRPTW. However, they define a TW for each cluster while a TW is associated with every vertex in the GTSPTW. They proved the effectiveness of the method by testing it on GVRP instances and multi-depot VRPTW instances.

Recently, Reyes et al. (2017) examined the special case where TWs within the same cluster do not overlap. They were inspired by the trunk delivery system we mentioned in Section 1 and considered what they called the VRP with Roaming Delivery Locations (VRPRDL). The authors developed construction and improvement heuristics for the problem, and their results illustrated the advantage of applying the trunk delivery over the traditional home delivery. Following this work, Ozbaygin et al. (2017) formulated VRPRDL as a set-partitioning problem and proposed a branch-and-price algorithm. Moreover, they came up with a hybrid delivery strategy combining trunk delivery and home delivery, in which case the TWs within a cluster are no longer non-overlapping. Their results revealed that employing this strategy led to an average savings of nearly 20% compared with the standard delivery system when only home delivery is used.

4 Valid inequalities for the GTSPTW

In this section, we present some inequalities we developed for GTSPTW, mainly adapted from valid inequalities for the Steiner tree problem, the GTSP or the TSPTW. Some of these inequalities are defined on x variables only, other inequalities also involve y variables. We also present lifted versions of some valid inequalities. Section 4.1 proposes a lifted version of the Miller-Tucker-Zemlin (MTZ) inequalities provided in formulations $\mathcal{F}1$ and $\mathcal{F}2$. Sections 4.2 and 4.3 propose polynomial-size families of valid inequalities while sections 4.4 to 4.7 provide exponential-size families of valid inequalities.

4.1 Strengthened MTZ-inequalities

Desrochers and Laporte (1991) observed that the subtour elimination constraints presented in the MTZ version as in (6) can be lifted by taking the reverse arcs $(j,i) \in \mathcal{A}$ into account.

Proposition 4.1. *In formulation $\mathcal{F}1$, for all $i,j \in \mathcal{V} \setminus \{0\}$ such that arcs (i,j) and $(j,i) \in \mathcal{A}$, time constraints* (6) *can be lifted as follows:*

$$t_i - t_j + T_{ij}x_{ij} + min\{-T_{ji}, E_j - E_i\}x_{ji} \leq L_i y_i - E_j y_j - (L_i - E_j)(x_{ij} + x_{ji}) \tag{15}$$

Proof. Consider the general constraints

$$t_i - t_j + T_{ij}x_{ij} + \alpha_{ji}x_{ji} \leq L_i y_i - E_j y_j - (L_i - E_j)(x_{ij} + x_{ji}).$$

If $x_{ji} = 0$, then the constraints is the classical MTZ constraint. Otherwise, we have $x_{ji} = 1$, and $x_{ij} = 0$, $y_i = 1$, $y_j = 1$. Hence the constraint can be written as:

$$t_i - t_j + \alpha_{ji} \leq 0.$$

Only two cases need to be considered.

Case 1: $E_j + T_{ji} \geq E_i$. This means whenever vertex j is left, there will be no waiting time at i. Hence, $t_i = t_j + T_{ji}$. Since $\alpha_{ji} \leq t_j - t_i$, then $\alpha_{ji} \leq -T_{ji}$.

Case 2: $E_j + T_{ji} < E_i$. If j is left at the beginning of its TW, the service at i will be after $E_j + T_{ji}$. Services at i and j can be spaced with a maximum value of $E_i - E_j$, i.e., $t_i \leq t_j + (E_i - E_j)$. Since $t_i \leq t_j - \alpha_{ji}$, then $\alpha_{ji} \leq E_j - E_i$.

Note that by definition of case 1, we have $-T_{ji} \leq E_j - E_i$, and by definition of case 2, we have $E_j - E_i < -T_{ji}$. Thus in general, $\alpha_{ji} \leq min\{-T_{ji}, E_j - E_i\}$. □

Similarly, for formulation $\mathcal{F}2$ in which the time variables are defined for clusters, we can lift constraints (12) and we have:

Proposition 4.2. *In formulation $\mathcal{F}2$, for all $h, k \in \mathcal{K} \setminus \{0\}$ such that there exists at least a pair of arcs (i,j) and $(j,i) \in \mathcal{A}$ with $i \in \mathcal{C}_h$ and $j \in \mathcal{C}_k$, time constraints (12) can be lifted as:*

$$\begin{aligned} &\tau_h - \tau_k + \sum_{\substack{i \in \mathcal{C}_h \\ j \in \mathcal{C}_k}} T_{ij} x_{ij} + \sum_{\substack{i \in \mathcal{C}_h \\ j \in \mathcal{C}_k}} min\{-T_{ji}, E_j - E_i\} x_{ji} \\ &\leq \sum_{i \in \mathcal{C}_h} L_i y_i - \sum_{j \in \mathcal{C}_k} E_j y_j - \sum_{\substack{i \in \mathcal{C}_h \\ j \in \mathcal{C}_k}} (L_i - E_j)(x_{ij} + x_{ji}) \end{aligned} \tag{16}$$

4.2 Arc selection inequalities

It is obvious that in any feasible solution at most one arc (i,j) or (j,i) is selected. Therefore, we have:

$$x_{ij} + x_{ji} \leq y_i \qquad \forall i, j \in \mathcal{V} \text{ such that } (i,j), (j,i) \in \mathcal{A}. \tag{17}$$

Since only one vertex is selected from each cluster, we can lift inequalities (17) and obtain:

Proposition 4.3. *The following constraints are valid inequalities for the GTSPTW:*

$$\sum_{j \in \mathcal{C}_k} x_{ij} + \sum_{j \in \mathcal{C}_k} x_{ji} \leq y_i \qquad \forall i \in \mathcal{V}, k \in \mathcal{K}, i \notin \mathcal{C}_k \text{ and } \exists j \in \mathcal{C}_k \text{ such that } (i,j), (j,i) \in \mathcal{A}. \tag{18}$$

Note that summing up constraints (18) over all vertices of cluster $\mathcal{C}_l, l \neq k$ provides a special case of the generalized subtour elimination constraints (GSEC) restricted to tours of length 2.

4.3 Arc-or-vertex inequalities

In a feasible solution, a vertex and an arc may not be simultaneously present due to TW constraints. We introduce *arc-or-vertex inequalities* to exploit this property and these inequalities impose that at most one of the vertex or the arc is chosen. Let $\mathcal{C}_k^{ij}$ be the subset of $\mathcal{C}_k$ containing vertices that cannot be visited before or after arc (i,j). A vertex $h \in \mathcal{C}_k$ belongs to $\mathcal{C}_k^{ij}$ if there is no feasible path going from h to j including (i,j) and there is no feasible path going from i to h traversing (i,j). Figure 2 depicts this situation. Formally, a vertex h belongs to $\mathcal{C}_k^{ij}$ if and only if:

1. $h \in \mathcal{C}_k$,
2. $E_h + SP_{hi} > L_i$, or $E_h + SP_{hi} + T_{ij} > L_j$,
3. $E_j + SP_{jh} > L_h$, or $E_i + T_{ij} + SP_{jh} > L_h$.

where SP_{ij} is the shortest traveling time between vertices i and j. When the triangle inequality is not satisfied, the shortest traveling time SP_{ij} going from vertex i to vertex j can include the visit of other vertices and can be lower than T_{ij}.

Proposition 4.4. *The following constraints are valid inequalities for the GTSPTW:*

$$x_{ij} + \sum_{h \in \mathcal{C}_k^{ij}} y_h \leq 1 \qquad \forall (i,j) \in \mathcal{A}, i, j \neq 0, \forall k \in \mathcal{K}, i, j \notin \mathcal{C}_k. \tag{19}$$

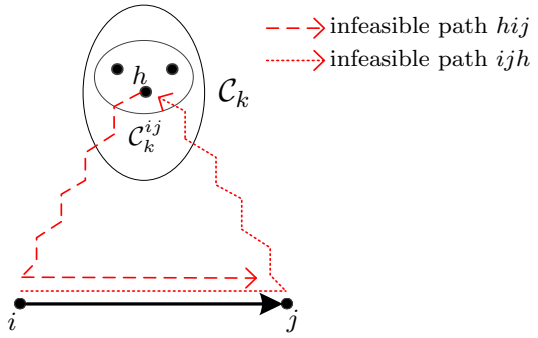
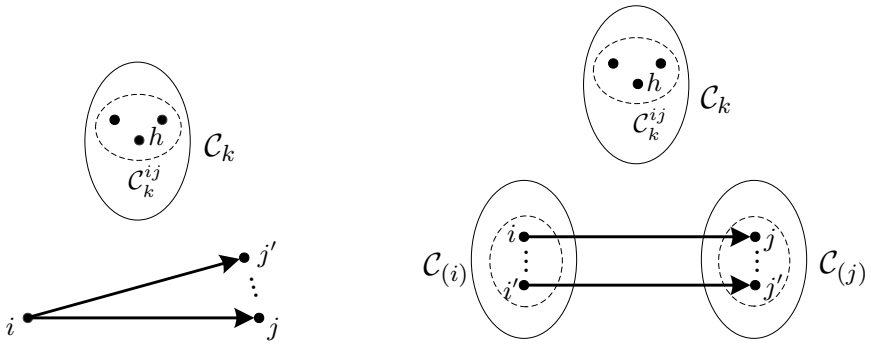


Figure 2: Arc-or-vertex inequalities example.

