Les Cahiers du GERAD ISSN: 0711–2440

Traveling salesman problem with time windows in postal services

A. Bretin, G. Desaulniers, L.-M. Rousseau

G–2018–30

April 2018

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

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

Traveling salesman problem with time windows in postal services

Alexis Bretin [a, b, c]

Guy Desaulniers [a, b]

Louis-Martin Rousseau [b, c]

[a] *GERAD HEC Montréal, Montréal (Québec), Canada, H3T 2A7*

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

[c] *CIRRELT, Montréal (Québec), Canada, H3T 1J4*

alexis.bretin@polymtl.ca
guy.desaulniers@gerad.ca
louis-martin.rousseau@cirrelt.net

April 2018
Les Cahiers du GERAD
G–2018–30

Abstract: This paper focuses on the traveling salesman problem with time windows (TSPTW) that arises in postal services and parcel deliveries and has features differing from the classical TSPTW. First, route duration is a significant concern, as human costs largely exceed vehicle costs, emphasizing the importance of reducing waiting time. Second, if commercial customers usually have a time window, private customers do not, making the NP-hard TSPTW even harder to solve. To address these issues, we present two multi-objective approaches based on a constraint programming formulation of the problem which allows to balance the optimization of both human and material resources. We also introduce a *cluster/solve/sequence* approach to reduce the size of the large real-world instances, which relies on a mathematical programming formulation to *sequence* the customer visits in the final step. This decomposition technique allows to produce high-quality solutions for industrial problems in about a minute of computation time.

Keywords: Traveling Salesman Problem with Time Windows, postal services, route duration, bi-objective, low time-window density, clustering heuristic

Acknowledgments: We are thankful to the personnel of GIRO Inc., in particular, Charles Fleurent and Patrick Saint-Louis, who provided to us the problem definition and the real-life datasets. We gratefully acknowledge the financial support of GIRO Inc. and the Natural Sciences and Engineering Research Council of Canada under the grant RDCPJ 4634633-14.

1 Introduction

In postal services, letters and parcels are not handled in the same way. Although each delivery employee is assigned to a predefined territory, the routes are not managed identically. For letters which are increasingly being replaced by email, the employee must still visit both sides of every street in the assigned area. For parcel deliveries, only a few residents and businesses must be visited. We consider the parcel-delivery problem with time windows (TWs). The TWs are designed to synchronize deliveries and recipients. This can be formulated as a vehicle routing problem with TWs (VRPTW). For large-scale instances involving 10 to 20 territories and up to almost 500 deliveries in a territory, it can be solved via a greedy cluster-first route-second (CFRS) procedure. The clustering phase determines well-designed territories according to the specific characteristics of the instance, and the routing phase designs the routes. The two phases are carried out alternatively in the following greedy fashion. First, a cluster containing a number of service points is proposed. Second, a route is determined for this cluster. If it is too long or too short compared to a usual employee working shift, the set of points proposed for this territory is adjusted and a new route is computed. This process repeats until obtaining a satisfactory route for this territory. When this is accomplished, the algorithm moves on to the next territory. The overall objective is to find a good solution in a limited time, generally at most one minute per territory. Our goal is to show that exact methods can be useful even with severe computational time restrictions. We focus on the routing phase, which is a time-constrained variant of a traveling salesman problem (TSP), called the TSP with TWs (TSPTW). It involves visiting every service point once and only once with a single vehicle while minimizing the cost of the route. In the traditional TSPTW, each customer must be visited within a restricted part of the horizon, called a TW.

The TSPTW has been extensively studied. Savelsbergh [14] showed that finding a feasible solution to this problem is NP-hard. From a 1981 branch-and-bound algorithm using a state-space-relaxation approach to compute the lower bounds [6], to dynamically generated time-expanded networks in 2017 [13], many approaches have been developed. To the best of our knowledge, the most efficient algorithm for the TSPTW when the objective function is to minimize the distance traveled is that of Baldacci et al. [5]. They combine column generation to compute lower bounds with dynamic programming to find feasible solutions based on the last bound found, given a certain tolerance.

Most studies minimize the distance traveled or the closely related travel time (TT). In postal services, focusing on TT may be costly. The best solution may have considerable waiting time, and drivers must be paid when they are waiting, i.e., when they arrive at a service point before the TW opens. With this in mind, we instead minimize the route duration (RD). It may be assimilated to the makespan, which in scheduling problems is the amount of time necessary to complete all the tasks. It could also be seen as the sum of the TT and the waiting time or as the completion time of the final task. However, when there are TWs, starting the route at time 0 may not be necessary or optimal.

Two other considerations led us to focus on RD. First, the average wage for a Canada Post delivery driver is around \$20/hour [2]. For a vehicle such as a Ford Transit, the fuel consumption is around 12 L/100km [1]. If the average speed is between 30 and 50 km/h the approximate cost of one hour of driving is $40 * 12/100 * 1.2 =$ \$5.76 where 1.2 is the average price of a liter of gasoline in Canada [3]. This may be increased to \$8 to take into account the depreciation of the vehicle and the fact that the fuel consumption is generally slightly higher than the manufacturer's nominal value. Therefore, it is profitable to extend the TT by 2 min to save 1 min of RD, and the idle time is reduced by 3 min. The ratio may vary, but the analysis suggests that one unit of RD is more valuable than one unit of TT. Reducing RD may also reduce idle time, and so improve driver satisfaction. In summary, the TT is the vehicle cost whereas the RD is the human cost of the route.

Second, in a CFRS approach to the VRPTW, minimizing the RD may make it possible to visit more clients on a route, thus reducing the number of territories. Routes that are more compact allow a better knowledge of the neighbourhood, enabling the drivers to make better decisions in the event of unexpected congestion or construction.

Reducing the RD may also be relevant in other applications. In armoured-truck routing, the less time a vehicle spends on the road, the safer the operation, and in fresh-food delivery, the less time the goods spend in the vehicle, the better their condition on arrival.

According to López-Ibáñez et al. [11], who minimize the completion time of the route, this variant of the problem can generally be solved more quickly. Savelsbergh [15] addressed the problem of unwanted idle time by introducing the concept of forward slack time in the context of an arc swap algorithm.

TWs are particularly useful for companies, where there are dedicated employees to receive and process deliveries. However, because of the rise in online shopping, a growing number of individual customers receive parcel deliveries. Some delivery companies assign TWs to only the largest customers (e.g., stores or industries) and not to individuals. This gives rise to instances in which less than 10% (perhaps just 2% or 3%) of the service points have TWs. This makes the problem harder because less preprocessing can be done to reduce the solution space.

The contribution of this paper is to identify proper solution approaches to large-scale, multi-objective TSPs with low TW density that arise in the postal services. To our knowledge, we are the first to handle the combined TT and RD objective functions, which we address with mathematical programming and constraint programming models. Furthermore we devise an efficient clustering approach to enable the solution of large instances within a tight computational budget. Finally, we evaluate our methods on instances from the literature as well as real industrial instances provided by our research partner.

After introducing a basic CP model for the TSPTW in Section 2, we will explain in Section 3 how to solve this problem while balancing the TT and RD minimization, using two multiobjective approaches. Section 4 presents a clustering-disaggregation algorithm for instances with a low TW density. Section 5 gives results for benchmark instances, and Section 6 reports results for real-world instances.

2 Problem formulation

Consider a directed graph $G = (V, A)$ where V contains the nodes, including p and q respectively representing the depot at the beginning and the end of the tour, and A contains the arcs. The TSPTW consists of finding a single tour that visits each node $i \in V$ exactly once while minimizing the routing cost. The visit at node i must be within the TW $[R_i, D_i]$ where R_i is the release time and D_i the deadline of node i. A service duration s_i may be associated with each $i \in V$. A TW defines the period in which the service to this node must occur. For node i, $s_i > 0$ indicates that the service could end after D_i but must begin before. Early arrival at node i is allowed, but the driver must wait until R_i. Because of the TWs, A does not contain all the couples $(i, j) \in V^2$ but only a subset; see Section 3.1. Traveling along arc $(i, j) \in A$ takes t_{ij} time units.

