ARTICLE TEMPLATE

The Traveling Salesman Problem with Time Windows in Postal Services

Alexis Bretin[a,b,c],Guy Desaulniers [a,c] and Louis-Martin Rousseau[a,b]

[a]École Polytechnique de Montréal, Montréal, CANADA; [b]CIRRELT, Montréal, CANADA;[c]GERAD, Montréal, CANADA

ARTICLE HISTORY
Compiled September 25, 2019

ABSTRACT
This paper focuses on a variant of the traveling salesman problem with time windows (TSPTW) that arises in postal services and parcel deliveries. It differs from the classical TSPTW as follows. First, route duration is a significant concern, as human costs largely exceed vehicle costs, emphasizing the importance of reducing waiting time. Second, customers are divided in two categories: commercial ones with time windows and private customers without time window restrictions, making the NP-hard TSPTW even harder to solve. To support the first statement, we present a multi-objective approach based on a constraint programming formulation of the problem which allows to balance the optimization of both human and material resources. To address the issue of large-scale sparsely time-constrained instances and reduce the size of the large real-world instances, we also introduce a *cluster/solve/sequence* approach that relies on a mathematical programming formulation to *sequence* the customer visits in the final step. This decomposition technique allows to produce high-quality solutions (with an average optimality gap of 1.31%) for industrial problems in less than one minute of computation time.

KEYWORDS
Traveling salesman problem with Time windows; Postal services; Route duration; Bi-objective; Low time-window density; Clustering heuristic.

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 position our work in the context of a parcel-delivery problem with time windows (TWs) which can be seen as a variant of the vehicle routing problem with TWs (VRPTW). The TWs are designed to synchronize deliveries and recipients. For large-scale instances involving 10 to 20 territories and up to almost 500 deliveries in a territory, this problem can be solved via a greedy cluster-first route-second (CFRS) procedure. The clustering phase determines

Alexis Bretin. Email: alexis.bretin@polymtl.ca
Guy Desaulniers. Email: guy.desaulniers@gerad.ca
Louis-Martin Rousseau. Email: louis-martin.rousseau@cirrelt.net

well-designed territories according to the specific characteristics of the instance, and the routing phase designs the routes. The two phases are carried out alternately in the following greedy fashion. First, a cluster containing a number of service points (customers) whose total demand does not exceed the capacity of a vehicle 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.

In this paper, we focus on the routing phase of this CFRS procedure, which solves a variant of the traveling salesman problem with TWs (TSPTW). The classical TSPTW consists of finding a route that starts and ends at a depot, visits every customer once within a pre-specified time interval called a TW, and minimizes the total routing cost. As discussed below, the postal services variant of the TSPTW differs from the classical one by considering a bi-objective cost function and no TW for a large proportion of the customers.

In most studies on the TSPTW, the routing cost is proportional to the total distance traveled along the route or the closely related travel time (TT), i.e., the time spent driving along the route, but waiting before the opening of a customer TW is usually not penalized. In postal services, focusing on TT may be costly because the optimal 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. Therefore, beside minimizing TT, the objective function also considers minimizing the route duration (RD), i.e., the difference between the end time and the start time of the route or, equivalently, the sum of the TT, the service time and the waiting time. Note that RD can be assimilated to the makespan, which is defined in scheduling problems as the span of time necessary to complete all the tasks. However, it differs from the completion time of the final task (return to the depot) because, when there are TWs, starting the route at time 0 may not be optimal.

Two observations support the consideration of RD in the objective function. First, in a CFRS approach to the VRPTW, minimizing the RD may make it possible to visit more customers 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. Note that reducing the RD may also be relevant in other applications. For instance, 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.

Second, the average wage for a Canada Post delivery driver is around \$20/h (see Indeed)[1]. For a vehicle such as a Ford Transit, the fuel consumption is around 12 L/100km (see Ford)[2]. If the average speed is between 30 and 50 km/h the approximate cost of one hour of driving is $40 \times 12/100 \times 1.2 = \$5.76$ where 1.2 is the average price of a liter of gasoline in Canada (see Natural Resources 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 minutes to save 1 minute of

[1] https://emplois.ca.indeed.com/cmp/Canada-Post/salaries

[2] http://www.ford.ca/commercial-trucks/transit-connect-cargo-van/2017/features/capability

[3] http://www2.nrcan.gc.ca/eneene/sources/pripri/price_map_e.cfm

RD, and the idle time is reduced by 3 minutes. 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 relates to the vehicle cost whereas the RD to the human cost of the route. In the following, we denote by γ^{TT} and γ^{RD}, the costs per time unit when the vehicle is moving and when the driver is working, respectively.

To further illustrate why minimizing RD can be interesting, consider the example presented in Figure 1, where D represents the depot, and C_l, $l \in \{1, \ldots, 6\}$, the customers. The intervals near the customer nodes are their TWs, whereas the numbers on the arcs are the arc TTs. The service times are assumed to be all equal to 0. The blue (dashed) path is the optimal route when only minimizing the TT. Its TT is $z_{TT} = 15$, but its RD is $z_{RD} = 19$ because the driver cannot leave the depot later than time 2 and return to it before time 21. This path induces a minimal waiting of 4 time units at C_2. The red (solid) path is the optimal solution when only minimizing the RD. This route also leaves depot D at time 2 but instead of visiting C_2 right after C_1 and waiting 4 time units, it visits C_3 between C_1 and C_2. This detour generates a longer TT but decreases the RD by 2. Its TT and RD are equal to $z'_{TT} = 17$ and $z'_{RD} = 17$, respectively. Consequently, the red path allows the replacement of 4 time units of waiting by 2 extra time units of traveling. Using weights equal to the above cost estimations ($\gamma^{TT} = 8$ and $\gamma^{RD} = 20$), these two solutions can be compared with respect to the bi-objective cost function $\gamma^{TT} \times TT + \gamma^{RD} \times RD$. The TT-optimal solution has a value of $15 \times 8 + 19 \times 20 = 500$, whereas the RD-optimal solution a value of $17 \times 8 + 17 \times 20 = 476$, which yields a saving of 4.8%.

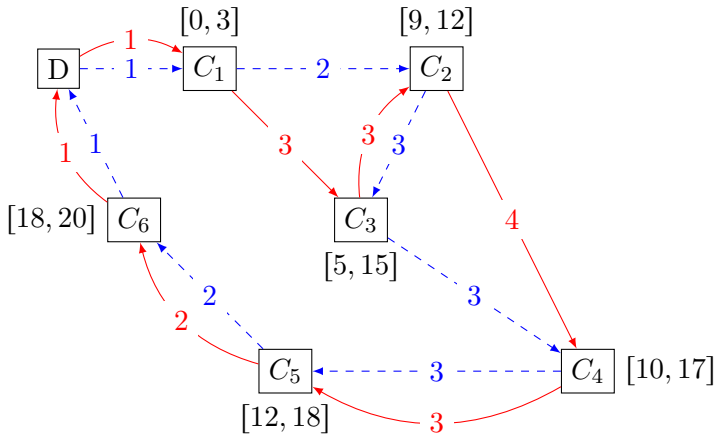

Figure 1. Two different solutions: the blue (dashed) route minimizes TT, the red (solid) one minimizes RD

Note, however, that focusing only on RD is not ideal in some cases because extra TT may be incurred when waiting is unavoidable. To see this, let us consider again the example of Figure 1 but assuming that the TW at C_6 is $[20, 22]$ instead of $[18, 20]$. In this case, the red path will have to wait 2 time units at C_6, increasing its RD to $z'_{RD} = 19$ and keeping $z'_{TT} = 17$. Given that the statistics for the blue path do not change ($z_{RD} = 19$ and $z_{TT} = 15$), the blue path becomes as good as the red path with respect to RD but remains better with respect to TT and should, therefore, be selected to reduce vehicle costs. This example justifies why the two criteria needs to be considered in the objective function.