We can lift the arc-or-vertex inequalities in two different ways as depicted in Figure 3. First, given an arc $(i,j) \in \mathcal{A}$ and a cluster $k \in \mathcal{K}$ with a set of incompatible vertices $\mathcal{C}_k^{ij}$, other arcs (i,j') may lead to the same set of incompatible vertices: $\mathcal{C}_k^{ij'} = \mathcal{C}_k^{ij}$. In this case, constraints (19) can be lifted by summing up over such variables $x_{i,j'}$ since at most one of these arcs can be present in a feasible solution. Thus, inequalities (19) can be strengthened as follows:

Proposition 4.5. *For all $(i,j) \in \mathcal{A}, i, j \neq 0$ and for all $k \in \mathcal{K}$ such that $i, j \notin \mathcal{C}_k$, consider $\mathcal{V}_k^{ij} = \{j' \in \mathcal{V} \setminus \mathcal{C}_k \mid \mathcal{C}_k^{ij'} = \mathcal{C}_k^{ij}\}$, then constraints (19) can be lifted as:*

$$\sum_{j' \in \mathcal{V}_k^{ij}} x_{ij'} + \sum_{h \in \mathcal{C}_k^{ij}} y_h \leq 1 \tag{20}$$



(a) Lift 1 (b) Lift 2

Figure 3: Lifting of arc-or-vertex inequalities.

Second, given an arc $(i,j) \in \mathcal{A}$ and a cluster $k \in \mathcal{K}$ with a set of incompatible vertices $\mathcal{C}_k^{ij}$, other arcs (i',j') may lead to the exactly same set of incompatible vertices: $\mathcal{C}_k^{i'j'} = \mathcal{C}_k^{ij}$. When i and i' belong to the same cluster, respectively j and j' belong to the same cluster, then constraints (19) can be lifted by summing up over such variables $x_{i',j'}$ since such arcs are defined between the same pair of clusters and at most one appears in any feasible solution.

Proposition 4.6. *Let $\mathcal{C}_{(i)}$ be the cluster containing vertex i. For all $(i,j) \in \mathcal{A}, i,j \neq 0$ and for all $k \in \mathcal{K}$ such that $i,j \notin \mathcal{C}_k$, consider $\mathcal{A}_k^{ij} = \{(i',j') \in \mathcal{A} \mid i' \in \mathcal{C}_{(i)}, j' \in \mathcal{C}_{(j)}, \mathcal{C}_k^{i'j'} = \mathcal{C}_k^{ij}\}$, then inequalities (19) can be strengthened as:*

$$\sum_{(i',j') \in \mathcal{A}_k^{ij}} x_{i'j'} + \sum_{h \in \mathcal{C}_k^{ij}} y_h \leq 1 \tag{21}$$

4.4 Generalized subtour elimination constraints (GSEC)

Although constraints (6) eliminate tours not including the depot in any feasible integer solution, subtours may be present when the integer requirement on variables x and y is relaxed. Thus, the subtour elimination constraints (SEC) defined by Dantzig et al. (1954) for the TSP can increase the linear relaxation value. These constraints can be generalized to take into account the presence of clusters (Fischetti et al., 1997). For subsets of clusters, they can be expressed as follows:

Proposition 4.7. *The constraints*

$$\sum_{(i,j) \in \delta^+(\mathcal{S})} x_{ij} \geq 1 \qquad \forall \mathcal{S} = \cup_{h \in \mathcal{H}} \mathcal{C}_h,\ \mathcal{H} \subset \mathcal{K},\ 2 \leq |\mathcal{H}| \leq K-1. \tag{22}$$

are valid for the GTSPTW.

4.5 SOP inequalities

SOP inequalities are based on the notion of precedence between pairs of vertices and were introduced by Balas et al. (1995) in the context of the precedence-constrained Asymmetric Traveling Salesman Problem (ATSP), also known as the Sequential Ordering Problem (SOP). These inequalities are also effective for TSPTW where precedences between nodes are inferred based on the TW restrictions (Dash et al., 2012; Ascheuer et al., 2001). Here we extend the SOP inequalities to the GTSPTW.

Recall that $\mathcal{C}_{(i)}$ denotes the cluster containing vertex i. We say that a vertex $i \in \mathcal{V}$ *precedes* vertex $j \in \mathcal{V} \setminus \mathcal{C}_{(i)}$ if i has to be visited before j in any feasible solution. We denote this relation as $i \prec j$. When the triangle inequality is satisfied, the precedence between two vertices i and j is defined as:

$$i \prec j \text{ if } E_i + T_{ij} \leq L_j \text{ and } E_j + T_{ji} > L_i. \tag{23}$$

The relation (23) can be extended to the case where the triangle inequality is not satisfied. Then, we consider SP_{ij} the shortest traveling time from vertex i to vertex j. The path with the shortest traveling time may include other vertices since the triangle inequality is not satisfied. Thus, the precedence relation becomes:

$$i \prec j \text{ if } E_i + SP_{ij} \leq L_j \text{ and } E_j + SP_{ji} > L_i. \tag{24}$$

If the triangle inequality is satisfied, both relations (24) and (23) are equivalent since $SP_{ij} = T_{ij}$.

For any given $i \in \mathcal{V} \setminus \{0\}$, we define $\pi(i) = \{j \in \mathcal{V} \setminus \mathcal{C}_{(i)} : j \prec i\}$ the set of vertices that precede vertex i, and $\sigma(i) = \{j \in \mathcal{V} \setminus \mathcal{C}_{(i)} : i \prec j\}$ the set of vertices that succeed vertex i. In the following, we summarize the classes of SOP inequalities we used in our implementation. For the ease of explanation, we introduce

the following notations. For any two vertex sets $\mathcal{U}, \mathcal{W} \subseteq \mathcal{V}$, let $(\mathcal{U} : \mathcal{W}) = \{(i,j) \in \mathcal{A} | i \in \mathcal{U}, j \in \mathcal{W}\}$. This set correspond to the cut between the two vertex sets $\mathcal{U}$ and $\mathcal{W}$. For any set of vertices $\mathcal{S} \subseteq \mathcal{V}$, we note $\bar{\mathcal{S}} = \mathcal{V} \setminus \mathcal{S}$ its complement.

Proposition 4.8. *For $\mathcal{S} \subseteq \mathcal{V} \setminus \{0\}$, $i \in \mathcal{S}$ such that $\pi(i) \neq \emptyset$, the predecessor inequalities (π-inequalities):*

$$x((\mathcal{S} \setminus \pi(i)) : (\bar{\mathcal{S}} \setminus \pi(i))) \geq y_i \tag{25}$$

are valid for GTSPTW. The π-inequalities (25) can be strengthened as:

$$x((\mathcal{S} \setminus \mathcal{P}_i) : (\bar{\mathcal{S}} \setminus \mathcal{P}_i)) \geq y_i \tag{26}$$

where $\mathcal{P}_i = \pi(i) \cup \mathcal{C}_{(i)} \setminus \{i\}$.

Proof. If $y_i = 0$, then the inequality is obviously verified. When $y_i = 1$, vertex i is visited in tour $\mathcal{T}$ representing a feasible solution. Let $\hat{s}$ be the last vertex of $\mathcal{S}$ visited by $\mathcal{T}$. Since $i \in \mathcal{S}$, $\hat{s} \notin \pi(i)$, so $\hat{s} \in \mathcal{S} \setminus \pi(i)$. Moreover, the successor $\hat{t}$ of $\hat{s}$ in tour $\mathcal{T}$ cannot be in $\pi(i)$, so $\hat{t} \in \bar{\mathcal{S}} \setminus \pi(i)$. Clearly, any feasible tour $\mathcal{T}$ contains at least one arc going from $\mathcal{S} \setminus \pi(i)$ to $\bar{\mathcal{S}} \setminus \pi(i)$. The strengthening can be deduced from the observation that only one vertex per cluster is visited in any feasible solution of the GTSPTW. □

Proposition 4.9. *For $\mathcal{S} \subseteq \mathcal{V} \setminus \{0\}$, $j \in \mathcal{S}$ such that $\sigma(j) \neq \emptyset$, the σ-inequalities:*

$$x((\bar{\mathcal{S}} \setminus \sigma(j)) : (\mathcal{S} \setminus \sigma(j))) \geq y_j. \tag{27}$$

are valid for GTSPTW. The σ-inequalities (27) can be strengthened as:

$$x((\bar{\mathcal{S}} \setminus \mathcal{Q}_j) : (\mathcal{S} \setminus \mathcal{Q}_j)) \geq y_j \tag{28}$$

where $\mathcal{Q}_j = \sigma(j) \cup \mathcal{C}_{(j)} \setminus \{j\}$.

Proposition 4.10. *For $\mathcal{S} \subseteq \mathcal{V} \setminus \{0\}$, $i \in \mathcal{S}$, $j \in \bar{\mathcal{S}}$ such that $i \prec j$, $\pi(i) \neq \emptyset$ and $\sigma(j) \neq \emptyset$, the (π, σ)-inequalities:*

$$x((\mathcal{S} \setminus (\pi(i) \cup \sigma(j))) : (\bar{\mathcal{S}} \setminus (\pi(i) \cup \sigma(j)))) \geq y_i + y_j - 1. \tag{29}$$

are valid for GTSPTW. The (π, σ)-inequalities (29) can be strengthened as:

$$x((\mathcal{S} \setminus \mathcal{W}_{ij}) : (\bar{\mathcal{S}} \setminus \mathcal{W}_{ij})) \geq y_i + y_j - 1 \tag{30}$$

where $\mathcal{W}_{ij} = \pi(i) \cup \sigma(j) \cup \mathcal{C}_{(i)} \cup \mathcal{C}_{(j)} \setminus \{i, j\}$.

Proof. If $y_i + y_j \leq 1$, then the inequality is obviously verified. When $y_i = y_j = 1$, vertex i and j are present in tour $\mathcal{T}$ representing a feasible solution. Since $i \prec j, i \in \mathcal{S}, j \in \bar{\mathcal{S}}$, then $i \in \mathcal{S} \setminus (\pi(i) \cup \sigma(j))$ and $j \in \bar{\mathcal{S}} \setminus (\pi(i) \cup \sigma(j))$. Since $i \prec j$, it is obvious that any feasible tour $\mathcal{T}$ contains at least one arc going from $\mathcal{S} \setminus (\pi(i) \cup \sigma(j))$ to $\bar{\mathcal{S}} \setminus (\pi(i) \cup \sigma(j))$. □

4.6 SOP inequalities defined on clusters

In Section 4.5 we presented the SOP inequalities based on the precedence relationship between vertices. We can also define the precedence between a vertex and a cluster.