We first present a CP formulation and then explain the use of bi-objective programming to minimize TT and RD.

2.1 Constraint programming model

CP is known to be efficient for machine scheduling problems. Since the TSPTW is equivalent to a one-machine scheduling problem with sequence-dependent setup times and TW constraints, a simple but reliable CP formulation is available. We introduce the variable $P_l, \forall l \in \{1, \ldots, |V|\}$, which models the l^{th} service point visited, and T_{P_l}, the time of the visit to this node.

$$\min z = T_q - T_p \tag{1}$$

subject to:

$$\texttt{AllDifferent}(P) \tag{2}$$

$$T_{P_l} + s_{P_l} + t_{P_l P_{l+1}} \leq T_{P_{l+1}} \qquad \forall l \in \{1, \ldots, |V| - 1\} \tag{3}$$

$$P_1 = p \tag{4}$$

$$P_{|V|} = q \tag{5}$$

$$P_l \in V, \qquad \forall l \in \{1, \ldots, |V|\} \tag{6}$$

$$T_{P_l} \in [R_{P_l}, D_{P_l}], \qquad \forall l \in \{1, \ldots, |V|\}. \tag{7}$$

The goal of minimizing the RD corresponds to $\min z = T_{P_{|V|}} - T_{P_1}$, but since we constrain the route to begin and end at the depot via constraints (4) and (5), we may write the objective function as shown in (1). Constraint (2) indicates that the service points visited for $l \in \{1, \ldots, |V|\}$ must be different. Combined with constraints (6), this ensures that every service point is visited exactly once. Constraints (3) ensure that there is sufficient time between two consecutive service point visits. Constraints (7) ensure that the TWs are respected.

As mentioned in Section 1, we distinguish our notion of RD from the makespan or the completion time, as used in machine scheduling. We define the RD to be the difference between the departure time from and the return time to the depot.

The model can be adapted for the TT problem by replacing (1) by

$$\min z = \sum_{l=1}^{|V|-1} t_{P_l P_{l+1}}. \tag{8}$$

2.2 Multiobjective model: Weighted objective function

For postal services, where the TSPTW is solved many times in a CFRS procedure, we wish to process as many customers as possible within the duration of the driver's shift. However, preliminary results have shown a significant reduction in the quality of the TT when it does not appear in the objective function. It sometimes causes a slight increase in the cost of the final solution. Here we are referring to the cost in dollars, calculated via the parameters γ^{TT} and γ^{RD} corresponding to the cost of one unit of time when respectively the vehicle is running or the driver is working. The reduction in the RD does not always compensate for the substantial increase in the TT. This led us to explore how to make the solution more profitable while trying to get the RD as low as possible.

There are many algorithms for multiobjective optimization problems; see [8]. A simple approach is to use a weighted objective function (WOF) that combines the various objectives into a single function. The challenge is to appropriately weight each objective. We decided to use the costs γ^{TT} and γ^{RD} as the weights. The CP model remains the same except that the objective function becomes

$$\min \quad \gamma^{TT} \sum_{l=1}^{|V|-1} t_{P_l P_{l+1}} + \gamma^{RD}(T_q - T_p). \tag{9}$$

The main drawbacks of this model are the sensitivity to the costs γ^{TT} and γ^{RD} and the lack of control of the RD objective function.

3 Solution approach

In this section, we describe the approach that we propose to solve the bi-objective TSPTW. First, we briefly discuss a preprocessing phase that can be conducted to reduce the solution space. Next, we introduce the CP-Optimizer implementation that we apply to solve the CP model with the objective of either minimizing the RD or the TT. This algorithm is used in the three-step approach that is described afterwards.

3.1 Preprocessing

The presence of TWs enables some preprocessing. Taking into account the spatiotemporal structure of the instance may reduce the solution space, and thus the combinatorial nature of the problem. TWs may be tightened and node precedence lists built [4]. For each node $i \in V$, the precedence list $V^+(i) = \{j \in V \backslash \{i\}, j \prec i\}$ gathers all the nodes j that have to be visited before i. Some arcs may be deleted based on the TWs and the precedence relationships [7].

3.2 CP-Optimizer implementation

CP-Optimizer, the Cplex solver dedicated to CP, contains many useful tools for scheduling problems. To use them, we have to rethink the model introduced in Section 2.1. The visits are associated with tasks that are modelled by a set of interval variables H. A visit to service point i is represented by the variable H_i. These variables have an associated length, which is equivalent to the service duration s_i, and earliest and latest start times, modelling the TWs. To solve the problem, we use a sequence interval variable $\pi = Sequence(H)$ that represents the full sequence performed by the driver. By definition, it generates a circuit in V through all the service points. The CP-Optimizer implementation has many similarities with the model introduced in Section 2.1, with $H_i.Begin = T_i$ being the time when delivery starts at node i.

With these variables and the notation introduced in Section 2, the CP model for minimizing RD may be implemented as follows:

$$\min z = H_q.Begin - H_p.Begin \tag{10}$$

subject to:

$$\pi = Sequence(H) \tag{11}$$
$$NoOverlap(\pi, tm) \tag{12}$$
$$\pi.First = H_p \tag{13}$$
$$\pi.Last = H_q \tag{14}$$
$$H_i.Begin \in [R_i, D_i], \qquad \forall i \in V. \tag{15}$$

The objective function (10) minimizes the RD. Constraint (11) defines π as a sequence of the complete set of tasks, H. In constraint (12) tm is the transition matrix, which is identical to the distance/time matrix described in Section 2 by t_{ij}. This constraint controls the passage time to every point, helped by the function NoOverlap. The latter prevents overlapping tasks, i.e., a visit to node j cannot begin until the previous visit to i has finished and time has been allowed to travel from i to j. It is similar to the disjunctive constraints (3) but is computationally more effective. All the visit times are restricted to the associated TWs via constraints (15), which are implemented directly when we define the interval variables.

The objective function (10) can again be modified for TT minimization:

$$\min z = \sum_{i \in V \setminus \{q\}} t_{ix_i} \tag{16}$$

where $x_i = H_i.Next$ represents the successor of node i in the sequence π.

3.3 Three-step approach

Controlling the RD is essential in postal services, since the problem cannot be reduced to a single TSPTW. In a CFRS approach to VRPTW, the first step generally aims to reduce the number of vehicles. To achieve this, the territories must include as many clients as possible while respecting the maximum duration of the driver schedules. TSPTWs are used for this, and from a global point of view, some more costly routes with a shorter RD may be preferred. On the other hand, we cannot focus only on the RD. When some TWs are restrictive, there may be idle time in the optimal RD solution. There are potentially multiple solutions with the same RD value within a given tolerance. Our goal is to find those with a shorter TT.

This is why we do not use a weighted sum of the two objective functions. We instead solve a multiobjective optimization problem with an ϵ-constraint. For a discussion of ϵ-constraint methods, see [12]. The main idea is to limit one or several objective functions by adding a new constraint to the model.

The proposed approach has three steps and is called the three-step approach (TSA):

Step 1: Minimize RD (let Z^*_{RD} be the best RD value found);
Step 2: Minimize TT subject to $Z_{RD} \leq Z^*_{RD} + \epsilon$ ($\epsilon \in \mathbb{N}$ is a tolerance);
Step 3: Minimize RD subject to the sequence computed in the previous step.

As seen in [11], solving the TSPTW-makespan (considered as the completion time of the final task) is generally not as hard as minimizing TT. In the TSA, the CP model (10)–(15) used for Step 1 can be solved efficiently. For Step 2, we add the constraint

$$H_q.Begin - H_p.Begin \leq Z^*_{RD} + \epsilon \tag{17}$$

to model (16), (11)–(15). Choosing a tight tolerance ϵ helps us to control the RD. Depending on the value of $Z^*_{RD} + \epsilon$, the TW for the return to the depot may be tightened to $[R_q, \min\{D_p + Z^*_{RD} + \epsilon, D_q\}]$. If $D_p + Z^*_{RD} + \epsilon < D_q$, then the procedure presented in Section 3.1 may be applied again and the solution space reduced once more. Step 3 ensures that the arrival times associated with the sequence found in Step 2 do not generate avoidable idle time. This is done by solving again model (10)–(15) but with a fixed sequence.