TWs are particularly useful for companies, where there are dedicated employees

to receive and process deliveries. Furthermore, they often receive express deliveries that have tight delivery deadlines, inducing TWs for the postal companies. On the other hand, 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 percentage is called TW density and, when it is not high enough, it makes the NP-hard TSPTW even harder to solve, as less preprocessing can be done to reduce the solution space.

The classical TSPTW has been extensively studied. Savelsbergh (1985) showed that even only finding a feasible solution to it is NP-hard. From branch-and-bound algorithm using a state-space-relaxation approach to compute the lower bounds (Christofides, Mingozzi, & Toth, 1981), to dynamically generated time-expanded networks (Boland, Hewitt, Vu, & Savelsbergh, 2017), many exact solution algorithms for the TSPTW have been developed. To the best of our knowledge, the most efficient one is that of Baldacci, Mingozzi, and Roberti (2012). 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.

Some heuristics have also been developed to solve the TSPTW. Gendreau, Hertz, Laporte, and Stan (1998) proposed an insertion heuristic, which is an adaptation of the GENIUS heuristic introduced by Gendreau, Hertz, and Laporte (1992) for the case without TWs. Calvo (2000) developed a heuristic that combines the solution of an assignment problem, a greedy insertion procedure and local search. Later, Ohlmann and Thomas (2007) designed a compressed-annealing heuristic, a simulated annealing variant, López-Ibáñez and Blum (2010) combined ant colony optimization with beam search, a heuristic that approximates a branch-and-bound method originally developed for scheduling problems, and Ferreira da Silva and Urrutia (2010) introduced a variable neighborhood search heuristic.

These papers deal with minimizing TT. Savelsbergh (1992) was the first to address RD minimization in the TSPTW, by extending the local search heuristic based on edge exchanges that he developed a few years before (Savelsbergh, 1990). Also he did not fix the departure time from the depot, and introduced the concept of forward slack time to help reducing waiting time. According to López-Ibáñez, Blum, Ohlmann, and Thomas (2013) who minimize the completion time of the route, *i.e.*, with a fixed departure time from the depot, this variant of the problem can generally be solved more quickly than minimizing TT. This is confirmed by the computational results obtained by Kara and Derya (2015) using mixed-integer programming (MIP) formulations. In these solution approaches, the focus is only on minimizing RD or the completion time, and the TT is let aside.

Note that the benchmark instances used to evaluate these solution methods have generally 100% TW density, enabling efficient preprocessing to reduce the TWs' width, especially when they are relatively narrow initially. This preprocessing relies mainly on some efficient techniques proposed by Desrochers, Desrosiers, and Solomon (1992) and on advanced arc elimination criteria introduced by Langevin, Desrochers, Desrosiers, Gélinas, and Soumis (1993).

To the best of our knowledge, there are no works in the literature that address the TSPTW variant introduced in this paper. Therefore, the contribution of this paper is to devise an effective solution algorithm to tackle large-scale, bi-objective TSPTW instances with low TW density that arise in the postal services. This clustering heuristic exploits the strength of constraint programming (CP) for minimizing the RD and of

MIP for minimizing the TT. To enable the solution of large instances within a tight computational budget, it proceeds in steps: it builds a clustered problem, solves it, and disaggregates the clustered solution. To our knowledge, we are the first to handle the combined TT and RD objective function. Therefore, to highlight the possible trade-offs between RD and TT in the solutions computed for different objective functions (minimizing TT only, RD only, or a linear combination of both), we first report computational results obtained on small and medium-sized TSPTW instances taken from the literature and for which all customers have a TW. Finally, to assess the performance of the proposed clustering heuristic, we present computational results obtained on real industrial instances provided by our research partner.

The rest of this paper is structured as follows. In Section 2, we introduce a CP model for the TSPTW where the objective function can be easily adapted to minimize RD or TT or a combination of both. We also summarize a MIP model (fully described in the Appendix A) previously introduced in the literature. In Section 3, we describe our clustering heuristic. Computational results on the benchmark instances are reported in Section 4 and those on the industrial instances in Section 5. Finally, a short conclusion is drawn in Section 6.

2. Problem formulations

Consider a directed graph $G = (V, A)$ where V contains the nodes, including p and q representing the depot at the beginning and the end of the tour respectively, 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. Some nodes $i \in V$ must be visited within a TW $[R_i, D_i]$, where R_i is the release time and D_i the deadline of node i. For the others, we assume that they must also be visited within an unconstraining TW $[R_i, D_i]$, where $R_i = 0$ and D_i is equal to a very large value. A service time 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, set A does not contain all the couples $(i, j) \in V^2$ but only a subset. Indeed, arc (i, j) exists only if $R_i + s_i + t_{ij} \leqslant D_j$, where t_{ij} is the traveling time along (i, j). Furthermore, some preprocessing can be applied to reduce the solution space, and thus the combinatorial nature of the problem. TWs may be tightened and node precedence lists built (Ascheuer, Fischetti, & Grötschel, 2001). For each node $i \in V$, the precedence list $V^+(i) = \{j \in V \backslash \{i\} \mid j \prec i\}$ gathers all the nodes j that have to be visited before i. Some arcs may be deleted based on the reduced TWs and the precedence relationships (Dash, Gunluk, Lodi, & Tramontani, 2012).

We first present a CP formulation before explaining the use of bi-objective programming to minimize TT and RD. Then, we summarize a MIP formulation called the time-bucket formulation (TBF).

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.

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

$$\begin{aligned}
\text{subject to:} \quad P_1 &= p && && (2)\\
P_{|V|} &= q && && (3)\\
P_l &\in V, && \forall l \in \{1,\dots,|V|\} && (4)\\
&\texttt{alldifferent}(P) && && (5)\\
T_{P_l} + s_{P_l} + t_{P_l P_{l+1}} &\leqslant T_{P_{l+1}} && \forall l \in \{1,\dots,|V|-1\} && (6)\\
T_{P_l} &\in [R_{P_l}, D_{P_l}], && \forall l \in \{1,\dots,|V|\} && (7)\\
T_j &\leqslant T_i, && \forall i \in V, j \in V^+(i). && (8)
\end{aligned}$$

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 (2) and (3), we may write the objective function as shown in (1). Constraint (5) is the well-known global constraint `alldifferent`, extensively detailed in van Hoeve (2001), which indicates that the service points visited for $l \in \{1,\dots,|V|\}$ must be pairwise different (*i.e.*, $P_l \neq P_k, \forall (l,k) \in \{1,\dots,|V|\}^2$ such that $l \neq k$). Combined with constraints (4), this ensures that every service point is visited exactly once. Constraints (6) impose sufficient time between two consecutive service point visits (P_l and P_{l+1}) to perform service at P_l and travel from P_l to P_{l+1}. Constraints (7) ensure that the TWs are respected, whereas constraints (8) enforce the precedence constraints identified by the preprocessing.

As mentioned in the introduction, 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

$$\min z = \sum_{l=1}^{|V|-1} t_{P_l P_{l+1}}. \tag{9}$$

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}. 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 Deb (2014). 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

$$\min \quad \gamma^{TT}\left(\sum_{l=1}^{|V|-1} t_{P_l P_{l+1}}\right) + \gamma^{RD}(T_q - T_p). \tag{10}$$

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. Moreover, the TT part of this WOF makes optimality harder to reach with this CP-model (as illustrated in Table B1 of Appendix B).

2.3. A time-bucket formulation

The TBF introduced by Dash et al. (2012) is a MIP model that partitions the TW associated with each node into time buckets and uses arc flow variables indexed by the time bucket containing the start traveling time along the arc. These time buckets provide an approximate modeling of the TWs that is a relaxation of these constraints and whose accuracy depends on the width of the buckets. Subtour elimination constraints (SECs) and infeasible path cuts (IPCs) are, thus, required to eliminate solutions that do not meet the TW restrictions and are not filtered out by the time bucket modeling. Compared to a time-indexed model, the main advantage of the TBF is a significant reduction in the number of variables when the buckets are sufficiently large. On the other hand, to get a good relaxation of the TW restrictions and avoid generating many SECs and IPCs, they should not be too large. Details on the TBF can be found in the Appendix A.