Let us denote by $i \prec \mathcal{C}_k$ (resp. $\mathcal{C}_k \prec i$) the precedence relation between a vertex and a cluster, i.e., $i \prec \mathcal{C}_k$ if and only if $i \prec j, \forall j \in \mathcal{C}_k$ (resp. $\mathcal{C}_k \prec i$ if and only if $j \prec i, \forall j \in \mathcal{C}_k$). It follows that, if vertex i belongs to a solution, then it has to be visited before (resp. after) any vertex of cluster $\mathcal{C}_k$. Let us indicate by $\mathcal{C}_h \prec \mathcal{C}_k$ the precedence relation between two clusters, i.e., $\mathcal{C}_h \prec \mathcal{C}_k$ if and only i $i \prec j, \forall i \in \mathcal{C}_h, j \in \mathcal{C}_k$. We define $\pi(\mathcal{C}_k) = \{i \in \mathcal{V} \setminus \mathcal{C}_k : i \prec \mathcal{C}_k\}$, $\sigma(\mathcal{C}_k) = \{i \in \mathcal{V} \setminus \mathcal{C}_k : \mathcal{C}_k \prec i\}$. Then we can extend the SOP inequalities as follows.

Proposition 4.11. *Let $\mathcal{S} \subseteq \mathcal{V} \setminus \{0\}$, $\mathcal{C}_k \subseteq \mathcal{S}$, $\pi(\mathcal{C}_k) \neq \emptyset$, the $\pi_{\mathcal{C}_k}$-inequalities:*

$$x((\mathcal{S} \setminus \pi(\mathcal{C}_k)) : (\bar{\mathcal{S}} \setminus \pi(\mathcal{C}_k))) \geq 1. \tag{31}$$

are valid for the GTSPTW.
Let $\mathcal{S} \subseteq \mathcal{V} \setminus \{0\}$, $\mathcal{C}_k \subseteq \mathcal{S}$, $\sigma(\mathcal{C}_k) \neq \emptyset$, the $\sigma_{\mathcal{C}_k}$-inequalities:

$$x((\bar{\mathcal{S}} \setminus \sigma(\mathcal{C}_k)) : (\mathcal{S} \setminus \sigma(\mathcal{C}_k))) \geq 1. \tag{32}$$

are valid for the GTSPTW.
Let $\mathcal{S} \subseteq \mathcal{V} \setminus \{0\}$, $\mathcal{C}_h \prec \mathcal{C}_k$, $\mathcal{C}_h \subseteq \mathcal{S}, \mathcal{C}_k \subseteq \bar{\mathcal{S}}$, $\pi(\mathcal{C}_h) \neq \emptyset, \sigma(\mathcal{C}_k) \neq \emptyset$, the $(\pi_{\mathcal{C}_h}, \sigma_{\mathcal{C}_k})$-inequalities:

$$x((\mathcal{S} \setminus (\pi(\mathcal{C}_h) \cup \sigma(\mathcal{C}_k))) : (\bar{\mathcal{S}} \setminus (\pi(\mathcal{C}_h) \cup \sigma(\mathcal{C}_k)))) \geq 1. \tag{33}$$

are valid for the GTSPTW.

4.7 Clique inequalities

Let $\mathcal{S} \subset \mathcal{V}$ be a subset of vertices belonging to different clusters, i.e., $|\mathcal{S} \cap \mathcal{C}_k| \leq 1, \forall k \in \mathcal{K}$. Such set $\mathcal{S}$ is said *infeasible* if due to the presence of TWs, there exists no feasible path visiting all vertices of $\mathcal{S}$. Then the number of vertices in $\mathcal{S}$ in any feasible solution must be strictly less than $|\mathcal{S}|$ (Padberg, 1973). The clique inequalities can be expressed as follows:

$$\sum_{i \in \mathcal{S}} y_i \leq |\mathcal{S}| - 1. \tag{34}$$

These inequalities can also be lifted.

Proposition 4.12. *For $\mathcal{S} \subset \mathcal{V}$ such that $\mathcal{S}$ is infeasible and $|\mathcal{S} \cap \mathcal{C}_k| \leq 1, \forall k \in \mathcal{K}$. For all $j \in \mathcal{S}$, $\mathcal{S}(j) = \{i \in \mathcal{C}_{(j)} \setminus \{j\} | \mathcal{S}' = (\mathcal{S} \setminus \{j\}) \cup \{i\}$ is infeasible$\}$. If $|\mathcal{S}(j)| \neq 0$, the clique inequality (34) can be strengthened as:*

$$\sum_{i \in \mathcal{S}} y_i + \sum_{i \in \mathcal{S}(j)} y_i \leq |\mathcal{S}| - 1 \qquad \forall j \in \mathcal{S}. \tag{35}$$

For all $h \in \mathcal{S} \setminus \{j\}$, $\mathcal{S}_j(h) = \{i \in \mathcal{C}_{(h)} \setminus \{h\} | (\mathcal{S} \cup \{j^*, i\}) \setminus \{j, h\}$ is infeasible, $\forall j^* \in \mathcal{S}(j) \cup \{j\}\}$. If $|\mathcal{S}_j(h)| \neq 0$, inequality (35) can be lifted as:

$$\sum_{i \in \mathcal{S}} y_i + \sum_{i \in \mathcal{S}(j)} y_i + \sum_{i \in \mathcal{S}_j(h)} y_i \leq |\mathcal{S}| - 1 \qquad \forall j, h \in \mathcal{S}. \tag{36}$$

5 The branch-and-cut algorithm

In this section, we describe the branch-and-cut algorithm we propose to solve the GTSPTW. The algorithm consists of three main phases. The first phase is the preprocessing step that is invoked before starting the optimization procedure. It is presented in Section 5.1. Then we apply a quick heuristic to obtain a feasible solution and to provide an upper bound of the optimal value. Details are given in Section 5.2. Finally, the main phase consists in solving the problem by using a branch-and-cut algorithm, based on the standard branch-and-cut scheme provided by the commercial solver CPLEX 12.6. The initial model is built based on the mixed integer linear programming formulations $\mathcal{F}1$ or $\mathcal{F}2$ with the strengthened MTZ-inequalities proposed in Section 4.1. The initial solution obtained by the heuristic is used as a warm start for the branch-and-cut procedure. Inside the branch-and-bound tree, each time a fractional solution is obtained, valid inequalities proposed in Section 4 are checked, and the ones violated by the current solution are added to the model. Details are given in Section 5.3. For valid inequalities with a polynomial number of constraints, we memorize all of them and scan the entire set to seek for those that are violated (see Section 5.3.3). For exponential-size families of constraints (GSEC and SOP inequalities), separation algorithms are applied to efficiently detect the violated inequalities (see Sections 5.3.1 and 5.3.2), and the number of the inequalities we choose to separate is limited (see Section 5.3.4).

5.1 Data preprocessing

As with many other combinatorial optimization problems, preprocessing is an important feature to enhance the resolution of the problem. In our case, preprocessing step consists in tightening the TWs and eliminating arcs that cannot be part of a feasible solution.

The TW width can be reduced by taking into account the earliest and the latest arrival and departure times at each vertex of the graph from or to another vertex. In particular, we consider the following conditions proposed by Desrochers et al. (1992):

- earliest arrival time from predecessors: $E_i = \max\{E_i, \min\{L_i, \min_{(j,i)\in\mathcal{A}}(E_j + T_{ji})\}\}$;
- earliest departure time to successors: $E_i = \max\{E_i, \min\{L_i, \min_{(i,j)\in\mathcal{A}}(E_j - T_{ij})\}\}$;
- latest arrival time from predecessors: $L_i = \min\{L_i, \max\{E_i, \max_{(j,i)\in\mathcal{A}}(L_i + T_{ji})\}\}$;
- latest departure time to successors: $L_i = \min\{L_i, \max\{E_i, \max_{(i,j)\in\mathcal{A}}(L_j - T_{ij})\}\}$.

These conditions are applied iteratively to all vertices until no TW can be reduced. After applying the TW reduction, we sparsify the graph by eliminating the arcs that cannot be part of a feasible solution. In particular, we remove:

- arcs $(i,j) \in \mathcal{A}$ such that $E_i + T_{ij} > L_j$;
- arcs $(i,j) \in \mathcal{A}, i,j \neq 0$ such that $\exists\, k \in \mathcal{K}, \forall h \in \mathcal{C}_k$, both $E_h + SP_{hi} + T_{ij} > L_j$ and $E_i + T_{ij} + SP_{jh} > L_h$ hold, i.e., arc (i,j) can be traversed neither before nor after visiting one cluster $\mathcal{C}_k$.

This results in the elimination of the corresponding x variables in the formulation.

5.2 Initial heuristic

We develop a simple and fast heuristic to identify a feasible solution for the GTSPTW. The solution obtained is used as a warm start in the branch-and-cut procedure.