For Step 2, we also tried a MIP model based on the time bucket formulation (TBF) [7] presented in the Appendix. It is competitive when the instances are small and/or have tight TWs but is outperformed by CP as the instances grow in size. Since postal-services instances are generally large, we do not explore TBF further for this algorithm but it will become useful in the disaggregation phase of the clustering approach described next.

4 Clustering approach

Some delivery companies assign TWs only to the largest customers and not to individuals. When there are fewer TWs, minimizing the RD also helps to reduce the TT: the idle time in the solution is sparse, so every improvement in the RD is generally obtained by reducing the TT. When dealing with instances where the TW density is critically low (below 5%) we will focus on RD minimization.

In this section, we develop a clustering approach for solving large-scale instances with sparse TWs. This approach proceeds in three steps. First, it clusters some service points to reduce the size of the instance, yielding a so-called clustered instance that approximates the original one. Second, it solves this clustered instance. Finally, given the sequence of the service points in the computed tour, the clusters are disaggregated to derive a solution to the original instance.

4.1 Clustering

Traditional models use the TWs to reduce the solution space; we instead cluster some service points. Starting from the original network G, we create a clustered network, i.e., a network which contains fewer nodes and arcs. Each node in the clustered network represents one or several nodes in V. The depot nodes p and q and the customer nodes with TWs are never clustered with other nodes. The proposed clustering algorithm is iterative. At each iteration, it clusters a pair of nodes without TWs (which may represent several original nodes) to create a new node associated with the location of one of the nodes it represents. The distance matrix is then updated after computing a Hamiltonian path passing through all the nodes represented by the new node.

A pseudo-code of the clustering algorithm is given in Algorithm 1. The following notation is used. First, we number the nodes in V from 0 to $|V|-1$ and associate nodes p and q to numbers 0 and $|V|-1$. Thus, the customer nodes are numbered from 1 to $|V|-2$. Let $J = \{1, \ldots, |V|-2\}$ and $\tilde{J} \subset J$ be the nodes of the customers with a TW. At each iteration i of the algorithm, the clustered instance is represented by the following vectors and matrix.

N_i: Vector of dimension $|V|-2$ of subsets of customer nodes. For all $j \in J$, $N_i(j)$ is the (possibly empty) subset of customers that are represented by customer j. This vector has the following properties: $N_i(j) = \{j\}$, $\forall j \in J \setminus \tilde{J}$; $\bigcup_{j \in J} N_i(j) = J$; $N_i(j_1) \cap N_i(j_2) = \emptyset$, $\forall (j_1, j_2) \in J^2$ such that $j_1 \neq j_2$.

S_i: Vector of dimension $|V|-2$ of total service times. For all $j \in J$, $S_i(j) = \sum_{v \in N_i(j)} s_v$ if $N_i(j) \neq \emptyset$. Otherwise, the value of $S_i(j)$ is irrelevant.

ITT_i: Vector of dimension $|V|-2$ of the node internal traveling times. For all $j \in J$, $ITT_i(j) = 0$ if $|N_i(j)| \leq 1$. Otherwise, $ITT_i(j)$ approximates the minimal travel time required to visit all customers in $N_i(j)$.

M_i: Matrix of dimension $|V|-2 \times |V|-2$ of the arc traveling times. For all $(j_1, j_2) \in J^2$, $M_i(j_1, j_2) = \infty$ if $j_1 = j_2$, $N_i(j_1) = \emptyset$ or $N_i(j_2) = \emptyset$. Otherwise, $M_i(j_1, j_2)$ approximates the traveling time from j_1 to j_2, including the internal traveling time at j_1.

Algorithm 1 Clustering

1: Initialize N_0, S_0, ITT_0, and M_0
2: $(k_1, l_1) \leftarrow \text{argmin}_{(k,l)\in\bar{J}^2} M_0(k, l)$
3: $d_1 \leftarrow M_0(k_1, l_1)$
4: $i \leftarrow 1$
5: **while** $d_i \leq d_{max}$ **do**
6: $\quad \mathcal{C}_i \leftarrow N_{i-1}(k_i) \cup N_{i-1}(l_i)$
7: $\quad N_i \leftarrow N_{i-1}$, $S_i \leftarrow S_{i-1}$, $ITT_i \leftarrow ITT_{i-1}$, and $M_i \leftarrow M_{i-1}$
8: $\quad$ **if** $d_i = 0$ **then**
9: $\quad\quad N_i(k_i) \leftarrow \mathcal{C}_i$ and $N_i(l_i) \leftarrow \emptyset$
10: $\quad\quad S_i(k_i) \leftarrow S_i(k_i) + S_i(l_l)$
11: $\quad\quad M_i(j, l_i) = M_i(l_i, j) = \infty, \quad \forall j \in J$
12: $\quad$ **else**
13: $\quad\quad p'_i \leftarrow \text{argmin}_{v\in V\setminus\mathcal{C}_i} \sum_{j\in\mathcal{C}_i} t_{vj}$ and $q'_i \leftarrow \text{argmin}_{v\in V\setminus(\mathcal{C}_i\cup\{p'_i\})} \sum_{j\in\mathcal{C}_i} t_{vj}$
14: $\quad\quad$ **if** $D_{q'_i} < R_{p'_i}$ **then**
15: $\quad\quad\quad (p'_i, q'_i) \leftarrow (q'_i, p'_i)$
16: $\quad\quad$ Compute a shortest Hamiltonian path $(p'_i, v_i^1, \ldots, v_i^{|\mathcal{C}_i|}, q'_i)$ between p'_i and q'_i and passing through all nodes $v_i^1, \ldots, v_i^{|\mathcal{C}_i|}$ in $\mathcal{C}_i$
17: $\quad\quad N_i(v_i^1) \leftarrow \mathcal{C}_i$ and $N_i(j) \leftarrow \emptyset, \quad \forall j \in \mathcal{C}_i \setminus \{v_i^1\}$
18: $\quad\quad S_i(v_i^1) \leftarrow S_i(k_i) + S_i(l_l)$
19: $\quad\quad ITT_i(v_i^1) \leftarrow \sum_{j=1}^{|\mathcal{C}_i|-1} t_{v_i^j v_i^{j+1}}$ and $ITT_i(j) \leftarrow 0, \quad \forall j \in \mathcal{C}_i \setminus \{v_i^1\}$
20: $\quad\quad$ **if** $k_i \neq v_i^1$ **then**
21: $\quad\quad\quad M_i(j, k_i) = M_i(k_i, j) = \infty, \quad \forall j \in J$
22: $\quad\quad$ **if** $l_i \neq v_i^1$ **then**
23: $\quad\quad\quad M_i(j, l_i) = M_i(l_i, j) = \infty, \quad \forall j \in J$
24: $\quad\quad$ **for** $j \in V \setminus \mathcal{C}_i$ such that $V_i(j) \neq \emptyset$ **do**
25: $\quad\quad\quad M_i(v_i^1, j) \leftarrow ITT_i(v_i^1) + t_{v_i^{|\mathcal{C}_i|}, j}$
26: $\quad\quad\quad M_i(j, v_i^1) \leftarrow ITT_i(j) + t_{j, v_i^1}$
27: $\quad i \leftarrow i + 1$
28: $\quad (k_i, l_i) \leftarrow \text{argmin}_{(k,l)\in\bar{J}^2} M_{i-1}(k, l)$
29: $\quad d_i \leftarrow M_{i-1}(k_i, l_i)$

The algorithm starts by initializing these vectors and matrix as follows. For all $j \in J$, $N_0(j) = j$, $S_0(j) = s_j$, and $ITT_0(j) = 0$. For all $(j_1, j_2) \in J^2$, $M_0(j_1, j_2) = \infty$ if $j_1 = j_2$. Otherwise, $M_0(j_1, j_2) = t_{j_1 j_2}$. In Step 2, the algorithm finds in matrix M_0 the minimal traveling time d_1 and selects a pair of nodes (k_1, l_1) yielding this time. It then sets the iteration counter i to 1 and check if d_i does not exceed a predetermined maximum time d_{max}. If this is the case, iteration i will merge the selected nodes k_i and l_i into a single node that will contain the customer nodes in $\mathcal{C}_i$. It starts by making copies of the vectors and matrix N_{i-1}, S_{i-1}, ITT_{i-1} and M_{i-1}. These copies will be updated to define the clustered instance at iteration i.