Dash et al. (2012) showed that the TSPTW can be efficiently solved by a branch-and-cut algorithm applied to the TBF when the objective aims at minimizing TT. As discussed in the Appendix A, it is more difficult to use the TBF to solve the TSPTW when the objective function consists of minimizing RD. Indeed, it might require a complete discretization of the TWs at the depot nodes p and q.

3. Solution algorithm for real-world instances

The CP models presented in Sections 2.1 and 2.2 can be solved using a commercial CP solver. Furthermore, the TBF discussed in Section 2.3 can be solved using a commercial-based branch-and-cut algorithm as described in the Appendix A. Preliminary computational tests have shown that the CP algorithm is efficient at finding, in short computational times, high-quality solutions for small and medium-sized instances when minimizing the RD (see Figure B1 in Appendix B). On the other hand, it struggles for large instances or for minimizing the TT. At the opposite, the TBF-based algorithm is much better at minimizing the TT than the RD. Its efficiency is, however, highly dependent on the tightness of the TWs. Finally, as for the CP algorithm, the TBF-based algorithm cannot solve large instances in relatively short computational times. Therefore, both algorithms cannot be used successfully to solve large-scale TSPTW instances with a bi-objective function and very few TWs. Computational results supporting these statements are presented in Appendix B.

As discussed in the introduction, priority is given to minimizing the RD over the TT, but minimizing the TT must not be neglected. In a context without TWs, minimizing the RD is equivalent to minimizing the TT because waiting is always avoidable. When

there are few 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. On the other hand, minimizing the TT does not necessarily yield a solution with a low RD because waiting is not penalized at all. This led us to develop, for large-scale postal services' TSPTW instances, a solution approach that does not consider directly a weighted sum of the two objective functions, but rather combines CP, which is really efficient to minimize the RD, with MIP, more likely to close the optimality gap while minimizing TT.

In this section, we present the proposed clustering heuristic for solving large-scale, real-world, postal services' TSPTW instances. This heuristic proceeds in three steps. First, it clusters some service points, among the ones without a TW, to reduce the size of the instance, yielding a so-called clustered problem that approximates the original one. Second, it solves this clustered problem. Finally, given the sequence of the service points in the computed tour, the clusters are disaggregated to derive a solution to the original instance. These steps are described in the following subsections.

3.1. Clustering

Traditional models use the TWs to reduce the solution space; we instead cluster some service points for real-world postal services' instances, as the TW density is sparse. 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 traveling time 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 without a TW. At each iteration i of the algorithm, the clustered problem 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: all customers with a TW is represented by itself ($N_i(j) = \{j\}, \forall j \in J \backslash \tilde{J}$); all customers must belong to one subset ($\bigcup_{j \in J} N_i(j) = J$); and a customer cannot belong to two different subsets ($N_i(j_1) \cap N_i(j_2) = \varnothing$, $\forall (j_1, j_2) \in J^2$ 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 $N_i(j) \neq \varnothing$. 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 $|N_i(j)| \leqslant 1$. 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 $(j_1, j_2) \in J^2$, $M_i(j_1, j_2) = \Delta$ if $j_1 = j_2$, $N_i(j_1) = \varnothing$ or $N_i(j_2) = \varnothing$, where Δ is a constant larger than the maximum distance between two nodes. Otherwise, $M_i(j_1, j_2)$ approximates the traveling time from j_1 to j_2, including the internal traveling

Algorithm 1 Pseudo-code of the clustering phase

1: Initialize N_0, S_0, ITT_0, and M_0
2: $(k_1, l_1) \leftarrow \text{argmin}_{(k,l)\in \tilde{J}^2} M_0(k, l)$
3: $d_1 \leftarrow M_0(k_1, l_1)$
4: $i \leftarrow 1$
5: **while** $d_i \leqslant d^{max}$ **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 $N_i(l_i) \leftarrow \varnothing$
10: $\quad\quad S_i(k_i) \leftarrow S_i(k_i) + S_i(l_i)$
11: $\quad\quad M_i(j, l_i) = M_i(l_i, j) = \Delta, \quad \forall j \in J$
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 $N_i(j) \leftarrow \varnothing, \quad \forall j \in \mathcal{C}_i\setminus\{v_i^1\}$
18: $\quad\quad S_i(v_i^1) \leftarrow S_i(k_i) + S_i(l_i)$
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: $\quad\quad\quad M_i(j, k_i) = M_i(k_i, j) = \Delta, \quad \forall j \in J$
22: $\quad\quad$ **if** $l_i \neq v_i^1$ **then**
23: $\quad\quad\quad M_i(j, l_i) = M_i(l_i, j) = \Delta, \quad \forall j \in J$
24: $\quad\quad$ **for** $j \in V\setminus\mathcal{C}_i$ such that $V_i(j) \neq \varnothing$ **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: $\quad (k_i, l_i) \leftarrow \text{argmin}_{(k,l)\in\tilde{J}^2} M_{i-1}(k, l)$
29: $\quad d_i \leftarrow M_{i-1}(k_i, l_i)$

time at j_1.

The algorithm starts by initializing these vectors and matrix (Step 1) as follows. For all $j \in J$, $N_0(j) = j$, $S_0(j) = s_j$, and $ITT_0(j) = 0$, which means that every node in N_0 represents a single customer, with its own service time, and no internal travel time, as no clustering has been applied yet. For all $(j_1, j_2) \in J^2$, $M_0(j_1, j_2) = \Delta$ if $j_1 = j_2$, to prevent the selection of (j, j) as a good candidate for clustering. Otherwise, $M_0(j_1, j_2) = t_{j_1 j_2}$. In Step 2, the algorithm identifies in matrix M_0 the pair of nodes (k_1, l_1) yielding the minimal traveling time, then stores it in d_1 (Step 3). It then sets the iteration counter i to 1 (Step 4). As long as d_i does not exceed a predetermined maximum time d^{max} (Step 5), the selected nodes k_i and l_i will be merged in iteration i, resulting in a node containing the customer nodes in $\mathcal{C}_i$, *i.e.,* the ones already included in $N_{i-1}(k_i)$ and $N_{i-1}(l_i)$ (Step 6). It starts by making copies of the vectors and matrix N_{i-1}, S_{i-1}, ITT_{i-1} and M_{i-1} (Step 7). These copies will be updated to define the clustered problem 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 (Steps 9 - 11). 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 (Steps 12-26). Given that, in the current clustered problem, 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) and none of the customers in $\mathcal{C}_i$ has a TW, 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 (Step 26), 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 $N_i(j)$. But, 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 problem 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 anymore 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 traveling time matrix. To avoid this, we also tested two other criterion. The first is to stop clustering when the number of nodes in the clustered problem 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 n^{max} on each subset $N_i(j)$, $j \in J$, is imposed.

3.2. Solution of the clustered problem

After the clustering, we solve the clustered problem using our CP model (Section 2.1) with the goal of minimizing RD. Even if the clustered problem is smaller than the original one, there are still some TWs, and the problem remains complex. Nevertheless, for clustered problem instances with around 100 nodes or fewer, high-quality solutions can be found in a reasonable time.

3.3. Disaggregation

After solving the clustered problem to find a clustered solution denoted by *Seq*, we compute a solution to the original problem, respecting the sequence of *Seq* to a certain extent. This solution is found by solving the original problem subject to additional precedence constraints derived from *Seq* and presented below. Two algorithms, described afterwards, can be used to compute this solution.