First, we extend the refinement procedure for GTSP proposed by Fischetti et al. (1997) to GTSPTW case. Suppose we have a visiting sequence $(h_1, \ldots, h_p)$ of p different clusters in $\mathcal{K} \setminus \{0\}$. Based on this sequence we construct a layered network (LN) as depicted in Figure 4. This network has $p+2$ layers corresponding to clusters $C_{h_0} = \mathcal{C}_0, \mathcal{C}_{h_1}, \ldots, \mathcal{C}_{h_p}, \mathcal{C}_{h_{p+1}} = \mathcal{C}_0$, with their respective vertices. Clusters $\mathcal{C}_{h_0}$ and $\mathcal{C}_{h_{p+1}}$ both represent the depot. The LN contains arcs $(i,j) \in \mathcal{A}$ such that $i \in \mathcal{C}_{h_f}, j \in \mathcal{C}_{h_{f+1}}, f = 0, \ldots, p$. The objective is to find a path in the LN that starts at $\mathcal{C}_{h_0}$ and ends at $\mathcal{C}_{h_{p+1}}$ visiting exactly one vertex of each layer, that is, one vertex from each cluster. The solution can be found by determining the shortest path with TWs from $\mathcal{C}_{h_0}$ to $\mathcal{C}_{h_{p+1}}$. If $p = K$, the resulting path (if it exists) provides a feasible GTSPTW solution.

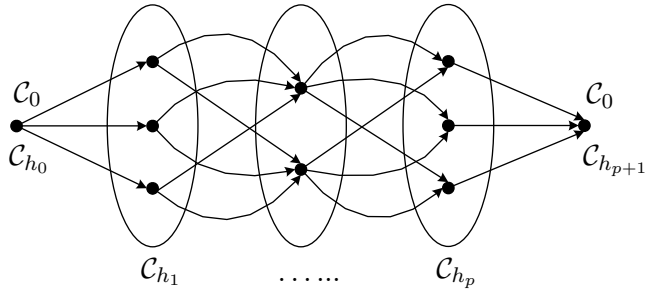


Figure 4: The layered network.

A labeling algorithm is applied to determine the shortest path with TWs on the LN. A label L_i associated with a vertex i consists of a pair (C_i, T_i) representing respectively the cost and service time of a feasible partial path that starts at $\mathcal{C}_{h_0}$ and arrives at vertex i. Let $\mathcal{L}(i)$ be the set containing all the labels associated with vertex i. Suppose that $\mathcal{C}_{cur}$ is the current cluster and $\mathcal{C}_{pre}$ is the previous one. First we compute the label set $\mathcal{L}(i)$, for all $i \in \mathcal{C}_{cur}$ by extending labels in $\mathcal{L}(j)$, for all $j \in \mathcal{C}_{pre}$. Extending a label $L_j \in \mathcal{L}(j)$ towards a vertex $i \in \mathcal{C}_{cur}$ consists in creating another label $L_i \in \mathcal{L}(i)$ such that:

$$C_i = C_j + C_{ij}; \tag{37}$$

$$T_i = \max\{E_i, T_j + T_{ji}\}. \tag{38}$$

If $T_i > L_i$, the partial path associated with the label is infeasible and this label is disregarded.

To make the algorithm efficient, we only keep non-dominated labels. A label L_i^1 dominates a label L_i^2 if and only if $C_i^1 \leq C_i^2$ and $T_i^1 \leq T_i^2$. It is easy to see that extending L_i^1 on the same arcs to the last vertex of the LN would always produce a better solution than extending L_i^2 in the same way.

To generate good sequences of clusters, we develop the following constructive procedure. The initial sequence of clusters is empty, hence the corresponding LN contains two layers: $\mathcal{C}_{h_0} = \mathcal{C}_{h_1} = \mathcal{C}_0$. Then, at each step, we randomly select a cluster from those that are not yet inserted into the sequence. Suppose that the current sequence is $(h_1, \ldots, h_p)$, and cluster $\mathcal{C}_{h_l}$ is chosen to be inserted next. It is obvious that there are $p+1$ possible insertion positions for index h_l into the sequence. The labeling algorithm described

above is invoked $p+1$ times, one for each candidate insertion, to determine the best insertion position. We record the sequence that provides the shortest path with TWs if such a sequence exists. If this is not the case, the sequence construction procedure is stopped. The cluster insertion procedure is repeated until $p = K$ to obtain a feasible solution for GTSPTW.

The sequence construction procedure is repeated R times, and the best solution is recorded. After preliminary experiments, we set $R = 2500$. A general description of the initial heuristic is provided in Algorithm 1. Note that when $K < 7$, there are only 720 possible cluster sequences. It is more efficient to enumerate all the sequences and to compute the shortest path with TWs for all of them. Since the labeling procedure is an exact procedure (given a sequence it finds the optimal path for that sequence), the optimal solution for the instance is found without calling the Branch-and-cut procedure. Therefore, the branch-and-cut algorithm is only applied for instances with $K \geq 7$ when applying the labeling procedure on all the sequences of size K becomes ineffective.

Algorithm 1 Heuristic to provide an initial solution.

$R \leftarrow 2500$
bestSol: best solution found
for $h = 1$ to $h = R$ **do**
    $\mathcal{H} \leftarrow \mathcal{K} \setminus \{0\}$ (the set of cluster index)
    $feas \leftarrow$ true
    $S \leftarrow \emptyset$ (the sequence of clusters)
    while $feas =$ true **and** $\mathcal{H}$ is not empty **do**
        $l \leftarrow$ an index randomly chosen in set $\mathcal{H}$
        Remove l from $\mathcal{H}$
        Try to insert l at its best position in sequence S
        if the insertion of l is not feasible **then**
            $feas \leftarrow$ false
    if $feas =$ true **then**
        Update *bestSol* if S provides a better solution than the current *bestSol*
return *bestSol*

5.3 Separation techniques

At each node of the branch-and-cut tree, a relaxation of the model $\mathcal{F}1$ or $\mathcal{F}2$ is solved. Let us denote by $R\mathcal{F}1$ and $R\mathcal{F}2$ these relaxations. When solving a relaxed model, the binary requirement on the variables x and y is relaxed. Therefore, a fractional solution can be obtained where some values of the x or y variables are in the interval $]0, 1[$. Then, a separation algorithm is used to detect violated inequalities Since $\mathcal{F}1$ and $\mathcal{F}2$ are compact formulations of the GTSPTW, the valid inequalities proposed in Section 4 are not needed to define the problem but can help to strengthen the relaxed model. In the subsequent sections, we describe the separation algorithms that we implemented for the different families of valid inequalities. We indicate by (x^*, y^*) the current fractional solution.

5.3.1 Separation of the GSEC inequalities

It is well known that the separation of the SEC for the ATSP can be done by computing the maximum flow between the depot and each node j in the support graph $\mathcal{G}^*$ which corresponds to the undirected version of the original graph $\mathcal{G}$ where the capacity c_e of edge $e = \{i, j\}$ is equal to $x^*_{ij} + x^*_{ji}$. If each maximum flow is greater or equal than 2, the associated SEC is not violated by x^*. Otherwise, the minimum $(0 - j)$ cut induces a violated SEC inequality (Nemhauser and Wolsey, 1999).

To separate the GSEC (22), we consider a capacitated graph $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$ where $\mathcal{V}^* = \mathcal{K}^* = \{0, 1, \ldots, K\}$, and $\mathcal{E}^* = \{\{k, l\} | k, l \in \mathcal{V}^*, k \neq l\}$. A capacity c_e is associated with each edge $e = \{k, l\} \in \mathcal{E}^*$, and defined as $c_e = \sum_{i \in \mathcal{C}_k, j \in \mathcal{C}_l} (x^*_{ij} + x^*_{ji})$. If the maximum flows in this graph $\mathcal{G}^*$ have a value lower than 2, then some of the GSEC (22) are violated, and the corresponding constraints are added into the model. To compute the maximum flow in $\mathcal{G}^*$ and detect all violated inequalities, the Gomory-Hu algorithm is applied, with a $\mathcal{O}(K^4)$ time complexity (Gomory and Hu, 1961). By using Gomory-Hu algorithm, we obtain the maximum flow and the corresponding minimum cut between each pair of distinct vertices in $\mathcal{V}^*$. Thus, all the minimum cuts in $\mathcal{G}^*$ with a value lower than 2 are detected, and the related GSEC (22) are added into the model.

5.3.2 Separation of the SOP inequalities

π-inequalities

We adapt the procedure used by Balas et al. (1995) to separate this family of inequalities. For any vertex i with $\pi(i) \neq \emptyset$ and $y^*_i > 0$, we consider a graph $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{A}^*)$, such that $\mathcal{V}^* = \mathcal{V} \setminus \{\pi(i) \cup \{\mathcal{C}_{(i)} \setminus \{i\}\}\}$ and $\mathcal{A}^* = \{(i, j) | i, j \in \mathcal{V}^*, x^*_{ij} > 0\}$. We associate with each arc in $\mathcal{A}^*$ a capacity equal to the corresponding x^* values and we compute the maximum flow from vertex i to the depot 0 in $\mathcal{G}^*$. If this flow is less than y_i, then the minimum $(i, 0)$ cut identifies a violated π-inequality.

σ-inequalities

To detect violated σ-inequalities we apply a procedure similar to the one described for the π-inequalities except that $\mathcal{V}^* = \mathcal{V} \setminus \{\sigma(i) \cup \{\mathcal{C}_{(i)} \setminus \{i\}\}\}$ and we compute the maximum from the depot 0 to vertex i in the graph $\mathcal{G}^*$. If this flow is lower than y_i, the corresponding minimum cut identifies a violated σ-inequality.

(π, σ)-inequalities

To detect violated (π, σ)-inequalities we consider each pair of vertices i, j such that $i \prec j, \pi(i) \neq \emptyset, \sigma(j) \neq \emptyset$ and $y^*_i + y^*_j - 1 > 0$. We then apply a procedure similar to the one described for the π-inequalities except that $\mathcal{V}^* = \mathcal{V} \setminus \mathcal{W}_{ij}$ where $\mathcal{W}_{ij} = \{0\} \bigcup \pi(i) \bigcup \sigma(j) \bigcup \mathcal{C}_{(i)} \bigcup \mathcal{C}_{(j)} \setminus \{i, j\}$ and we compute the maximum from vertex i to vertex j in the graph $\mathcal{G}^*$. If this flow is lower than $(y_i + y_j - 1)$, the corresponding minimum cut identifies a violated (π, σ)-inequality.