Two cases can occur. If $d_i = 0$ (test at Step 8), a case that may happen when several customers are located in the same building, all customers represented by k_i and l_i are arbitrarily assigned to k_i and all information related to l_i become obsolete. Note that, if $d_i = 0$, the traveling time between any pair of nodes in $\mathcal{C}_i$ is equal to 0 because in all previous iterations i', $d_{i'}$ was also equal to 0. If $d_i > 0$, the merging process is more complex. Given that, in the current clustered instance, the customers in $\mathcal{C}_i$ will be visited consecutively, we would like to approximate the minimal traveling time ITT required to visit the customers in $\mathcal{C}_i$. To do so, we first select two nodes p'_i and q'_i not in $\mathcal{C}_i$ that might immediately precede and succeed to the node representing $\mathcal{C}_i$ in the solution. We choose these nodes as the two closest neighbours on average to all nodes in $\mathcal{C}_i$ (Step 13). If they have TWs and are badly ordered, we switch them in Step 15. We then compute in Step 16 a shortest Hamiltonian path that starts in p'_i, ends in q'_i and visits all nodes in $\mathcal{C}_i$. ITT is then set to the total traveling time between the first node v_i^1 after p'_i and the last node $v_i^{|\mathcal{C}_i|}$ before q'_i in this path, i.e, $ITT = \sum_{j=1}^{|\mathcal{C}_i|-1} t_{v_i^j v_i^{j+1}}$. Given that $|\mathcal{C}_i|$ is relatively small (less than 25 in our tests), these shortest paths can be computed rapidly with a general-purpose MIP solver. The customers in $\mathcal{C}_i$ are then assigned to node v_i^1 in Step 17. Some entries of the traveling time matrix M_i are updated in Steps 20 to 26. Note that, in this last

step, we could use t_{j',v_i^1} instead of t_{j,v_i^1} where j' is the next-to-last node in a shortest Hamiltonian path going through all nodes in $N_i(j)$. Given that j and j' are typically very close to each other when $j' \in N_i(j)$, we have decided to use t_{j,v_i^1} as a proxy for t_{j',v_i^1}. The current iteration ends by increasing the iteration counter and selecting in Step 28 the next pair of nodes to cluster.

Once the algorithm terminates, the clustered instance is obtained by creating one node for each non-empty subset $N_{i-1}(j)$, $j \in J$, using the corresponding service times in vector S_{i-1} and the corresponding traveling times in matrix M_{i-1}. The vector ITT_{i-1} is not required as the internal traveling times are included in the traveling times provided in matrix M_{i-1}.

Algorithm 1 stops when the traveling time d_i between the next pair of nodes (k_i, l_i) to cluster exceeds a predetermined threshold as clustering these two nodes seems riskier. Indeed, in preliminary tests, we observed that being too aggressive in the clustering phase may have significant drawbacks as too many approximations are introduced especially when updating the distance matrix. To avoid this, we also tested two other criterion. The first is to stop clustering when the number of nodes in the clustered instance reached a prefixed minimum number. The second is local and forbids clustering too many original nodes into a single one, i.e., a maximal cardinality on each subset $N_i(j)$, $j \in J$, is imposed.

4.2 Solution of the clustered instance

After the clustering, we solve the clustered instance using our CP-model (Section 2.1). Even if the clustered instance is smaller than the original one, there are still some TWs, and the problem remains complex. Nevertheless, a clustered instance with fewer than 100 nodes can be solved in a reasonable time.

4.3 Disaggregation

After solving the clustered instance to find a clustered solution denoted by *Seq*, we compute a solution to the original instance, respecting the sequence of *Seq* to a certain extent. This solution is found by solving the original problem augmented by the following precedence constraints.

Let us partition the nodes in *Seq* (except the depot nodes) in two subsets K and W. Subset $K = \{K_1, \ldots, K_{n^K}\}$ contains the n^K nodes without a TW, hereafter called the clustered nodes even if such a node corresponds to a single original node. Subset $W = \{W_1, \ldots, W_{n^W}\}$ contains the n^W nodes with a TW (they all remained unclustered). Node K_i is the i^{th} visited clustered node, but not necessarily the i^{th} node of *Seq* if some nodes of W are visited earlier. Let j^- (resp. j^+) be the index i of the closest clustered node K_i before (resp. after) W_j in *Seq*. We assume that $i = 0$ if W_j is the first node visited in *Seq* and that $i = n^K + 1$ if W_j is the last one visited.

Let $\theta \geq 1$ be an integer flexibility parameter that indicates how many neighboring clustered nodes may be merged. The precedence constraints are divided into two groups, depending on the types of nodes involved.

- For pairs of clustered nodes, the following constraints allow rearrangement of the original nodes, respecting *Seq* with flexibility θ:

$$a \prec b, \quad \forall i \in \{1, \ldots, n^K - \theta\}, \quad \forall a \in \mathcal{C}_{K_i}, \quad \forall b \in \mathcal{C}_{K_{i+\theta}} \tag{18}$$

  where, as in Section 3.1, $a \prec b$ means that a must be visited before b. When $\theta = 1$, the nodes within each clustered node can be rearranged but they must respect their order with respect to the nodes in the preceding and succeeding clustered nodes.
- The nodes with TWs are now allowed to merge with their θ closest clustered neighbors:

$$b \prec W_j, \quad \forall\, W_j \in W \mid j^- - \theta \geq 1, \quad \forall b \in \mathcal{C}_{K_{(j^- - \theta)}} \tag{19}$$

$$a \succ W_j, \quad \forall\, W_j \in W \mid j^+ + \theta \leq n^W, \quad \forall a \in \mathcal{C}_{K_{(j^+ + \theta)}} \tag{20}$$

  If $\theta = 1$, every node W_j can be reordered with the nodes in its immediate predecessor and successor clustered node to allow a better positioning of this time constrained node.

As an example, let $Seq = \{p - \hat{A} - \hat{B} - W_1 - \hat{C} - W_2 - \hat{D} - q\}$ be the solution of the clustered problem, where $K = \{\hat{A}, \hat{B}, \hat{C}, \hat{D}\}$ is the clustered node set and W_1 and W_2 are nodes with TWs. Let $\theta = 1$.

The precedence constraints are:

$$a \prec b, \quad \forall a \in \mathcal{C}_{\hat{A}}, \quad \forall b \in \mathcal{C}_{\hat{B}} \tag{21}$$
$$b \prec c, \quad \forall b \in \mathcal{C}_{\hat{B}}, \quad \forall c \in \mathcal{C}_{\hat{C}} \tag{22}$$
$$c \prec d, \quad \forall c \in \mathcal{C}_{\hat{C}}, \quad \forall d \in \mathcal{C}_{\hat{D}} \tag{23}$$

for the clustered nodes, and

$$W_1 \succ a, \quad \forall a \in \mathcal{C}_{\hat{A}} \tag{24}$$
$$W_1 \prec d, \quad \forall d \in \mathcal{C}_{\hat{D}} \tag{25}$$
$$W_2 \succ b, \quad \forall b \in \mathcal{C}_{\hat{B}} \tag{26}$$

for the nodes with TWs.