3.3.1. 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 $\theta \geqslant 1$ 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:

$$a \prec b, \quad \forall i \in \{1, \ldots, n^K - \theta\}, \quad \forall a \in \mathcal{C}_{K_i}, \quad \forall b \in \mathcal{C}_{K_{i+\theta}} \tag{11}$$

  where, as in Section 2, $a \prec b$ 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:

$$b \prec W_j, \quad \forall\, W_j \in W \mid j^- - \theta \geqslant 1, \quad \forall b \in \mathcal{C}_{K_{(j^- - \theta)}} \tag{12}$$

$$a \succ W_j, \quad \forall\, W_j \in W \mid j^+ + \theta \leqslant n^W, \quad \forall a \in \mathcal{C}_{K_{(j^+ + \theta)}} \tag{13}$$

  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:

$$a \prec b, \quad \forall a \in \mathcal{C}_{\hat{A}}, \quad \forall b \in \mathcal{C}_{\hat{B}} \tag{14}$$

$$b \prec c, \quad \forall b \in \mathcal{C}_{\hat{B}}, \quad \forall c \in \mathcal{C}_{\hat{C}} \tag{15}$$

$$c \prec d, \quad \forall c \in \mathcal{C}_{\hat{C}}, \quad \forall d \in \mathcal{C}_{\hat{D}} \tag{16}$$

for the clustered nodes, and

$$W_1 \succ a, \quad \forall a \in \mathcal{C}_{\hat{A}} \tag{17}$$
$$W_1 \prec d, \quad \forall d \in \mathcal{C}_{\hat{D}} \tag{18}$$
$$W_2 \succ b, \quad \forall b \in \mathcal{C}_{\hat{B}} \tag{19}$$

for the nodes with TWs.

For more flexibility, θ can be increased. In our example, with $\theta = 2$, (14)–(16) are replaced by

$$a \prec c, \quad \forall a \in \mathcal{C}_{\hat{A}}, \quad \forall c \in \mathcal{C}_{\hat{C}} \tag{20}$$
$$b \prec d, \quad \forall b \in \mathcal{C}_{\hat{B}}, \quad \forall d \in \mathcal{C}_{\hat{D}} \tag{21}$$

and (17)–(19) become

$$W_2 \succ a, \quad \forall a \in \mathcal{C}_{\hat{A}}. \tag{22}$$

These additional precedence constraints increase the precedence lists $V^+(i)$, $i \in V$, defined in Section 2. They are, therefore, imposed in the CP model through constraints (8) and in the TBF by eliminating arcs in set A. 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.

Taking into account the precedence constraints, the problem can be solved using one of the two methods described in Sections 3.3.2 and 3.3.3.

3.3.2. Pure CP disaggregation algorithm

To compute a final disaggregated route, we can resort to a pure CP algorithm which simply solves the CP-WOF model augmented with all precedence constraints stated above. To speed up this algorithm, we tighten the TWs on the depot nodes p and q using the upper bound on the optimal RD provided by the value $\bar{z}_{RD}$ of the solution found for the clustered problem. More precisely, the TW at node p can be replaced by $[\max\{R_p, R_q - \bar{z}_{RD}\}, D_p]$ and that at node q by $[R_q, \min\{D_p + \bar{z}_{RD}, D_q\}]$, where we assume here that $[R_p, D_p]$ and $[R_q, D_q]$ are the TWs resulting from the preprocessing procedure mentioned in Section 2. Obviously, if one of these TWs is reduced in this way, preprocessing can be applied again.

3.3.3. Mixed CP and TBF disaggregation algorithm

Given that the TBF-based branch-and-cut algorithm is not efficient at minimizing the RD while the CP algorithm is, we propose an alternative disaggregation algorithm that combines both algorithms. This algorithm is as follows.

Step Dis-1: Minimize RD via CP to obtain a good upper bound for Step Dis-2.
Step Dis-2: Minimize TT via the TBF after imposing this upper bound on the RD.
Step Dis-3: Minimize RD via CP subject to the sequence computed in Step Dis-3.

In Step Dis-1, the CP algorithm is ran for only 1 second to solve the CP-model with the additional precedence constraints derived from the clustered solution found and by setting $\theta = 1$. Indeed, preliminary experiments (see Appendix B) have shown that

this algorithm finds very rapidly a high-quality solution but struggles to improve it afterwards or prove its optimality. Moreover, increasing the value of θ makes it difficult to compute a good-quality solution very rapidly. In Step Dis-2, the TBF also includes the precedence constraints arising from the clustered solution. However, the efficiency of the TBF-based algorithm to minimize the TT when the number of precedence constraints is sufficiently large allows to set θ to a slightly larger value (*e.g.*, $\theta = 2$ or 3), providing more flexibility in this step. In fact, to increase its efficiency, 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 $\mathcal{C}_{K_{i+\theta}}$. These artificial TWs are only used to reduce the number of variables in the TBF; they are not taken into consideration when identifying infeasible paths. Finally, Step Dis-3 simply ensures that all avoidable idle time is removed from the computed route.

4. Results on small and medium-sized benchmark instances

In this section, we consider small and medium-sized instances from the literature with dense TWs to motivate the use of a bi-objective cost function in the TSPTW considered. Given the size of these instances, we solved them only with the CP solver. For our tests, we used only one set of well-known instances, namely, that of Gendreau et al. (1998), hereafter called the GHLS instances. They are grouped in classes denoted nXXwYY, where XX is the number of nodes (between 20 and 100) and YY indicates the width of the TWs. There are 21 classes and 5 instances per class, for a total of 105 instances. Considering the size of these instances and the fact that all nodes have a TW, no clustering is applied for these instances. Note that we also performed tests on other instance sets, including that of Ascheuer et al. (2001) and Dumas, Desrosiers, Gelinas, and Solomon (1995), and obtained similar results.

Our computational experiments were ran on an Intel(R) Xeon(R) X5675 at 3.07 GHz using a single thread. The code was compiled in C++ and uses IBM CPLEX CP-Optimizer 12.7.1.0. Since the TSPTW may be solved many times in a CFRS procedure, we imposed a tight time limit of 60 seconds. For the WOF model, we used $\gamma^{TT} = 8$ and $\gamma^{RD} = 20$ except for the parameter sensitivity analysis. Furthermore, we define z^{WOF}, the WOF value of a solution, as $z^{WOF} = 8z^{TT} + 20z^{RD}$, where z^{TT} and z^{RD} are the TT and RD of this solution, respectively.

4.1. Comparative results

Figures 2 and 3 compare the solutions obtained by minimizing the WOF (first approach) versus those produced by minimizing TT or RD only (the reference approach), respectively. The figures show, for each instance class, the average savings (in percentage and represented by the dots) on the WOF value obtained by the solutions of the first approach over those of the reference one. A negative saving means that, on average, the reference approach yields a solution with a lower WOF value. Furthermore, the histograms indicate, for each class, the average differences in TT and in RD between these solutions. For example, in Figure 2, the bars represent the average variations in TT (blue) and in RD (red) between the solutions obtained with the WOF and when minimizing TT only.

From these results, we make the following observations. Compared to the TT solutions (Figure 2), 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 2 shows that the increase in the TT is usually compensated for by the reduction in the RD, and so the delivery companies could have routes with shorter durations while lowering their routing costs. This is not the case for the instance group n100w80, where TT-optimal solutions already have a good RD value, and the tight time limit does not allow the solver to generate such good solutions for the WOF (even if the average cost increases only by 0.24% for this group). Figure 3 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. This leads to savings ranging from -0.09% to 4.63%, with a global average of 2.52%.

Both series of results clearly highlight that focusing on a single objective can produce solutions that are significantly sub-optimal when both criteria play an important role in the routing costs. Therefore, a bi-objective function should be considered during optimization.

4.2. Sensitivity analysis on the weights of the WOF

Results for different ratios of the parameters γ^{TT} and γ^{RD} are presented in Table 1. For each setting, we report average results computed over the solutions obtained for all GHLS instances. In this table, the first column specifies the values of γ^{TT} and γ^{RD} used in the WOF model. The next three columns provide the average TT and RD values of the solutions, as well as the average computational time in seconds. Then, the last two columns report the average z^{WOF} savings in percentage with respect to the solutions computed when minimizing TT only and RD only, respectively. In the last two lines, we provide as references the average TT and RD values of the solutions computed when minimizing TT only and RD only.