Note that the graph $\mathcal{G}^*$ can be sparsified by deleting: i) all vertices k such that path (i, k, j) is infeasible, i.e., $E_i + SP_{ik} + SP_{kj} > L_j$; ii) all arcs (u, v) such that path (i, u, v, j) is infeasible, e.g., $E_i + SP_{iu} + T_{uv} + SP_{vj} > L_j$.

$\pi_{\mathcal{C}_k}$-inequalities

To separate $\pi_{\mathcal{C}_k}$-inequalities, we consider any cluster $\mathcal{C}_k$ such that $\pi(\mathcal{C}_k) \neq \emptyset, |\mathcal{C}_k| > 1$. $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{A}^*)$ is such that $\mathcal{V}^* = \{s_k\} \cup \mathcal{V} \setminus \pi(\mathcal{C}_k)$ where s_k is an additional vertex and $\mathcal{A}^* = \mathcal{A}_1^* \cup \mathcal{A}_2^* = \{(i,j)|i,j \in \mathcal{V}^*, x_{ij}^* > 0\} \cup \{(s_k, j)|j \in \mathcal{C}_k\}$. We associate with each arc in $\mathcal{A}_1^*$ a capacity equal to the corresponding x^* value and with each arc in $\mathcal{A}_2^*$ a very large capacity. In the resulting graph $\mathcal{G}^*$, we compute the maximum flow from vertex s_k to the depot 0. If it is strictly less than 1, a violated $\pi_{\mathcal{C}_k}$ inequality is identified.

$\sigma_{\mathcal{C}_k}$-inequalities

These inequalities are detected by adapting the explained procedure to identify violated $\pi_{\mathcal{C}_k}$-inequalities.

$(\pi_{\mathcal{C}_k}, \sigma_{\mathcal{C}_k})$-inequalities

To detect violated $(\pi_{\mathcal{C}_h}, \sigma_{\mathcal{C}_k})$-inequalities we consider each pair of clusters $\mathcal{C}_h, \mathcal{C}_k$ such that $\mathcal{C}_h \prec \mathcal{C}_k$, $|\mathcal{C}_h| > 1, |\mathcal{C}_k| > 1$. $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{A}^*)$ is such that $\mathcal{V}^* = \{s_h, s_k\} \cup \mathcal{V} \setminus \{\pi(\mathcal{C}_h) \cup \sigma(\mathcal{C}_k)\}$ where s_h and s_k are two additional vertices and $\mathcal{A}^* = \mathcal{A}_1^* \cup \mathcal{A}_2^* \cup \mathcal{A}_3^* = \{(i,j)|i,j \in \mathcal{V}^*, x_{ij}^* > 0\} \cup \{(s_h, j)|j \in \mathcal{C}_h\} \cup \{(j, s_k)|j \in \mathcal{C}_k\}$. We associate with each arc in $\mathcal{A}_1^*$ a capacity equal to the corresponding x^* value and with each arc in $\mathcal{A}_2^*$ and $\mathcal{A}_3^*$ a very large capacity. In the resulting graph $\mathcal{G}^*$, we compute the maximum flow from vertex s_h to vertex s_k. If it is strictly less than 1, a violated $(\pi_{\mathcal{C}_h}, \sigma_{\mathcal{C}_k})$-inequality is identified.

5.3.3 Separation of the arc selection inequalities, arc-or-vertex inequalities and clique inequalities

Arc orientation inequalities (18), arc-or-vertex inequalities (19) and their lifted versions (20) and (21) are polynomial in the size of the input. The clique inequalities (34) and their lifted versions (35) and (36), we consider, are restricted to $\mathcal{S} \subset \mathcal{V}$, $|\mathcal{S} \cap \mathcal{C}_k| \leq 1$ for all $k \in \mathcal{K}$ with $|\mathcal{S}| = 2, 3$. Therefore, whenever a fractional solution is obtained, we scan the entire set of these inequalities to seek for those that are violated.

5.3.4 Separation strategy

During the branch-and-cut procedure, at each time a fractional solution is obtained, the separation procedures for GSEC, SOP, clique, arc orientation and arc-or-vertex inequalities is called. GSEC (22) are separated using Gomory-Hu algorithm. One call to the algorithm provides all the violated cuts. However, SOP cuts and SOP cuts on clusters require repeated calls to maximum flow algorithm: $\mathcal{O}(N)$ times for π and σ-inequalities, $\mathcal{O}(N^2)$ times for (π, σ)-inequalities, $\mathcal{O}(K)$ times for $\pi_{\mathcal{C}_k}$ and $\sigma_{\mathcal{C}_k}$-inequalities, and $\mathcal{O}(K^2)$ times for $(\pi_{\mathcal{C}_h}, \sigma_{\mathcal{C}_k})$-inequalities.

Solving these maximum flow problems would be time-consuming. Therefore, we introduce a parameter α to control the percentage of SOP inequalities that we choose to separate. For each class of SOP inequalities, we first determine the eligible vertices or clusters (for example for π-inequalities, all the vertices i such that $\pi(i) \neq \emptyset$ and $y_i^* > 0$), and then α percent are randomly chosen to be separated.

In addition, to improve the influence of the cuts added to the relaxed model, they should be significantly violated. Thus, we introduce a parameter ε to control the lowest violation of the cuts we add. For each family of inequalities, after the call to the separation algorithm, only the cuts having a violation of at least ε are added into the model.

Detailed results on the setting of these parameters α and ε are presented in section 6.2.

6 Computational experiments

The algorithms were implemented in C++ in Visual Studio environment and uses CPLEX 12.6 and the Concert framework. Experiments were performed on a PC Intel(R) Core(TM) i5-6200U CPU 2.20GHz and 64G RAM. The computation time limit (TL) was set to 3600 seconds.

6.1 Problem instances

Since no testbed is available for GTSPTW in the literature, we created three groups of instances to test the proposed algorithm.

The first group indicated by *G1* includes 47 instances. *G1* is generated by performing suitable modifications to the existing benchmark for the GTSP proposed by Karapetyan (2012). The GTSP does not take time into account, so for each arc of the graph, the traveling time is set equal to the traveling cost. The GTSP instances do not necessarily contain a cluster with a unique vertex that could be the depot. Hence, based on a GTSP instance, a depot is added. As coordinates are not always available in GTSP instances, the traveling time from the depot to other vertices is fixed to be 0. The TW of the depot is $[0, T]$, where T initially equals twice the best objective value of the original GTSP instance. Then, the TW $[E_i, L_i]$ for each vertex $i \in \mathcal{V} \setminus \{0\}$ is generated according to the method described by Solomon (1987). The center of the TW, denoted as c_i, is randomly generated from a uniform distribution in the interval $[0, T]$. For the width w_i of the TW, a number r_i is randomly generated from the standard normal distribution, then $w_i = |r_i| \min\{c_i, T - c_i\}$. Thus the TW $[E_i, L_i]$ can be obtained as $E_i = c_i - w_i, L_i = c_i + w_i$. Once the above procedure has been applied, we impose a modification procedure to ensure that a feasible solution exists for the instance created. First, a sequence of clusters is randomly generated. Then, for each cluster, one vertex is randomly selected in order to get a tour. Starting at time 0 from the depot, we compute the service time at each vertex from its previous vertex in this tour. If the service time at vertex i exceeds the upper bound L_i, L_i is then updated with the service time value. At the end of the tour, if the arrival time at the depot is greater than T, then T is updated to the arrival time value. In this way, we ensure that a feasible solution exists for the instance generated. The instance name is obtained by adding TW_ before its original GTSP name, e.g., we create TW_3burma14 based on 3burma14. The number in the middle of the name represents the number of clusters, and the number at the end represents the number of vertices, both excluding the depot.

In GTSP instances, some clusters contain a large number of vertices. This does not correspond to the last mile delivery application. Therefore, we generated a second group indicated by *G2* which includes 39 instances. *G2* is obtained in a similar way to the set *G1*, except that the maximum number of vertices per cluster $N_{\max}$ is fixed. In our experiments, we set $N_{\max} = 5$ which is reasonable in the case of last mile

delivery. Based on a GTSP instance, when the number of vertices in a cluster $\mathcal{C}_k$ is greater than $N_{\max}$, we divide the corresponding cluster into $M^{'}$ clusters, where $M^{'} = \lceil |\mathcal{C}_k|/N_{\max} \rceil$. The first $N_{\max}$ vertices in $\mathcal{C}_k$ constitute the first cluster, and so on. The following steps are as described above to create instances in *G1*. The instance name is similar to *G1*, while the number in the middle of the name changes to the number of clusters in the instance excluding the depot, e.g., TW_4burma14 is created based on 3burma14.

The third group with 72 instances, indicated by *G3* is obtained from the instances proposed by Reyes et al. (2017) for the VRPRDL. As mentioned above, the TWs in these instances have a particular structure, i.e., the TWs associated with vertices in a same cluster do not overlap. Since the VRPRDL instances are for the multi-vehicle case, we use only part of the instance information. Given one VRPRDL instance, we consider the best corresponding solution, and define an instance for the GTSPTW by the clusters visited along the longest route in the solution. The data about the distances, the traveling times and TWs are not modified.