For more flexibility, θ can be increased. In our example, with $\theta = 2$, (21)–(23) are replaced by

$$a \prec c, \quad \forall a \in \mathcal{C}_{\hat{A}}, \quad \forall c \in \mathcal{C}_{\hat{C}} \tag{27}$$
$$b \prec d, \quad \forall b \in \mathcal{C}_{\hat{B}}, \quad \forall d \in \mathcal{C}_{\hat{D}} \tag{28}$$

and (24)–(26) become

$$W_2 \succ a, \quad \forall a \in \mathcal{C}_{\hat{A}}. \tag{29}$$

Note that no precedence constraints are imposed between two nodes in W, giving the opportunity to change their order of visit if this change respects the TWs and the precedence relationships between these nodes and the neighbouring clustered nodes. Taking into account the precedence constraints, the problem can be solved using one of the two methods described in Sections 4.3.1 and 4.3.2.

In our computational experiments, we also consider a case $\theta = 0$ for which the precedence constraints are not described as above. When $\theta = 0$, the precedence constraints ensure that the order of the service points in each clustered node is preserved. Moreover, the service points with TWs cannot be mixed with their clustered neighbours.

4.3.1 Disaggregation phase: Pure CP

The value of the objective function found for the clustered instance is an upper bound for the original problem. Because of the time limit, we may not find an optimal solution to the clustered problem. Moreover, the internal path of each clustered node, visiting for instance the customers in $\mathcal{C}_i$, has been proved optimal for only one pair of closest neighbours (p'_i, q'_i); this sequence may or may not be respected in the optimal solution. This upper bound can be used to tighten the TWs on the nodes p and q representing the depot at the beginning and end of the tour.

In the pure CP case, the clustered solution is disaggregated by solving the original problem augmented by the precedence constraints using CP.

4.3.2 Disaggregation phase: CP and TBF

Empirically, we have observed that after the first second of computation, few improvements are made to the disaggregated solution. Since CP often struggles at proving the optimality of a solution, we use a MIP model, namely, the TBF presented in the Appendix, to close the optimality gap. Given that the TBF is not well designed to minimize RD, we propose a disaggregation procedure similar to the TSA of Section 3.3 in which the TBF is used to minimize TT.

Step Dis-1: Minimize RD via CP with a time limit of 1 s to obtain a good upper bound for Step Dis-2.

Step Dis-2: Minimize TT via the TBF after imposing an upper bound on the RD.

Step Dis-3: Minimize RD via CP subject to the sequence computed in the previous step.

As in Section 4.3.1, precedence constraints are added to the original formulation, based on the solution found for the clustered instance and θ. Additionally, an artificial TW derived from the visiting times T_i of the clustered nodes $K_i \in K$ in the clustered solution is created for every service point that was not originally time-constrained. For node $a \in \mathcal{C}_{K_i}$, the artificial TW is $[T_{i-\theta}, T_{i+1+\theta}]$ because a node in $\mathcal{C}_{K_i}$ should be visited after the first node in $\mathcal{C}_{K_{i-\theta}}$ and before the last one in $\mathcal{C}_{K_{i+\theta}}$. In this procedure, we want to use the strength of TBF to help minimize the RD, since it is closely related to the TT, especially given the sparse TW density.

5 Results on benchmarks

In this section, we consider instances from the literature. Most of the instances have dense TWs, but the results show that the human cost (RD) is higher than the vehicle cost (TT), and focusing on TT may not be appropriate. We tested two sets of well-known instances: the Gendreau et al. (Section 5.1) and Ascheuer et al. (Section 5.2) benchmarks.

The results were obtained on an Intel(R) Xeon(R) X5675 at 3.07 GHz with a single thread. The code was compiled in C++ and uses CP-Optimizer 12.7.1.0. Since the TSPTW may be solved many times, we imposed tight time limits. For the TSA, we limited Step 1 and Step 2 to 60 s each but Step 1 typically required just a few seconds. Parameter ϵ was set equal to 5 unless otherwise specified. When directly solving the WOF model by CP, a time limit of 60 s was enforced and we used $\gamma^{TT} = 8$ and $\gamma^{RD} = 20$ except for the parameter sensitivity analysis.

In the following, we compare the solutions obtained with two different approaches, for instance, with the WOF model versus when minimizing TT only. The histograms show a) the savings in percentage, represented by the dots in the figures, given a cost of $\gamma^{RD} = \$20$/h for RD and $\gamma^{TT} = \$8$/h for TT and b) the differences in TT and RD between the solution computed by the first approach and the reference one. For example, in Figure 1, the bars represent the variations in TT (blue) and RD (red) between the solutions obtained with the WOF and when minimizing TT only.

5.1 Gendreau et al. instances

These instances, hereafter called the GHLS instances, extend the TWs of the Dumas et al. instances [9] and are presented in Gendreau et al. [10]. They are referred to as nXXwYY where XX is the number of nodes, and YY indicates the width of the TWs. There are five instances for each XX-YY combination, for a total of 105 instances. In most cases the RD solution is found and proved optimal within 1 s, although n80w160.002 requires 93 s.

5.1.1 WOF results

Compared to the TT solutions, the WOF solutions yield savings for an instance class ranging from -0.24% to 6.76%, and an average global saving of 2.53%. Figure 1 shows that the increase in the TT is usually compensated for by the reduction in the RD, and so the delivery companies could have shorter routes and lower their routing costs. Figure 2 shows that when there are many TWs, even if they are relatively wide, it is too costly to minimize only the RD. Indeed, in most cases, the WOF solutions exhibit a small increase in RD but a large decrease in TT compared to the solutions obtained by minimizing only RD.

5.1.2 TSA results

Figure 3 compares the solutions of the TSA and of the WOF model. Neither method clearly outperforms the other, but on average the solutions of the WOF model give slightly better costs (savings of 2.53% over the solution obtained by minimizing TT only, compared to 2.35% for TSA).

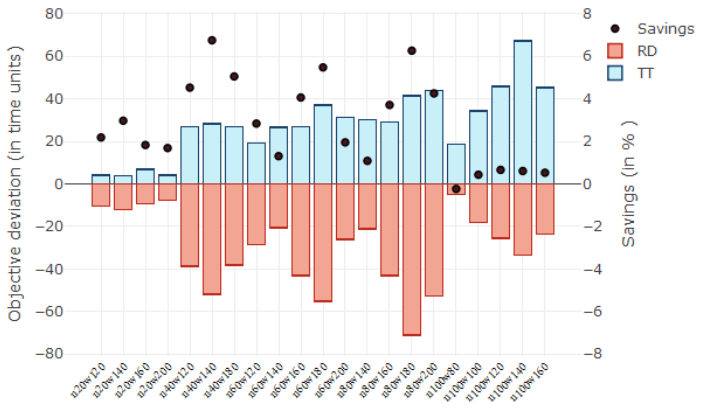

Figure 1: Savings and average TT and RD variations per class, for WOF versus TT only (GHLS instances)

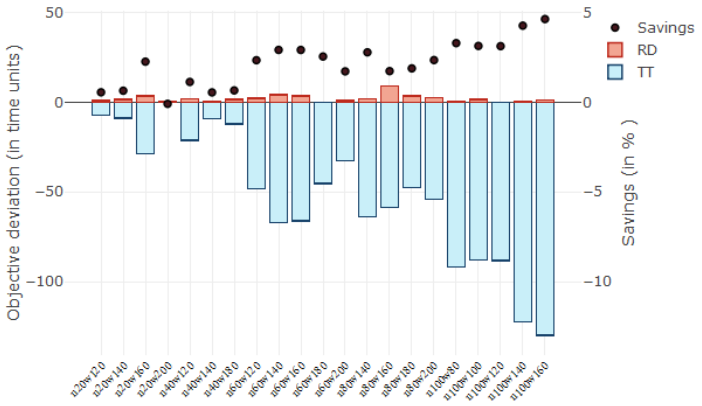

Figure 2: Savings and average TT and RD variations per class, for WOF versus RD only (GHLS instances)