This table first shows that it is always relevant to consider a bi-objective function because savings are always positive for the tested ratios. Furthermore, the lower the ratio γ^{TT}/γ^{RD} is, the closer the RD is to optimality, which is not surprising as a larger proportion of the cost is dedicated to minimizing RD, and the CP algorithm performs well on this part of the objective function. Finally, we observe that the average computational time increases as the ratio γ^{TT}/γ^{RD} converges to one, indicating that it is difficult to obtain a perfect tradeoff between TT and RD.

Table 1. Comparing the WOF solutions with different parameter settings

γ^{TT}/γ^{RD}	TT	RD	Time (s)	z^{WOF} savings (%)	
				vs TT-only	vs RD-only
1/20	475.9	539.9	46.8	5.07%	0.34%
5/20	470.2	541.0	52.3	3.43%	1.69%
8/20	469.1	541.5	56.5	2.53%	2.52%
13/20	466.1	543.3	58.5	1.39%	3.64%
20/20	463.5	544.5	59.0	0.43%	4.96%
TT-Only	440.6	571.8			
RD-Only	521.1	539.6			

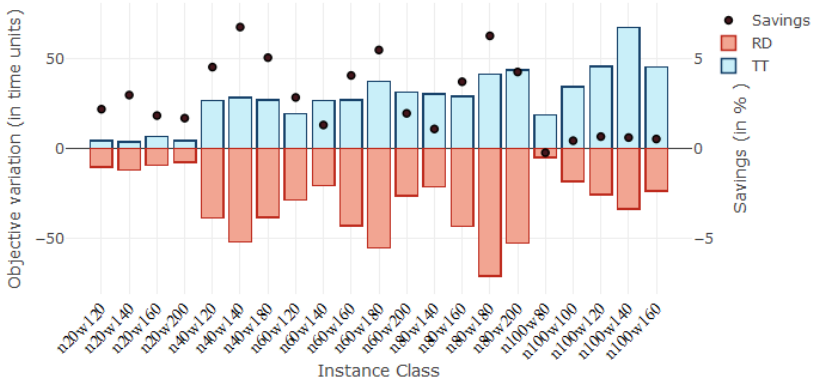

Figure 2. Savings and average TT and RD variations per class, for WOF versus TT only.

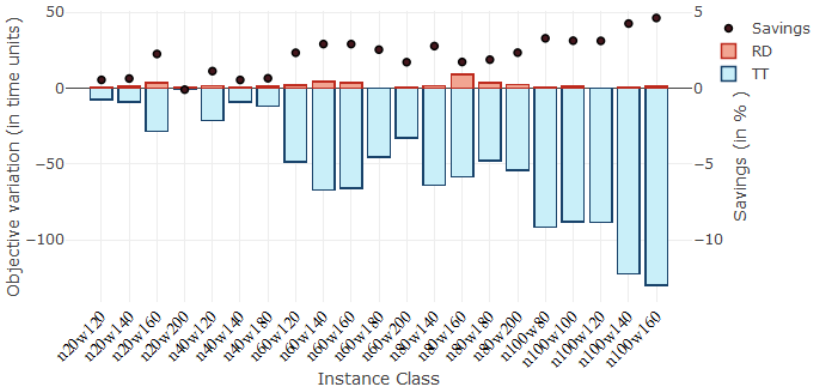

Figure 3. Savings and average TT and RD variations per class, for WOF versus RD only.

5. Results for large industrial 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 (Full_a) and 458 (Full_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 (where g = a or b) 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. Note that, in the original instances, the total service duration accounts for more than 50% of the total route duration, and this time is incompressible.

We present in this section several results about our clustering approach described in Section 3. First, we compare the usage of the two algorithms presented in Sections 3.3.2 and 3.3.3 for the disaggregation phase (Section 5.1). Then, we perform a sensitivity analysis on the value of some parameters used in the disaggregation phase (Section 5.2) and the aggregation phase (Section 5.3). Finally, we compare the performance of the clustering approach and the direct CP-WOF approach (Section 5.4) and provide optimality gaps for the solutions computed by the clustering algorithm (Section 5.5).

In the following tables, $\#N$ denotes the number of nodes remaining in the clustered problem, z^{WOF}_{Clus} the WOF value (cost) of the solution computed for the clustered problem, z^{WOF}_{Fin} the cost of the final solution and $Time$ the total computation time in seconds. For each instance, the best cost is highlighted in bold. Upon equality, only the cost obtained with fastest method is emboldened. Note that the WOF value of a solution is defined as in the previous section, i.e., $z^{WOF} = 8z^{TT} + 20z^{RD}$.

5.1. Comparison of disaggregation algorithms

Sections 3.3.2 and 3.3.3 describe two algorithms for disaggregating a clustered solution. The first solves a CP model whereas the second combines this CP model with a TBF. In Table 2, we report the results obtained with both algorithms when a maximum of 20 seconds is allocated to compute a solution in each phase (the total time may exceed 40 seconds because of the time required to cluster the nodes and build the models) and the parameters are set as follows: $d^{max} = 20$, $n^{max} = 20$, and $\theta = 1$.

From these results, we observe first that the costs of the final solutions are much less than those of the clustered solutions. In general, the final solutions computed by both disaggregation algorithms are of similar quality, with a small advantage for the mixed CP and TBF algorithm. On the other hand, this algorithm is much faster than the pure CP algorithm, with an average total time reduction of 30% (obtained by reducing the disaggregation time by 70% on average). Given that most times required by the mixed CP and TBF algorithm are below the 40-second time limit, we conclude that the corresponding solutions computed by this algorithm are optimal for the disaggregation phase. Moreover, with the pure CP algorithm, increasing θ to 2 or more yields poor-quality solutions in a limited computational time. These observations led us to use the mixed CP and TBF disaggregation algorithm to produce the subsequent results.

Table 2. Comparative results for the disaggregation algorithms

			Pure CP		CP + TBF	
Instance	#N	z^{WOF}_{Clus}	z^{WOF}_{Fin}	Time (s)	z^{WOF}_{Fin}	Time (s)
rd.200.1a	75	152.63	151.05	41.5	**151.05**	23.4
rd.200.2a	69	164.55	162.45	41.3	**162.45**	23.2
rd.200.3a	72	162.08	159.47	41.4	**159.47**	23.2
rd.300.1a	93	188.51	185.13	42.6	**184.87**	26.9
rd.300.2a	86	185.50	181.35	42.7	**181.26**	26.7
rd.300.3a	89	189.65	186.45	42.8	**186.10**	26.9
Full_a	94	232.35	226.78	48.0	**226.75**	42.9
rd.200.1b	79	177.89	176.42	41.3	**176.42**	22.9
rd.200.2b	75	174.10	172.19	41.4	**172.19**	23.1
rd.200.3b	70	166.26	165.23	41.6	**165.23**	22.9
rd.300.1b	84	207.86	**204.54**	42.8	204.73	41.9
rd.300.2b	84	206.25	204.27	42.8	**203.78**	26.9
rd.300.3b	88	208.49	205.23	42.6	**205.06**	26.7
Full_b	95	252.31	247.55	47.5	**247.51**	41.7
Avg	82	190.60	187.72	42.9	**187.63**	28.5

5.2. Impact of the flexibility parameter value

In this section, we assess the impact of the value of flexibility parameter θ on the solution quality and the total computational time. This parameter controls the number of clustered nodes that can be reordered around each node in the disaggregation phase. In Table 3, we present the results obtained for three different values of θ, namely, $\theta = 1$ as in the previous section, as well as $\theta = 2$ and $\theta = 3$ which allow more flexibility during disaggregation. For these tests, a maximum time limit of 300 seconds for the whole solution process was set to yield optimal solutions for the disaggregation phase in all instances except for two with $\theta = 3$. From these results, we observe that increasing the flexibility enables slightly improving the quality of the final solution (by 0.13% and 0.18% for $\theta = 2$ and $\theta = 3$, respectively) but could be costly in time consumption. Because $\theta = 2$ offers a good trade-off between solution quality and computational time, we use this value in the following sections.