To create larger instances, we combine several routes belonging to a solution of one VRPRDL instance. In this case, TWs need to be modified to obtain a set of customers that can be visited in a single route. Notice that the vertices within the same cluster are ordered based on the earliest visit times. An example is given in Figure 5 for the case with two routes. Suppose that $\mathcal{S}_1$ and $\mathcal{S}_2$ are the cluster sets which include all the clusters visited in the routes r_1 and r_2, respectively. We note $(\mathcal{C}_{r_1,1}, \mathcal{C}_{r_1,2}, \ldots, \mathcal{C}_{r_1,|\mathcal{S}_1|})$ the sequence of clusters visited by route r_1, and $(i_{r_1,1}, i_{r_1,2}, \ldots, i_{r_1,|\mathcal{S}_1|})$ the sequence of vertices visited in route r_1. Obviously, $i_{r_1,k} \in \mathcal{C}_{r_1,k}$, $\forall k = 1, \ldots, |\mathcal{S}_1|$. The same notation is used for route r_2. The time horizon of the original VRPRDL instances lasts $\Delta = 12$ hours. We then move forward the TWs of all the vertices belonging to the clusters in $\mathcal{S}_2$ by Δ plus the traveling time from $i_{r_1,|\mathcal{S}_1|}$ the last visited vertex in r_1 to $i_{r_2,1}$ the first visited vertex in r_2. Then for each cluster $\mathcal{C}_{r_1,k}$ in $\mathcal{S}_1$, the TWs of vertices which are before $i_{r_1,k}$ the vertex visited in this cluster by r_1 are moved forward by $\Delta + \delta(\mathcal{C}_{r_1,k})$. $\delta(\mathcal{C}_{r_1,k})$ corresponds to the traveling time from the last vertex of $\mathcal{C}_{r_1,k}$ to the first vertex of $\mathcal{C}_{r_1,k}$, where vertices are ordered based on the earliest visit times. In the same way, for each cluster $\mathcal{C}_{r_2,k}$ in $\mathcal{S}_2$, the TWs of vertices which are after $i_{r_2,k}$ the vertex visited in this cluster by r_2 are moved backward by $\Delta + \delta(\mathcal{C}_{r_2,k})$. In this way, we inherit the property of non-overlapping TWs within a cluster and ensure that there is a feasible solution $r = [r_1, r_2]$ in the created instance. Similarly, we can use more than two routes of a solution to create instances.

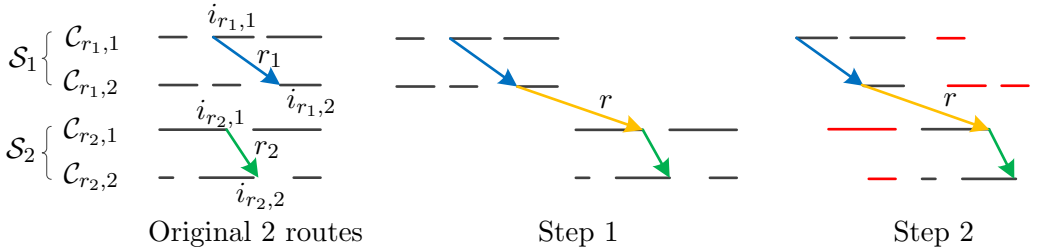


Figure 5: Create *G3* instances from 2 routes.

We use sequences *a-b-c-d* to name instances in *G3*, where *a* and *b* indicate the number of clusters and vertices in the instance respectively, similar to *G1* and *G2*, excluding the depot. *c* and *d* indicate

that the instance is created based on the dth VRPRDL instance using its c longest routes in the solution, e.g., 14-41-2-6 means that using the two longest routes in the solution of the sixth VRPRDL instance, we create a *G3* instance with 14 clusters and 41 vertices.

6.2 Parameter tuning results

In the separation procedure described in Section 5.3.4 we introduced two parameters, α and ε, which respectively control the percentage of SOP inequalities that we randomly choose to separate and the least violation of all the cuts that we add. We limit the impact of α to the SOP inequalities due to their large cardinality. Here we discuss the tuning phase of α and ε.

To observe significant differences between results obtained from different parameter settings, we choose instances of comparatively medium and large size in sets *G1* and *G2*. We do not consider instances of *G3* since preliminary results showed that they are the easiest to solve.

We choose 7 instances: 4 in *G1* (TW_20kroA100, TW_25pr124, TW_26ch130, TW_35si175) and 3 in *G2* (TW_28rat99, TW_30eil101, TW_32gr120). We consider values of α in $\{40; 60; 80; 100\}$ and values of ε in $\{0.05; 0.1; 0.15\}$ and perform experiments for all possible pairs (α, ε). The average optimality gaps for the two formulations $\mathcal{F}1$ and $\mathcal{F}2$ are reported in Table 1. Computation time is limited to one hour per execution.

Table 1: Tuning results on $\mathcal{F}1$ and $\mathcal{F}2$.

ϵ	α	Average gap $\mathcal{F}1$	Average gap $\mathcal{F}2$
0.05	40	1.54%	0.38%
	60	2.06%	1.37%
	80	2.65%	1.10%
	100	2.13%	1.23%
0.1	40	1.79%	0.65%
	60	1.77%	0.82%
	80	2.22%	0.48%
	100	2.21%	0.97%
0.15	40	1.69%	***0.13%***
	60	1.88%	1.02%
	80	2.14%	0.89%
	100	2.19%	0.37%

Our results in Table 1 show that the average optimality gaps obtained from $\mathcal{F}2$ are always smaller than those obtained from $\mathcal{F}1$. This indicates that formulation $\mathcal{F}2$ turns to be superior to formulation $\mathcal{F}1$. It is also clear that the best parameter setting for $\mathcal{F}2$ is $(\alpha, \epsilon) = (40, 0.15)$ with the smallest average gap of 0.13%. Therefore, we conduct the following experiments using formulation $\mathcal{F}2$ with $(\alpha, \epsilon) = (40, 0.15)$.

6.3 Computational results

We now present the results obtained with the branch-and-cut algorithm presented in this paper. The algorithm was tested on the 158 instances in *G1*, *G2* and *G3* and, due to the results presented in Section 6.2, experiments were carried out with formulation $\mathcal{F}2$. Table 2 provides the column headings

used in the following tables. Detailed results on *G1*, *G2* and *G3* are presented in Table 3, 4 and 5 respectively.

Table 2: Column headings.

Column heading	Description
Instance	name of the problem instance
Obj	value of the best solution obtained
initS	value of initial heuristic solution
rLB	lower bound at the root node
fLB	lower bound at the termination/at the time limit
rGAP	gap at the root node (%): rGAP = $\frac{\text{rUB}-\text{rLB}}{\text{rUB}}$ where rUB is the upper bound at the root node
fGAP	gap at the termination/at the time limit (%): fGAP = $\frac{\text{Obj}-\text{fLB}}{\text{Obj}}$
nbNode	number of nodes in the branch-and-cut tree
nbCuts	number of generated cuts
sep-time	time spent in separation procedure in seconds
time	total computation time in seconds (equals to 1 hour if optimality is not achieved)

As mentioned in Section 5.2, the labeling algorithm can be executed on all sequences with less than seven clusters within reasonable computation times. In this case, instead of searching for an initial solution, we just enumerate all feasible sequences to get an optimal solution without calling the branch-and-cut algorithm. For this reason, we report only the objective values and computation times for the first ten lines of Table 3 and the first four lines of Table 4.

In Table 3, we report results on instances in *G1*. Our results indicate that our algorithm is able to solve most instances up to 30 clusters and 150 vertices within one hour. However, one instance with 26 clusters (TW_26bier127), one instance with 28 clusters (TW_28pr136) and one instance with 30 (TW_30kroB150) clusters remain unsolved. Moreover, all instances with up to 24 clusters and 124 vertices are systematically solved in less than three minutes. For all the instances solved to optimality, we calculate the average gap between their initial solution obtained from the heuristic $initS$ and the final optimal solution Obj, which equals to 0.59%. This proves the efficiency of the heuristic.

When we consider instances in *G2* (see Table 4), we can observe that the largest solved instance within the given time frame involves 36 clusters and 130 vertices. Instances with up to 28 clusters and 100 vertices are systematically solved to optimality in one hour of computation time. Initial heuristic is also very efficient on instance set *G2*. For all the instances solved to optimality, the average gap between their initial solution obtained from the heuristic $initS$ and the final optimal solution Obj equals to 0.10%.

Table 5 reports results on Instances in *G3* (see Table 5). All the instances are solved to optimality in very short computation times. Most instances are solved at the root node. The largest instance with 32 clusters and 125 vertices is solved to optimality in less than one minute. For most of the instances, the initial solutions obtained from the heuristic turn out to be the optimal solutions, except instance 16-51-2-38 with a gap 0.47% and instance 28-99-3-31 with a gap 0.71% from their optimal solutions. From the results, we can conclude that the special non-overlapping TW structure that characterizes vertices in clusters of instances in *G3* makes the problem easier to solve.

The computational results on the three groups of instances show that the proposed branch-and-cut

Table 3: Results on *G1* ($\mathcal{F}2$).