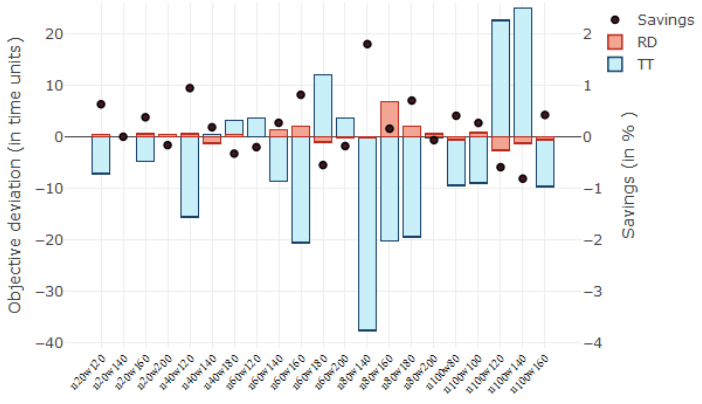

Figure 3: Savings and average TT and RD variations per class, for WOF versus TSA (GHLS instances)

5.1.3 Sensitivity analysis

Results for different parameter settings are presented in Table 1. For each setting, we report average results computed over the solutions obtained over all GHLS instances. In this table, the first two columns specify the tolerance ϵ used in Step 2 of the TSA and the ratio of γ^{TT} to γ^{RD} used in the WOF model. The next four columns provide the average WOF value (z^{WOF}) of the solutions when computed with $\gamma^{TT} = 8$ and $\gamma^{RD} = 20$, the TT and RD of the computed solutions as well as the average computational time in seconds. Then, columns Sav-TT and Sav-RD report the average savings in percentage with respect to the solutions computed when minimizing TT only and RD only, respectively. These savings are calculated by considering the value z^{WOF} for all solutions. Finally, column TolV 5/10 indicates for how many instances (out of 105), the computed solution would violate the tolerance ϵ if it was set to 5 or 10.

Table 1: Comparing TSA and WOF with different parameter settings (GHLS instances)

Method	ϵ	γ^{TT}/γ^{RD}	z^{WOF}	TT	RD	Time (s)	Sav-TT (%)	Sav-RD (%)	TolV5/10
WOF	-	1/100	14,629.2	479.4	539.7	42.2	2.21	2.21	1/0
WOF	-	1/20	14,605.2	475.9	539.9	46.8	2.37	2.37	3/0
WOF	-	5/20	14,581.6	470.2	541.0	52.3	2.53	2.53	6/4
WOF	-	8/20	14,582,8	469.1	541.5	56.5	2.53	2.52	12/6
WOF	-	13/20	14,594.8	466.1	543.3	58.5	2.45	2.44	23/13
WOF	-	20/20	14,598.0	463.5	544.5	59.0	2.42	2.42	31/18
TSA	5	-	14,609.2	473.4	541.1	53.9	2.35	2.34	0/0
TSA	10	-	14,625.6	470.2	543.2	57.8	2.24	2.24	39/0

Minimizing the WOF should provide the best z^{WOF} value when $\gamma^{TT} = 8$ and $\gamma^{RD} = 20$. Here, the best parameter setting is rather $\gamma^{TT} = 5$ and $\gamma^{RD} = 20$ by a slight margin over $\gamma^{TT} = 8$ and $\gamma^{RD} = 20$. This is explained by the fact that not all computed solutions are optimal given that the solution process reached the time limit for some instances. For the WOF model, we observe that the average computational time increases with the ratio γ^{TT}/γ^{RD} that converges to one, indicating that it is difficult to obtain a perfect tradeoff between TT and RD.

As expected, increasing ϵ from 5 to 10 in the TSA produces solutions with less TT but larger RD. It also increases the average computational time given that the solution space is enlarged in Step 2.

Finally, note that even if a slight average RD increase of 0.4 is observed between the solutions computed by the TSA with $\epsilon = 5$ and those obtained by the WOF model with $\gamma^{TT} = 8$ and $\gamma^{RD} = 20$, the RD has a greater variance in the latter solutions. Indeed, 12 of 105 solutions would have violated the tolerance $\epsilon = 5$, and 6 would have violated the limit $\epsilon = 10$. Logically, the closer the ratio γ^{TT}/γ^{RD} is to 1, the more the tolerances are violated, since there is little interest in saving RD rather than TT.

5.2 Ascheuer et al. instances

These 50 instances, hereafter the AFG instances, were presented in [4]. The number of nodes varies from 12 to 233. These instances are much harder to solve when minimizing RD instead of TT. Indeed, 9 of 50 instances could not be solved by CP to proven optimality within 5 min.

5.2.1 WOF results

Compared to the TT solutions, the WOF solutions give savings of -0.16% to 9.90%, for an overall saving of 0.98%. In most cases, the price of the route is reduced (there is only one increase of more than 0.1%), as Figure 4 shows. The comparative results with the RD minimization are similar to those for the first benchmark and are presented in Figure 5.

5.2.2 TSA results

Figure 6 presents a comparison of the solutions obtained with the WOF model and the TSA. Neither method clearly outperforms the other. In three instances WOF gives a saving of more than 1.5%, but TSA is slightly better on average (by 1.03% overall).

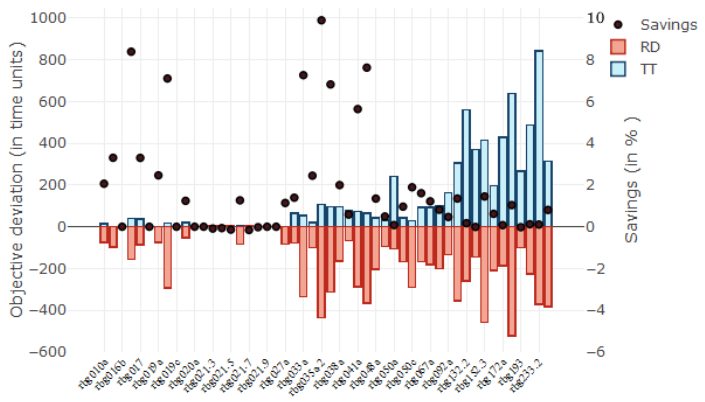

Figure 4: Savings, and TT and RD variations per instance, for WOF versus TT only (AFG instances)

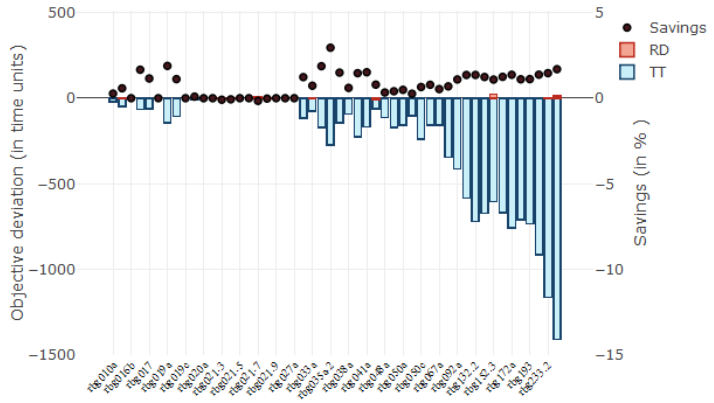

Figure 5: Savings, and TT and RD variations per instance, for WOF versus RD only (AFG instances)

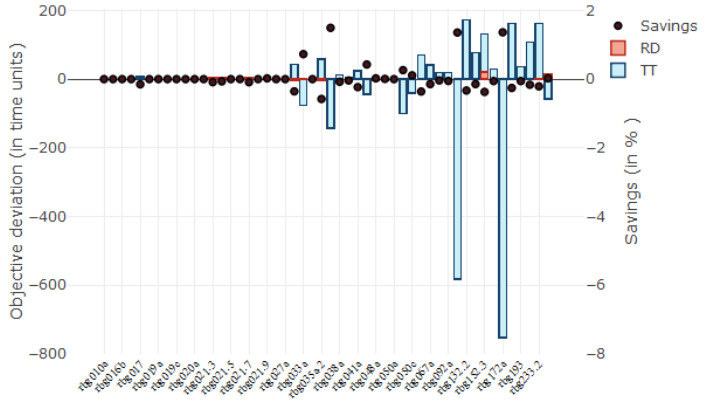

Figure 6: Savings, and TT and RD variations per instance, for WOF versus TSA (AFG instances)

6 Results for instances with sparse TWs

We have studied two instances, labelled a and b, provided by our industrial partner, Giro. They commercialize the optimization software GeoRoute which is used by some of the largest postal organizations worldwide. These instances have 475 (a) and 458 (b) nodes and just a few TWs (resp. 6 and 8), which is an extreme case of flexibility. We also generated instances with 200 and 300 nodes, selected randomly from those available, retaining all of the nodes with TWs. We refer to the instances as rd_XXX_Yg where XXX is the number of nodes selected from the original instance Full_g and Y is an identifier. For our tests, we used the same computer as described above, still with a single thread. In addition to CP-Optimizer, the disaggregation phase using TBF relied on the MIP solver Cplex, version 12.7.1.0.