Table 3. Comparative results for three values of θ

			$\theta = 1$		$\theta = 2$		$\theta = 3$	
Instance	#N	z^{WOF}_{Clus}	z^{WOF}_{Fin}	Time (s)	z^{WOF}_{Fin}	Time (s)	z^{WOF}_{Fin}	Time (s)
rd.200.1a	75	152.63	151.05	23.4	**150.72**	25.1	150.72	29.2
rd.200.2a	69	164.55	162.45	23.2	**162.26**	25.7	162.26	34.2
rd.200.3a	72	162.08	159.47	23.2	159.28	24.2	**159.02**	42.3
rd.300.1a	93	188.51	184.87	26.9	184.24	35.5	**183.91**	76.9
rd.300.2a	86	185.50	181.26	26.7	181.02	59.6	**180.95**	95.3
rd.300.3a	89	189.65	186.10	26.9	186.13	57.0	**185.31**	102.4
Full_a	94	232.35	226.75	42.9	**226.52**	71.7	226.92	300.0
rd.200.1b	79	177.89	176.42	22.9	176.42	24.2	**176.39**	30.1
rd.200.2b	75	174.10	172.19	23.1	172.10	24.6	**171.82**	32.1
rd.200.3b	70	166.26	165.23	22.9	165.23	24.3	**165.09**	28.4
rd.300.1b	84	207.86	204.54	65.8	204.05	35.4	**203.98**	80.0
rd.300.2b	84	206.25	203.78	26.9	**203.52**	44.7	203.52	164.8
rd.300.3b	88	208.49	205.06	26.7	204.57	37.3	**204.20**	115.9
Full_b	95	252.31	247.51	41.7	**246.97**	78.8	247.65	300.0
Avg	82	190.60	187.62	30.2	187.36	40.6	**187.27**	102.3

5.3. Impact of the clustering parameter values

As mentioned above, all our basic tests were run using $d^{max} = 20$ and $n^{max} = 20$, yielding clustered problems with an average of 82 clustered nodes and a maximum of 95 nodes. To show the impact of using clustered problems that are less clustered, we solved all instances using a less aggressive clustering procedure. This procedure

stopped when the number of clustered nodes reached 100 (resp. 120) nodes, except when the minimum traveling time d_i is equal to the previous one. In this case, the aggregation continues until finding a larger minimum traveling time, yielding a clustered problem with slightly less than 100 (resp. 120) nodes. The results of these tests are reported in Table 4 for the three different stopping criterion, where the latter are denoted $\#N \approx 100$ and $\#N \approx 120$.

Table 4. Comparative results for three clustering procedure stopping criteria

	$d^{max} = 20$, $n^{max} = 20$			$\#N \approx 100$			$\#N \approx 120$		
Instance	#N	z_{Fin}^{WOF}	Time	#N	z_{Fin}^{WOF}	Time	#N	z_{Fin}^{WOF}	Time
rd.200.1a	75	**150.72**	25.7	100	151.23	24.2	117	151.02	23.6
rd.200.2a	69	**162.26**	26.2	100	162.33	24.4	113	162.33	23.8
rd.200.3a	72	159.28	24.7	100	**159.16**	24.0	104	159.44	24.0
rd.300.1a	93	184.24	36.1	94	**183.33**	34.7	119	185.92	31.1
rd.300.2a	86	**181.02**	58.7	99	181.23	35.8	119	181.49	28.6
rd.300.3a	89	185.47	70.3	100	185.78	32.2	115	**184.75**	31.6
Full_a	94	226.52	76.4	99	**225.87**	94.7	120	227.83	63.2
rd.200.1b	79	176.42	24.8	100	**173.94**	24.3	115	173.99	23.5
rd.200.2b	75	172.10	25.4	92	172.17	24.2	110	**171.49**	23.7
rd.200.3b	70	165.23	25.1	94	**164.20**	24.4	105	164.81	23.7
rd.300.1b	84	**204.05**	35.9	98	204.78	34.2	118	205.76	28.3
rd.300.2b	84	**203.52**	45.3	100	203.68	30.9	120	204.10	28.8
rd.300.3b	88	**204.57**	37.9	100	205.44	32.2	120	206.72	29.8
Full_b	95	246.97	80.1	100	**244.78**	93.6	120	249.98	59.3
Avg	82.4	187.31	42.3	98.3	**186.99**	38.1	115.4	187.83	31.7

These results show that the second approach ($\#N \approx 100$) produces better solutions on average than the other two. The third one ($\#N \approx 120$) is the fastest. It may be explained by the fact that the clustered nodes contain less customers and, thus, the disaggregation problem is easier to solve, but less permissive. The first approach ($d^{max} = 20$, $n^{max} = 20$) has the most aggressive clustering phase, increasing the probability of sub-optimal inner routes in the cluster nodes and a poorer approximation. This could be counter-balanced by the fact that the clustered problem is easier to solve, due to fewer nodes in the instances. For the following section, the results were produced using the clustering setting $\#N \approx 100$.

5.4. Direct CP-WOF algorithm versus clustering algorithm

In this section, we compare the direct CP-WOF algorithm of Section 2 and the clustering heuristic. For the latter, we imposed a time limit of 20 seconds for solving the clustered problem. For both algorithms, a 60-second overall time limit was enforced.

The results are reported in Table 5 where the last column (Imp) gives the cost improvement in percentage obtained by the solutions produced by the clustering heuristic over those computed by the direct CP-WOF algorithm. On average, the clustering heuristic yields an average cost improvement of 1.60%, in around half the time allowed for the CP-WOF method. Note that the disaggregation method terminated before the time limit for all the instances except the full ones, ensuring that there is no better feasible disaggregation given the computed clusters and clustered sequence. Observe also that the cost z_{Clus}^{WOF} of the clustering phase solution is, on average, less than that of the direct CP-WOF algorithm solution (z^{WOF}).

Figure 4 presents the cost of the best solution found in function of the computational time for the two original instances Full_a and Full_b and two different algorithms, namely, the direct CP-WOF algorithm (solid lines) and the proposed clustering algorithm (dashed-dotted lines). For the latter algorithm, the solution process stops when the disaggregation problem is solved to optimality at around 90 seconds. For these

Table 5. Comparative results between the CP-WOF algorithm and the clustering algorithm

	WOF		Clustering				
Instance	z^{WOF}	Time	#N	z^{WOF}_{Clus}	z^{WOF}_{Fin}	Time	Imp
rd.200.1a	157.28	60.0	100	151.96	**151.23**	23.4	3.37%
rd.200.2a	164.15	60.0	100	163.01	**162.33**	23.5	0.96%
rd.200.3a	159.30	60.0	100	160.54	**159.16**	23.5	0.08%
rd.300.1a	185.55	60.0	94	187.20	**183.33**	34.3	1.01%
rd.300.2a	186.18	60.0	99	184.24	**181.23**	35.4	2.26%
rd.300.3a	186.03	60.0	100	188.60	**185.78**	31.5	0.12%
Full_a	234.90	60.0	99	231.70	**226.43**	60.0	3.17%
rd.200.1b	176.84	60.0	100	175.32	**173.94**	23.6	1.41%
rd.200.2b	173.15	60.0	92	173.50	**172.17**	23.7	0.49%
rd.200.3b	166.42	60.0	94	165.23	**164.20**	23.9	1.16%
rd.300.1b	210.33	60.0	98	208.16	**204.78**	33.8	2.22%
rd.300.2b	206.44	60.0	100	205.64	**203.68**	30.3	1.12%
rd.300.3b	209.89	60.0	100	207.49	**205.44**	31.6	1.79%
Full_b	251.82	60.0	100	249.79	**245.15**	60.0	2.29%
Avg	190.59	60.0	98	189.46	**187.06**	32.7	1.60%

two instances, we clearly see that the clustering algorithm can provide much better quality solutions than the direct CP-WOF algorithm, even if a 60-second time limit was imposed.