Instance	Obj	initS	rGAP(%)	rLB	fGAP(%)	fLB	nbNodes	nbCuts	sep-time	time
TW_3burma14	908	908								0.01
TW_4br17	19	19								0.01
TW_4gr17	962	962								0.01
TW_4ulysses16	2392	2392								0.01
TW_5gr21	1165	1165								0.01
TW_5gr24	263	263								0.01
TW_5ulysses22	3287	3287								0.01
TW_6bayg29	476	476								0.02
TW_6bays29	628	628								0.02
TW_6fri26	354	354								0.03
TW_7ftv33	416	416	0.00	416.0	0.00	416.0	0	41	0.01	1.41
TW_8ftv36	538	554	6.11	520.2	0.00	538.0	5	207	0.19	2.70
TW_8ftv38	384	410	0.00	384.0	0.00	384.0	0	124	0.02	1.89
TW_9dantzig42	322	322	6.51	301.0	0.00	322.0	85	531	1.45	5.67
TW_10att48	4113	4113	0.00	4113.0	0.00	4113.0	0	95	0.04	5.31
TW_10gr48	1437	1515	2.64	1429.3	0.00	1437.0	6	164	0.26	6.13
TW_10hk48	5268	5268	4.34	5039.6	0.00	5268.0	49	560	0.70	5.90
TW_11berlin52	3632	3632	9.74	3278.4	0.00	3632.0	438	836	5.40	13.38
TW_11eil51	151	157	0.00	151.0	0.00	151.0	0	253	0.50	6.22
TW_12brazil58	13503	13503	2.82	13122.7	0.00	13503.0	18	416	0.47	7.96
TW_14st70	289	289	5.57	272.9	0.00	289.0	1326	1479	48.39	77.62
TW_16eil76	205	205	4.33	196.1	0.00	205.0	79	1931	4.38	24.26
TW_16pr76	57164	57164	0.84	56683.6	0.00	57164.0	19	1018	2.71	19.85
TW_20gr96	33128	33128	3.52	31960.9	0.00	33128.0	68	1538	22.17	65.98
TW_20kroA100	10209	10209	7.76	9417.2	0.00	10209.0	120	3525	56.17	131.33
TW_20kroB100	9862	9975	4.05	9570.9	0.00	9862.0	62	3131	24.01	76.71
TW_20kroC100	9728	9728	3.38	9399.1	0.00	9728.0	246	2944	29.43	76.21
TW_20kroD100	9210	9210	3.87	8854.0	0.00	9210.0	70	1494	22.92	66.75
TW_20kroE100	9514	9514	3.58	9173.5	0.00	9514.0	121	2761	13.85	55.17
TW_20rat99	478	478	6.11	448.8	0.00	478.0	206	3871	27.04	81.86
TW_20rd100	3446	3505	2.58	3357.0	0.00	3446.0	142	2535	16.50	61.51
TW_21eil101	247	247	5.59	233.2	0.00	247.0	144	3334	28.21	87.96
TW_21lin105	7582	7582	2.15	7419.1	0.00	7582.0	90	2740	25.26	77.53
TW_22pr107	21352	21481	1.27	21208.1	0.00	21352.0	9	1272	9.39	53.41
TW_24gr120	2811	2825	5.18	2678.6	0.00	2811.0	123	3384	51.69	157.79
TW_25pr124	36520	36520	7.29	33859.3	0.00	36520.0	6241	12716	875.16	2360.25
TW_26bier127	86761	86761	17.44	71627.8	14.86	73867.5	4716	12654	990.31	TL
TW_26ch130	2891	2896	5.17	2746.4	0.00	2891.0	4405	10393	903.71	1785.17
TW_28gr137	34123	34201	4.61	32625.9	0.00	34123.0	1777	6217	270.17	554.41
TW_28pr136	45998	45998	7.50	42549.9	1.63	45246.6	6124	16707	1788.74	TL
TW_29pr144	43639	43639	1.13	43143.7	0.00	43639.0	188	4910	174.72	356.05
TW_30ch150	2895	2895	6.29	2712.8	2.02	2836.6	4842	14537	1980.22	TL
TW_30kroA150	11475	11475	4.18	10995.1	0.00	11475.0	384	10779	201.97	495.47
TW_30kroB150	12759	12869	6.89	11981.7	2.52	12437.8	2129	15662	1728.91	TL
TW_31pr152	55128	55128	17.45	45506.3	15.80	46419.8	8600	15081	1548.08	TL
TW_32u159	22846	22846	6.99	21249.5	2.29	22323.0	2124	12451	1747.28	TL
TW_35si175	5521	5561	2.42	5426.7	0.91	5470.8	1180	13828	1865.05	TL

algorithm is very efficient and is able to solve instances with 30 clusters. This represents the maximal number of deliveries that a courier can perform during a day in an urban context. Moreover, the initial solutions provided by the heuristic are of high quality with respect to the best solutions obtained.

Table 4: Results on *G2* ($\mathcal{F}2$).

Instance	Obj	initS	rGAP(%)	rLB	fGAP(%)	fLB	nbNodes	nbCuts	sep-time	time
TW_4burma14	957	957								0.01
TW_5br17	19	19								0.01
TW_5gr17	766	766								0.01
TW_6ulysses16	2521	2521								0.02
TW_7gr21	1475	1475	0.00	1475.0	0.00	1475.0	0	55	0.02	0.88
TW_7gr24	365	365	0.00	365.0	0.00	365.0	0	60	0.01	0.93
TW_8ulysses22	3461	3461	0.00	3461.0	0.00	3461.0	0	78	0.01	1.15
TW_8bayg29	515	515	0.00	515.0	0.00	515.0	0	56	0.00	1.31
TW_8bays29	595	595	0.00	595.0	0.00	595.0	0	47	0.02	1.53
TW_9fri26	498	498	0.00	498.0	0.00	498.0	0	62	0.00	1.34
TW_10ftv33	467	467	0.00	467.0	0.00	467.0	0	113	0.06	2.33
TW_10ftv36	500	500	2.61	487.0	0.00	500.0	3	148	0.17	2.95
TW_10ftv38	469	469	8.50	429.1	0.00	469.0	48	490	0.5	4.05
TW_12dantzig42	364	364	0.00	364.0	0.00	364.0	0	204	0.41	4.66
TW_13gr48	2010	2010	1.77	1974.4	0.00	2010.0	7	611	0.25	5.23
TW_14att48	5415	5415	7.13	5028.7	0.00	5415.0	383	951	5.75	16.08
TW_14hk48	5989	5989	6.40	5605.9	0.00	5989.0	175	855	2.69	10.37
TW_15eil51	172	172	3.22	166.5	0.00	172.0	5	367	0.73	9.34
TW_16berlin52	3821	3821	3.88	3672.6	0.00	3821.0	253	914	3.97	15.45
TW_17brazil58	12665	12665	4.35	12114.5	0.00	12665.0	54	896	4.05	18.39
TW_20st70	329	329	5.21	311.9	0.00	329.0	255	1605	12.01	35.47
TW_21eil76	245	245	9.74	221.1	0.00	245.0	6718	3629	266.32	463.47
TW_22pr76	71098	71098	6.85	66229.7	0.00	71098.0	202	1969	28.09	67.67
TW_24kroC100	9932	9932	2.35	9698.9	0.00	9932.0	25	1404	13.23	68.43
TW_26kroA100	10515	10541	3.66	10130.6	0.00	10515.0	74	2569	25.33	88.10
TW_28rat99	618	628	10.91	559.5	0.00	618.0	10014	10811	1789.50	3418.14
TW_28kroB100	11937	11976	6.19	11234.7	0.00	11937.0	1245	4579	152.68	282.56
TW_28kroD100	10720	10720	8.71	9785.8	0.00	10720.0	347	3642	116.67	240.92
TW_29gr96	35064	35064	7.64	32383.6	0.00	35064.0	1698	5145	323.82	558.95
TW_29kroE100	11275	11275	5.97	10602.0	0.00	11275.0	16295	5457	726.72	1130.78
TW_29rd100	4380	4380	9.65	3957.5	0.85	4342.7	12627	9801	1585.25	TL
TW_30eil101	292	292	6.24	273.8	0.00	292.0	638	4078	142.63	272.15
TW_32lin105	9944	9965	10.36	8932.6	3.98	9548.7	9584	10072	2056.20	TL
TW_32gr120	3307	3307	9.34	2998.0	0.00	3307.0	9010	12733	1108.94	2788.73
TW_32pr124	37655	37655	7.19	34947.4	2.71	36635.4	4870	14028	1878.86	TL
TW_34pr107	34349	34349	19.42	27677.0	13.96	29554.6	8897	12852	1514.82	TL
TW_36ch130	3309	3345	3.71	3220.8	0.00	3309.0	7411	6138	1186.88	1756.03
TW_38ch150	3257	3288	7.30	3019.1	4.44	3112.5	2525	12398	2251.49	TL
TW_39gr137	40203	41177	7.73	37995.8	1.72	39512.6	4297	12894	2148.52	TL

The instances in *G1* and *G2* are derived from the same GTSP instances. Instances in *G2* have the maximum number of vertices per cluster set to $N_{\max} = 5$. This feature does not seem to make instance resolution in *G2* any easier than in *G1*. At first glance, one could say that slightly larger instances can be solved in *G2* than in *G1* (36 clusters versus 30 clusters). On the other hand, a more in-depth analysis shows that instances with the same number of vertices are solved faster when they belong to *G1* rather than to *G2*. This can be seen by considering the results on instances with 48 to 137 vertices, i.e., from TW_13gr48 to TW_39gr137. The average computation time is 378.35 seconds for instances in *G1* while it increases to 1313.85 seconds for instances in *G2*. Note that if an instance is not solved to optimality, we

Table 5: Results on *G3* ($\mathcal{F}2$).