Because there are few TWs, minimizing the RD also gives a good TT, as it is almost the only way to reduce the tour duration (there is little idle time). The reverse is not true since a long waiting time will not be penalized at all. Thus, for these instances (TW density below 5%) a multiobjective function is not really necessary. Given the size of the test instances, we want to show in this section the efficiency of the clustering approach described in Section 4.

In Figures 7 and 8, the lines represent the evolution of the RD of the best solution found with the direct CP model (i.e., without clustering) introduced in Section 2, minimizing the RD. On average, this is better than minimizing the TT or the WOF, or using the TSA for the same time without clustering. The points (resp. crosses or circles) represent the RD value of the solution computed by the clustering approach (resp. with the pure CP and CP-TBF disaggregation methods). We impose a time limit of 20 s for solving the clustered instance and also of 20 s for the disaggregation phase. The additional time mainly comes from the clustering step or the building of the different models. The number of nodes in the clustered instances varies between 67 and 96 in these instances. For these results, we set $\theta = 1$ in the CP-TBF disaggregation phase.

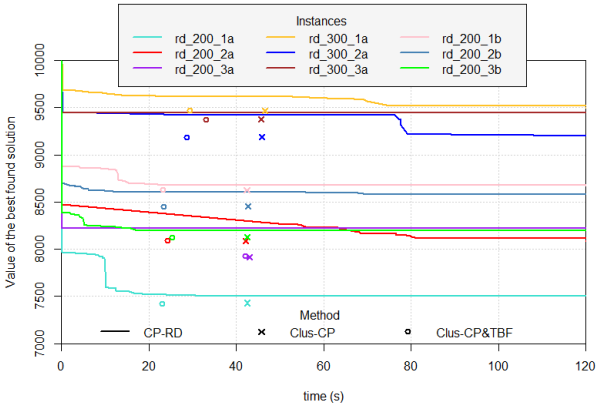

Figure 7: Minimizing RD for sparse-TW instances (tour duration up to 10,000 time units)

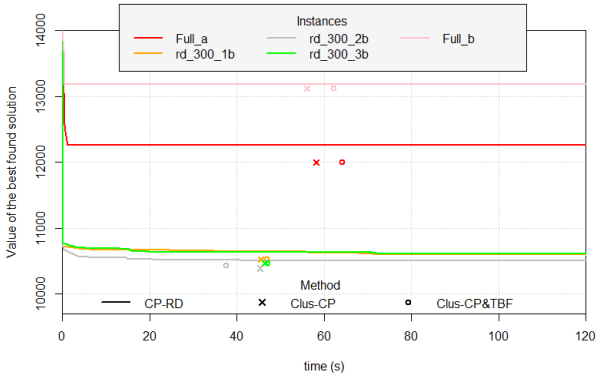

Figure 8: Minimizing RD for sparse-TW instances (tour duration above 10,000 time units)

On average, compared to the direct CP method, the clustering method gives a reduction in the RD of 1.11% (for pure CP) and 1.08% (for TBF-CP), in respectively 74 s and 84 s less than the 120 s allowed for the CP method. In addition to the time savings for some instances, the TBF method ensures that there is no better feasible disaggregation given the computed clusters and clustered sequence.

Table 2 shows the results of the four direct methods, all computed with CP models. RD is the most efficient, but even if we pick the best method for each instance, the clustering approach still reduces the RD by 0.84% on average, in less than half the time.

We conducted further tests with two different values of θ for which we do not report detailed results. For $\theta = 0$, which provides a rough way of disaggregating the clustered instance, the results are almost equivalent to those of the other methods, and they are obtained in less time. For $\theta = 2$, there is much more flexibility in the disaggregation phase, but it is not possible to obtain good results quickly.

Table 2: RD percentage reduction compared to the direct methods (TT, RD, TSA, and WOF) and to the best of the four

Instance	TT	RD	TSA	WOF	Best of 4
rd_200_1a	2.41%	1.05%	1.05%	2.15%	1.05%
rd_200_2a	0.01%	0.43%	1.79%	4.72%	0.01%
rd_200_3a	2.70%	3.89%	3.89%	3.89%	2.70%
rd_300_1a	1.65%	0.53%	1.54%	-0.24%	-0.24%
rd_300_2a	0.78%	0.30%	2.58%	2.73%	0.30%
rd_300_3a	0.36%	0.71%	0.71%	0.71%	0.36%
Full_a	1.93%	2.13%	1.76%	2.06%	1.76%
rd_200_1b	1.69%	0.65%	0.65%	1.49%	0.65%
rd_200_2b	1.75%	1.55%	1.73%	1.66%	1.55%
rd_200_3b	1.12%	0.86%	0.89%	1.35%	0.86%
rd_300_1b	1.89%	0.75%	0.95%	1.29%	0.75%
rd_300_2b	0.79%	1.17%	1.17%	1.40%	0.79%
rd_300_3b	3.14%	1.48%	1.59%	1.80%	1.48%
Full_b	0.05%	0.51%	0.38%	0.51%	0.05%
Avg.	1.40%	1.11%	1.41%	1.70%	0.84%

Conclusion

In this paper we highlighted the particularities of the well-known traveling salesman problem with time windows that arise in the context of postal operations, namely, the importance of minimizing not only the total travel time, or distance, but also the total route duration. To tackle this issue, we investigated different solution mechanisms, such as weighted combination of objectives and a hierarchical three-step approach.

We also consider large-scale instances where very few customers have time windows. By clustering the unconstrained customers, solving the reduced problem, and then sequencing the customers inside clustered nodes, we were able to achieve better solutions than a direct approach and in shorter computational times.

As a future work, we will tackle the postal territory design problem, considering that this design is a strategic decision which is taken approximately once per year perhaps, while optimizing the TSPTW occurs at an operational, daily, decision level.

Appendix

The TBF [7] is a MIP model which gathers several variables of a time-indexed formulation into time buckets, to reduce the combinatorial nature of the problem. Each bucket $b = [\underline{b}, \bar{b}]$ represents a portion of the time horizon. Let $\mathcal{B}$ be the set of all buckets used. With the directed graph $G = (V, A)$ introduced in Section 2, the TBF has three types of variables:

- $x_{ij} = 1$ if arc $(i, j) \in A$ is used; 0 otherwise.
- $z_i^b = 1$ if node i is visited in bucket b; 0 otherwise.
- $y_{ij}^b = 1$ if arc $(i, j) \in A$ is used and node i is visited in bucket b; 0 otherwise.

We define three subsets:

- $B_i \subseteq \mathcal{B}$ is the subset of buckets allowed for node i.
- $V^+(i) \subseteq V \backslash \{p\}$ (resp. $V^-(i) \subseteq V \backslash \{q\}$) is the subset of possible successors (resp. predecessors) of i. It can be tightly defined, as presented in Section 3.1.
- $I_k(i, b)$ is the subset of B_k representing the potential starting buckets of node k if arc (k, i) is used and the visit to node i started in bucket b. We have

$$I_k(i, b_l) = \{b \in B_k : \bar{b}_{l-1} < \underline{b} + t_{ki} \leq \bar{b}_l\}$$

where $b_0 = -\infty$ is assumed.