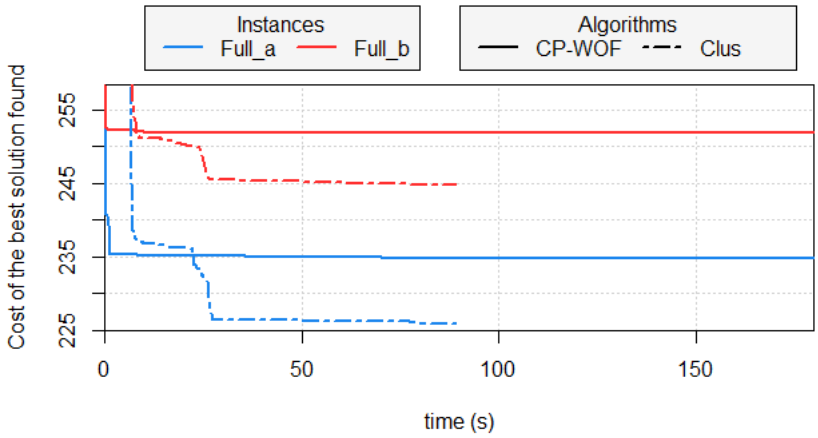

Figure 4. Cost of the best solution found in function of the computational time

5.5. Optimality gaps of the clustering algorithm solutions

To conclude the assessment of the quality of the solutions produced by the clustering algorithm, we computed for each instance a lower bound on its optimal value by solving the corresponding traveling salesman problem (TSP), defining the arc costs as the arc TTs only (i.e., omitting the service times). Because $z^{WOF} = 8z^{TT} + 20z^{RD}$ and $z^{RD} \geqslant z^{TT} + ST$, where ST is the total service time, we deduce that $z^{WOF} \geqslant 28z^{TT} + 20ST$. Therefore, a valid lower bound (LB) on the optimal WOF value is given by $LB = 28z^{TT}_{TSP} + 20ST$, where z^{TT}_{TSP} is the optimal value of the corresponding TSP. For each instance, Table 6 reports this lower bound (LB) and the relative optimality gap between LB and z^{WOF}_{Fin}, the cost of solution computed by the clustering algorithm. The results indicate that the computed solutions are all within 1.8% of optimality,

with an average of 1.3%. Given that the lower bounds are not necessarily tight (TWs are relaxed in the TSP), these gaps show the effectiveness of the proposed clustering algorithm.

Table 6. Optimality gaps

Instance	LB	z_{Fin}^{WOF}	Gap (%)
rd.200.1a	148.88	151.23	1.58
rd.200.2a	160.77	162.33	0.97
rd.200.3a	156.99	159.16	1.38
rd.300.1a	180.88	183.33	1.35
rd.300.2a	179.25	181.23	1.11
rd.300.3a	183.40	185.78	1.30
Full_a	223.37	226.43	1.37
rd.200.1b	171.96	173.94	1.15
rd.200.2b	169.13	172.17	1.79
rd.200.3b	161.99	164.20	1.37
rd.300.1b	202.68	204.78	1.04
rd.300.2b	200.77	203.68	1.45
rd.300.3b	201.89	205.44	1.76
Full_b	243.26	245.15	0.78
Avg	184.66	187.06	1.30

6. 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 but also the total route duration, and the fact that very few customers have time windows, reducing significantly the efficiency of the standard TW preprocessing techniques. To solve this new TSPTW variant, we devised a three-step clustering heuristic which first clusters the unconstrained customers, then solves the clustered problem, before sequencing the customers inside the clustered nodes, with some flexibility between the consecutive nodes. Compared to a CP algorithm that tackles the problem at once, this heuristic is able to produce better-quality solutions (with an average optimality gap of 1.3%) in shorter computational times on industrial instances involving up to 475 customers.

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.

Acknowledgements

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.

References

Ascheuer, N., Fischetti, M., & Grötschel, M. (2001, May 01). Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. Mathematical Programming, 90(3), 475–506.

Baldacci, R., Mingozzi, A., & Roberti, R. (2012). New State-Space Relaxations for Solving the Traveling Salesman Problem with Time Windows. INFORMS Journal on Computing, 24(3), 356–371.

Boland, N., Hewitt, M., Vu, D. M., & Savelsbergh, M. (2017). Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks. In Integration of ai and or techniques in constraint programming: 14th international conference, cpaior 2017, padua, italy, june 5-8, 2017, proceedings (pp. 254–262). Cham: Springer International Publishing.

Calvo, R. W. (2000). A new heuristic for the traveling salesman problem with time windows. Transportation Science, 34(1), 113–124.

Christofides, N., Mingozzi, A., & Toth, P. (1981). State-space relaxation procedures for the computation of bounds to routing problems. Networks, 11(2), 145–164.

Dash, S., Gunluk, O., Lodi, A., & Tramontani, A. (2012). A time bucket formulation for the traveling salesman problem with time windows. INFORMS Journal on Computing, 24(1), 132-147.

Deb, K. (2014). Multi-objective optimization. In E. K. Burke & G. Kendall (Eds.), Search methodologies: Introductory tutorials in optimization and decision support techniques (pp. 403–449). Boston, MA: Springer US.

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

Dumas, Y., Desrosiers, J., Gelinas, E., & Solomon, M. M. (1995, apr). An optimal algorithm for the traveling salesman problem with time windows. Operations Research, 43(2), 367–371.

Ferreira da Silva, R., & Urrutia, S. (2010). A general vns heuristic for the traveling salesman problem with time windows. Discrete Optimization, 7, 203–211.

Gendreau, M., Hertz, A., & Laporte, G. (1992). New insertion and postoptimization procedures for the traveling salesman problem. Operations Research, 40(6), 1086–1094.

Gendreau, M., Hertz, A., Laporte, G., & Stan, M. (1998, mar). A generalized insertion heuristic for the traveling salesman problem with time windows. Operations Research, 46(3), 330–335.

Kara, I., & Derya, T. (2015). Formulations for minimizing tour duration of the traveling salesman problem with time windows. Procedia Economics and Finance, 26, 1026–1034.

Langevin, A., Desrochers, M., Desrosiers, J., Gélinas, S., & Soumis, F. (1993). A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. Networks, 23(7), 631–640.

López-Ibáñez, M., & Blum, C. (2010). Beam-aco for the travelling salesman problem with time windows. Computers & Operations Research, 37(9), 1570 - 1583.

López-Ibáñez, M., Blum, C., Ohlmann, J. W., & Thomas, B. W. (2013). The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. Applied Soft Computing, 13(9), 3806–3815.

Ohlmann, J. W., & Thomas, B. W. (2007). A compressed-annealing heuristic for the traveling salesman problem with time windows. INFORMS Journal on Computing, 19(1), 80–90.

Savelsbergh, M. (1985). Local search in routing problems with time windows. Annals of Operations Research, 4(1), 285–305.

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

Savelsbergh, M. (1992). The Vehicle Routing Problem with Time Windows : minimizing route duration. ORSA Journal on Computing, 4(2), 146-154.

van Hoeve, W.-J. (2001). The alldifferent constraint: A survey. arXiv preprint cs/0105015.

Appendix A. Time Bucket Formulation

The TBF (Dash et al., 2012) 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 discussed in Section 2.
- $I_k(i, b)$ 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

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

where $b_0 = -\infty$ is assumed.