Instance	Obj	initS	rGAP%	rLB	fGAP%	fLB	nbNodes	nbCuts	sep-time	time
5-10-1-3	326	326								0.01
5-13-1-1	329	329								0.01
5-14-1-2	186	186								0.01
5-25-1-15	231	231								0.01
6-15-1-9	201	201								0.01
6-16-1-5	292	292								0.01
6-19-1-22	301	301								0.01
6-22-1-4	344	344								0.01
6-24-1-0	226	226								0.01
6-25-1-7	272	272								0.01
6-25-1-17	315	315								0.01
6-29-1-11	283	283								0.01
7-21-1-18	271	271	0.00	271.0	0.00	271.0	0	0	0.01	0.50
7-26-1-27	299	299	0.00	299.0	0.00	299.0	0	5	0.01	0.60
7-27-1-13	267	267	0.00	267.0	0.00	267.0	0	0	0.01	0.62
7-27-1-14	312	312	0.00	312.0	0.00	312.0	0	21	0.01	0.63
7-30-1-29	337	337	0.00	337.0	0.00	337.0	0	81	0.01	0.69
7-31-1-20	245	245	0.00	245.0	0.00	245.0	0	0	0.01	0.71
8-14-1-19	318	318	0.00	318.0	0.00	318.0	0	14	0.01	0.54
8-24-1-10	281	281	0.00	281.0	0.00	281.0	0	67	0.01	0.82
8-24-1-12	273	273	0.00	273.0	0.00	273.0	0	40	0.01	0.93
8-27-1-28	306	306	0.00	306.0	0.00	306.0	0	0	0.01	0.81
8-28-1-8	213	213	0.00	213.0	0.00	213.0	0	19	0.01	0.66
8-29-1-38	216	216	0.00	216.0	0.00	216.0	0	22	0.01	0.92
8-30-1-25	272	272	0.00	272.0	0.00	272.0	0	50	0.01	0.91
8-36-1-23	278	278	0.00	278.0	0.00	278.0	0	74	0.01	0.94
9-26-1-35	347	347	0.00	347.0	0.00	347.0	0	61	0.01	0.72
9-27-1-21	341	341	0.00	341.0	0.00	341.0	0	71	0.01	0.75
9-34-1-16	333	333	0.00	333.0	0.00	333.0	0	83	0.01	1.01
9-34-1-26	232	232	0.00	232.0	0.00	232.0	0	69	0.02	1.22
10-29-1-6	281	281	0.00	281.0	0.00	281.0	0	29	0.01	1.12
10-32-1-31	350	350	0.00	350.0	0.00	350.0	0	267	0.07	1.35
10-40-1-36	324	324	0.00	324.0	0.00	324.0	0	124	0.01	1.45
10-41-1-30	196	196	0.00	196.0	0.00	196.0	0	96	0.01	1.74
10-42-1-24	258	258	0.00	258.0	0.00	258.0	0	132	0.03	1.93
11-27-1-37	333	333	0.00	333.0	0.00	333.0	0	32	0.01	1.44
11-33-1-39	312	312	0.00	312.0	0.00	312.0	0	198	0.07	1.95
11-43-1-32	347	347	0.00	347.0	0.00	347.0	0	46	0.01	1.25
12-45-1-34	260	260	0.00	260.0	0.00	260.0	0	68	0.02	2.21
13-41-1-33	347	347	0.00	347.0	0.00	347.0	0	135	0.03	2.42
14-34-2-19	478	478	0.00	478.0	0.00	478.0	0	123	0.03	2.10
14-41-2-6	514	514	0.00	514.0	0.00	514.0	0	210	0.02	2.64
14-44-2-27	575	575	0.00	575.0	0.00	575.0	0	295	0.10	3.43
14-44-2-28	500	500	0.00	500.0	0.00	500.0	0	120	0.02	3.52
14-52-2-12	622	622	0.00	622.0	0.00	622.0	0	211	0.07	2.69
15-56-2-23	531	531	0.00	531.0	0.00	531.0	0	407	0.03	4.43
16-51-2-38	429	431	0.00	429.0	0.00	429.0	0	356	0.15	5.43
16-60-2-25	589	589	0.00	589.0	0.00	589.0	0	396	0.05	3.59
16-60-2-26	498	498	0.00	498.0	0.00	498.0	0	185	0.04	4.76
16-69-2-24	415	415	0.00	415.0	0.00	415.0	0	2502	0.07	5.57
17-48-2-35	606	606	10.23	544.0	0.00	606.0	7	697	0.78	5.09

Instance	Obj	initS	rGAP%	rLB	fGAP%	fLB	nbNodes	nbCuts	sep-time	time
17-50-2-21	563	563	0.00	563.0	0.00	563.0	0	190	0.03	2.69
19-61-2-31	564	564	11.07	501.6	0.00	564.0	21	2555	4.22	13.38
19-72-2-30	510	510	0.00	510.0	0.00	510.0	0	682	0.09	8.67
20-60-2-37	600	600	0.00	600.0	0.00	600.0	0	610	0.29	5.42
20-82-2-36	643	643	0.00	643.0	0.00	643.0	0	589	0.12	6.94
21-71-2-39	554	554	0.00	554.0	0.00	554.0	0	425	0.23	9.39
21-78-2-32	662	662	0.00	662.0	0.00	662.0	0	1305	0.24	10.47
23-83-2-34	448	448	0.00	448.0	0.00	448.0	0	508	0.18	12.38
23-91-2-33	532	532	0.00	532.0	0.00	532.0	0	448	0.20	11.70
24-76-3-21	835	835	0.00	835.0	0.00	835.0	0	1105	1.17	9.74
24-78-3-38	677	677	1.62	666.1	0.00	677.0	10	5470	2.66	13.32
24-86-3-25	850	850	0.00	850.0	0.00	850.0	0	1169	0.29	13.42
25-76-3-35	830	830	0.00	830.0	0.00	830.0	0	1320	1.13	8.71
28-99-3-31	844	850	3.34	815.8	0.00	844.0	41	6422	21.42	42.57
28-110-3-30	720	720	0.00	720.0	0.00	720.0	0	2087	1.29	27.51
29-95-3-37	850	850	3.45	820.6	0.00	850.0	6	7773	5.76	23.09
29-95-3-39	850	850	0.00	850.0	0.00	850.0	0	2521	0.66	19.54
30-114-3-36	929	929	0.00	929.0	0.00	929.0	0	2001	0.45	20.97
31-118-3-32	921	921	0.00	921.0	0.00	921.0	0	2567	0.57	20.04
31-128-3-33	856	856	0.00	856.0	0.00	856.0	0	1159	0.60	28.67
32-125-3-34	721	721	0.00	721.0	0.00	721.0	0	1875	0.62	36.48

consider the computation time as one hour. Therefore, it seems that instances with the same number of vertices but a larger number of clusters are more difficult to solve. However, this is not always the case, as, for example, TW_20kroA100 in *G1* is solved in 131.33 seconds while TW_26kroA100 in *G2* is solved in 88.10 seconds.

7 Conclusions

In this paper, we have presented two formulations $\mathcal{F}1$ and $\mathcal{F}2$ for the Generalized Traveling Salesman Problem with Time Windows, a new generalization of the classical TSPTW and GTSP. We proposed several families of valid inequalities, which contain polynomial or exponential numbers of constraints. They were incorporated in a branch-and-cut framework through dedicated separation procedures. A high quality initial solution was constructed based on a heuristic and used as a warm start in the branch-and-cut algorithm. We tested the algorithm on three groups of instances with different characteristics. The results clearly demonstrate the efficiency of the proposed branch-and-cut algorithm and the quality of formulation $\mathcal{F}2$. The proposed branch-and-cut algorithm based on formulation $\mathcal{F}2$ can solve instances around 30 clusters within one hour of computation time.

8 Acknowledgment

This work is partially supported by the CSC (China Scholarship Council) and by the ELSAT 2020 project. This support is gratefully acknowledged.

References

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

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

Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410.

Dash, S., Günlük, O., Lodi, A., and Tramontani, A. (2012). A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147.

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

Desrochers, M. and Laporte, G. (1991). Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36.

Dimitrijević, V. and Šarić, Z. (1997). An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs. *Information Sciences*, 102(1-4):105–110.

eMarketer (2018). Retail ecommerce sales worldwide, 2016-2021 (trillions, % change and % of total retail sales). `http://www.emarketer.com/Chart/Retail-Ecommerce-Sales-Worldwide-2016-2021-trillions-change-of-total-retail-sales/215138`. Online, accessed February 2019.

Farber, M. (2016). Consumers are now doing most of their shopping online. `http://fortune.com/2016/06/08/online-shopping-increases/`. Online, accessed March 2019.

Fischetti, M., Salazar González, J. J., and Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394.

Ghiani, G. and Improta, G. (2000). An efficient transformation of the generalized vehicle routing problem. *European Journal of Operational Research*, 122(1):11–17.

Gomory, R. E. and Hu, T. C. (1961). Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570.

Gutin, G. and Karapetyan, D. (2010). A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, 9(1):47–60.

Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130.

Helsgaun, K. (2009). General k-opt submoves for the lin–kernighan tsp heuristic. *Mathematical Programming Computation*, 1(2-3):119–163.

Helsgaun, K. (2015). Solving the equality generalized traveling salesman problem using the lin–kernighan–helsgaun algorithm. *Mathematical Programming Computation*, 7(3):269–287.

Karapetyan, D. (2012). Gtsp instances library. `http://www.cs.nott.ac.uk/~pszdk/gtsp.html`. Online, accessed August 2018.

Karapetyan, D. and Gutin, G. (2012). Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem. *European Journal of Operational Research*, 219(2):234–251.

Laporte, G. and Semet, F. (1999). Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, 37(2):114–120.

Lowe, R. and Rigby, M. (2014). The last mile - exploring the online purchasing and delivery journey. Technical report, Barclays.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329.

Moccia, L., Cordeau, J. F., and Laporte, G. (2012). An incremental tabu search heuristic for the generalized vehicle routing problem with time windows. *Journal of the Operational Research Society*, 63(2):232–244.

Nemhauser, G. and Wolsey, L. (1999). Application of special-purpose algorithm. In *Integer and combinatorial optimization*. John Wiley & Sons.

Noon, C. E. and Bean, J. C. (1991). A lagrangian based approach for the asymmetric generalized traveling salesman problem. *Operations Research*, 39(4):623–632.

Noon, C. E. and Bean, J. C. (1993). An efficient transformation of the generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, 31(1):39–44.

Ozbaygin, G., Karasan, O. E., Savelsbergh, M., and Yaman, H. (2017). A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations. *Transportation Research Part B: Methodological*, 100:115–137.

Padberg, M. W. (1973). On the facial structure of set packing polyhedra. *Mathematical programming*, 5(1):199–215.

Reyes, D., Savelsbergh, M., and Toriello, A. (2017). Vehicle routing with roaming delivery locations. *Transportation Research Part C: Emerging Technologies*, 80:71–91.

Smith, S. L. and Imeson, F. (2017). Glns: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, 87:1–19.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265.