Given this notation the time bucket relaxation (TBR) for the TSPTW is

$$\min \sum_{(i,j)\in A} t_{ij}x_{ij} \tag{30}$$

subject to:

$$\sum_{b\in B_i} z_i^b = 1, \qquad \forall i \in V \tag{31}$$

$$\sum_{j\in V^+(i)} y_{ij}^b = z_i^b, \qquad \forall i \in V\backslash\{q\}, \quad \forall b \in B_i \tag{32}$$

$$\sum_{k\in V^-(i)} \sum_{\beta\in I_k(i,b)} y_{ki}^\beta = z_i^b, \qquad \forall i \in V\backslash\{p\}, \quad \forall b \in B_i \tag{33}$$

$$\sum_{b\in B_i} y_{ij}^b = x_{ij}, \qquad \forall (i,j) \in A \tag{34}$$

$$x_{ij}, y_{ij}^b, z_i^b \in \{0,1\}, \qquad \forall i \in V, \quad \forall (i,j) \in A, \quad \forall b \in B_i. \tag{35}$$

The objective function (30) minimizes the TT. Constraints (31) ensure that each service point is visited in a single bucket. Constraints (32) and (33) link the variables y and z, while guaranteeing that a valid bucket is used via the set $I_k(i,b)$. Constraints (34) link x and y.

Model (30)–(35), as its name implies, is a relaxation of the TSPTW. The feasibility checks of the model are made as if every time was at the beginning of a bucket, so as shown in Figure 9, subtour elimination constraints (SECs) and infeasible path cuts (IPCs) should be added to obtain an exact formulation, the so-called TBF.

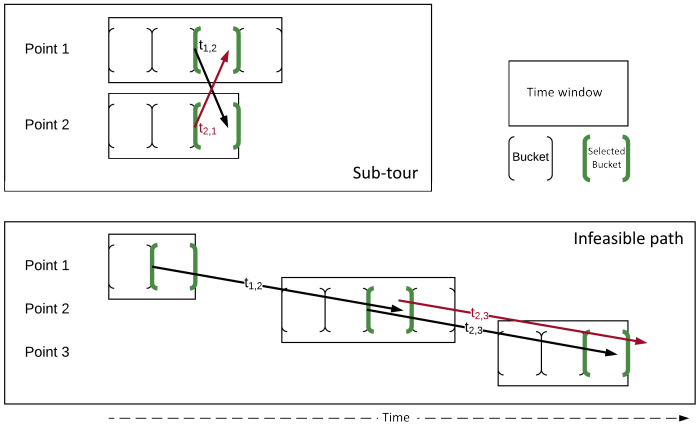

Figure 9: Subtour and infeasible path examples

If the TT between two nodes is shorter than the width of a time bucket, subtours may occur. In dense areas or when the bucket width is large, the subtours may contain more than two nodes. In Figure 9, the infeasible path $(1[b_{1,2}], 2[b_{2,3}], 3[b_{3,3}])$, using respectively the second, third, and third buckets of nodes $1, 2$, and 3, is allowed by the TBR since the feasibility depends on $I_k(i,b)$, which selects buckets in B_k as long as a "time index" of the buckets matches with one time index of b ($\in B_i$).

For every subtour detected, let $S \subset V\backslash\{p,q\}$ be the associated nodes. The following SEC must be added:

$$\sum_{(i,j)\in\delta(S)} x_{ij} \geq 1 \tag{36}$$

where $\delta(S)$ is the subset of arcs (i,j) of A such that $i \in S$ and $j \in V\backslash\{S\}$.

For every infeasible path $\mathcal{P}$ (which would violate some TWs), we add the IPC:

$$\sum_{(i,j)\in\mathcal{P}} x_{ij} \leq |\mathcal{P}| - 1. \tag{37}$$

The narrower the buckets, the more accurate the model and the fewer SECs and IPCs have to be added while solving the MIP. However, increasing the number of variables increases the combinatorial nature of the problem. With wider buckets, a node may not be assigned to the right bucket. However, if the solution remains feasible, we can retain it, because only the sequence matters. For this reason, the TBF may greatly reduce the TT.

We can try to capture the RD information of the sequence found, but it is not accurate, since we know only in which bucket the route finishes. Intuitively, the objective function may be modified to min z where z is defined by the following constraint, leading to an overestimation of the real RD objective function:

$$z \geq \sum_{b\in B_q} \bar{b} z_q^b - \sum_{b\in B_p} \underline{b} z_p^b. \tag{38}$$

However, it is possible to be more accurate, especially when the waiting time is low, by replacing (38) by:

$$z \geq \sum_{b\in B_q} \underline{b} z_q^b - \sum_{b\in B_p} \bar{b} z_p^b \tag{39}$$

$$z \geq \sum_{(i,j)\in A} t_{ij} x_{ij}. \tag{40}$$

In both cases, the buckets have to be carefully assigned, and this requires the addition of some constraints; see [7]. Moreover, to obtain the exact RD, we must refine the buckets to increase the accuracy in terms of the departure time from and the arrival time at the depot. These two values greatly affect the computational time, and hence we do not use the TBF to minimize RD. When an upper bound on RD is imposed like in Section 4.3.2, it is enforced through the addition of IPCs.

In the TBF, the bucket generation may be key to the performance. We generate the time buckets on a fixed base L so $b_0 = [0, L-1], b_1 = [L, 2L-1]$, and so on. If necessary, we replace the first and last bucket of each service point i by more accurate ones that match the TW. If $L = 10$ and for some i, $[R_i, D_i] = [27, 44]$, B_i is defined by $\{[27, 29], [30, 39], [40, 44]\}$ instead of $\{[20, 29], [30, 39], [40, 49]\}$. This allows us to discard some buckets in $I_k(i, b)$ that could have been considered with the original buckets.

The TBR is solved using Cplex 12.7.1.0, and SECs and IPCs are added via callbacks during the process. We use the lazycallback technology, i.e., we check for subtours and infeasible paths at every integer solution and then add the corresponding constraints (36) and (37) to the model.

References

[1] ford.ca. http://www.ford.ca/commercial-trucks/transit-connect-cargo-van/2017/features/capability/. Accessed: 2017-06-01.

[2] indeed.com. https://emplois.ca.indeed.com/cmp/Canada-Post/salaries. Accessed: 2017-06-01.

[3] Natural resources canada : Retail fuel prices by province. http://www2.nrcan.gc.ca/eneene/sources/pripri/price_map_e.cfm. Accessed: 2017-11-13.

[4] Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. Mathematical Programming, 90(3):475–506, May 2001.

[5] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New State-Space Relaxations for Solving the Traveling Salesman Problem with Time Windows. INFORMS Journal on Computing, 24(3):356–371, 2012.

[6] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. State-space relaxation procedures for the computation of bounds to routing problems. Networks, 11(2):145–164, 1981.

[7] Sanjeeb Dash, Oktay Günlük, Andrea Lodi, and Andrea Tramontani. A time bucket formulation for the traveling salesman problem with time windows. INFORMS Journal on Computing, 24(1):132–147, 2012.

[8] Kalyanmoy Deb. Multi-objective optimization. In Edmund K. Burke and Graham Kendall, editors, Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, pages 403–449. Springer US, Boston, MA, 2014.

[9] Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. Operations Research, 43(2):367–371, April 1995.

[10] Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. Operations Research, 46(3):330–335, March 1998.

[11] Manuel López-Ibáñez, Christian Blum, Jeffrey W. Ohlmann, and Barrett W. Thomas. The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. Applied Soft Computing, 13(9):3806–3815, 2013.

[12] George Mavrotas. Effective implementation of the ϵ-constraint method in multi-objective mathematical programming problems. Applied Mathematics and Computation, 213(2):455–465, July 2009.

[13] Boland Natashia, Hewitt Mike, Vu Duc Minh, and Savelsbergh Martin W.P. Solving the Traveling Salesman Problem with Time Windows Through Dynamically Generated Time-Expanded Networks, pages 254–262. Springer International Publishing, Cham, 2017.

[14] Martin W. P. Savelsbergh. Local search in routing problems with time windows. Annals of Operations Research, 4(1):285–305, 1985.

[15] Martin W.P. Savelsbergh. The Vehicle Routing Problem with Time Windows : minimizing route duration. ORSA Journal on Computing, 4(2):146–154, 1992.