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

$$\min \sum_{(i,j)\in A} t_{ij} x_{ij} \tag{A1}$$

subject to:

$$\sum_{b\in B_i} z_i^b = 1, \qquad \forall i \in V \tag{A2}$$

$$\sum_{j\in V^+(i)} y_{ij}^b = z_i^b, \qquad \forall i \in V\backslash\{q\}, \quad \forall b \in B_i \tag{A3}$$

$$\sum_{k\in V^-(i)} \sum_{\beta\in I_k(i,b)} y_{ki}^\beta = z_i^b, \qquad \forall i \in V\backslash\{p\}, \quad \forall b \in B_i \tag{A4}$$

$$\sum_{b\in B_i} y_{ij}^b = x_{ij}, \qquad \forall (i, j) \in A \tag{A5}$$

$$x_{ij}, y_{ij}^b, z_i^b \in \{0, 1\}, \qquad \forall i \in V, \quad \forall (i, j) \in A, \quad \forall b \in B_i. \tag{A6}$$

The objective function (A1) minimizes the TT. Constraints (A2) ensure that each service point is visited in a single bucket. Constraints (A3) and (A4) link the variables y and z, while guaranteeing that a valid bucket is used via the set $I_k(i, b)$. Constraints (A5) link x and y.

Model (A1)–(A6), 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 A1, subtour elimination constraints (SECs) and infeasible path cuts (IPCs) should be added to obtain an exact formulation, the so-called TBF.

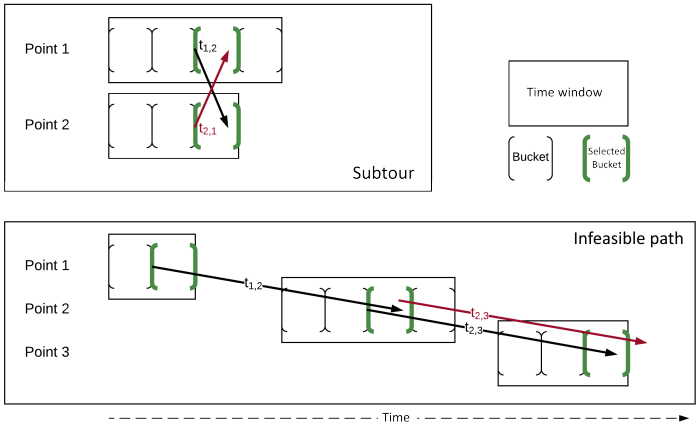

Figure A1. Subtour and infeasible path examples.

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 A1, 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:

$$\sum_{(i,j)\in\delta(S)} x_{ij} \geqslant 1 \tag{A7}$$

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:

$$\sum_{(i,j)\in\mathcal{P}} x_{ij} \leqslant |\mathcal{P}| - 1. \tag{A8}$$

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 minimize z, where z is defined by the following constraint, leading to an overestimation of the real RD objective function:

$$z \geqslant \sum_{b\in B_q} \bar{b} z_q^b - \sum_{b\in B_p} \underline{b} z_p^b. \tag{A9}$$

However, it is possible to be more accurate, especially when the waiting time is low,

by replacing (A9) by:

$$z \geqslant \sum_{b \in B_q} \underline{b} z_q^b - \sum_{b \in B_p} \bar{b} z_p^b \tag{A10}$$

$$z \geqslant \sum_{(i,j) \in A} t_{ij} x_{ij}. \tag{A11}$$

In both cases, the buckets have to be carefully assigned, and this requires the addition of some constraints; see Dash et al. (2012). 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 the RD. When an upper bound on RD is imposed like in Section 3.3.3, 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 (A7) and (A8) to the model.

Appendix B. Supporting Computational Results

In this appendix, we present some computational results to support our claims that the CP algorithm is efficient at minimizing RD for small and medium-sized instances while it is outperformed by the TBF-based algorithm when minimizing TT.

First, using the CP algorithm, we solved the 105 GHLS instances (Gendreau et al., 1998) with the objective of minimizing RD only. Figure B1 reports the performance profile curve for these experiments. It shows the number of instances solved to optimality (vertical axis) with respect to a time limit (horizontal axis) given in milliseconds and using a logarithmic scale. We can observe that 96 out of the 105 instances can be solved to optimality within one second of computational time and that the most difficult instance is solved in 71 seconds.

We conducted another set of experiments that consists of solving the 60 GHLS instances with less than 60 nodes and the objective of minimizing TT, using both the CP and the TBF-based algorithms. A maximum time of 300 seconds was imposed to solve each instance. In Figure B2, we report for each instance the optimality gaps obtained for the CP algorithm (blue) and the TBF-based algorithm (red) (some circles are superimposed). A cross indicates that the TBF-based algorithm cannot find a feasible solution for the corresponding instance within the time limit. These results show that the CP algorithm always returns a solution, whereas the TBF-based algorithm cannot for 19 instances. However, when the TBF-based algorithm can find a solution for an instance, it generally solves it to optimality. Table B1 reports the number of instances solved to optimality by both algorithms with respect to several time limits.

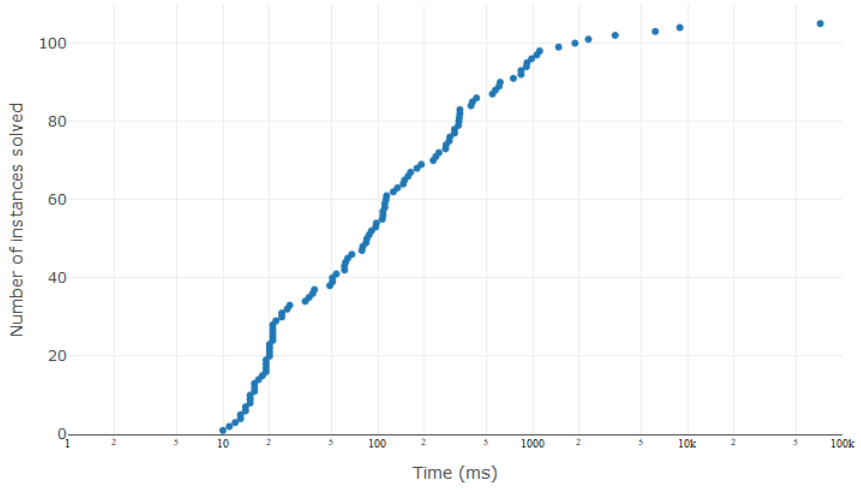

Figure B1. Performance profile curve for minimizing RD with the CP algorithm

From these results, we deduce that the TBF-based algorithm is much faster than the CP algorithm to prove optimality.

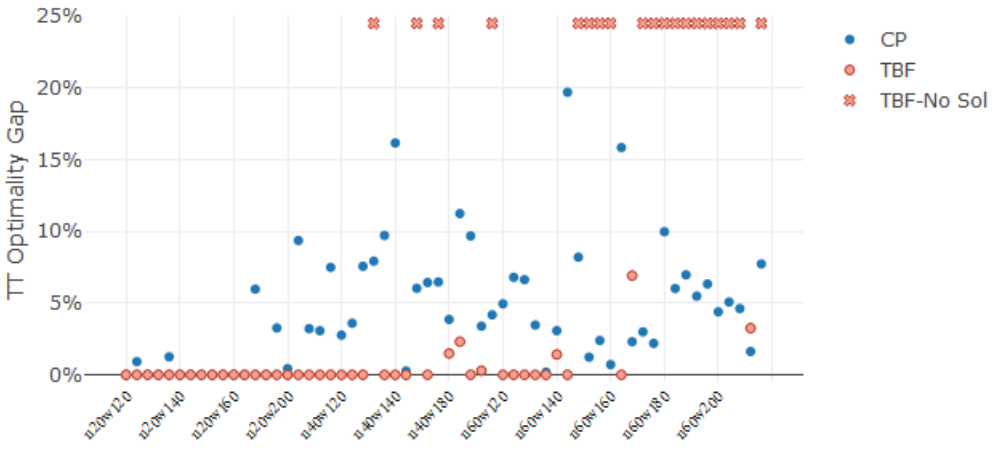

Figure B2. Optimality gap when minimizing TT using each algorithm

Table B1. Number of instances solved to optimality with respect to a time limit when minimizing TT using each algorithm

	# instances solved	
Time limit (s)	CP	TBF
1	0	6
5	0	11
30	0	31
120	1	34
300	2	35