CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A General Variable Neighborhood Search Heuristic for the Traveling Salesman Problem with Time Windows under Completion Time Minimization

Khalid Amghar
Jean-François Cordeau
Bernard Gendron

July 2019

CIRRELT-2019-29

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

A General Variable Neighborhood Search Heuristic for the Traveling Salesman Problem with Time Windows under Completion Time Minimization

Khalid Amghar[1,2,*], Jean-François Cordeau[1,3], Bernard Gendron[1,2]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-Ville, Montréal, Canada H3C 3J7

[3] Department of Logistics and Operations Management, HEC Montréal, 3000 Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

Abstract. We propose a two-phase general variable neighborhood search (GVNS) heuristic to solve the traveling salesman problem with time windows (TSPTW) when the objective is to minimize the completion time (that is, the arrival time at the depot). The first phase aims to find a feasible solution using variable neighborhood search (VNS). In the second phase, we adapt and improve an existing GVNS-based method, which gave good results for the TSPTW when the objective is to minimize the distance. We use efficient methods for checking the feasibility and the profitability of a move, and for exploring the neighborhoods. A basic preprocessing and the properties of the travel time matrix allow us to reduce the search space without excluding any feasible solution. The results indicate that our method is at least as good as state-of-the-art heuristics. In particular, we find at least six new best known solutions for the TSPTW when the objective is to minimize the completion time.

Keywords. GVNS, TSPTW, completion time, neighborhood evaluation, neighborhood search.

---

* Corresponding author: Khalid.Amghar@cirrelt.ca

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

1. Introduction

The traveling salesman problem (TSP) has been widely studied by the operations research community. Many variants of the problem are considered in the literature (Ilavarasi and Joseph, 2014). One of the best known variants is the traveling salesman problem with time windows (TSPTW). This problem has several applications, including package delivery, schoolbus transportation and machine scheduling. The TSPTW is defined on a directed graph $G = (N^c \cup \{0, n+1\}, R)$ where $N^c = \{1, 2, ..., n\}$ is the set of customers, node 0 is the origin depot, node $n+1$ is the destination depot (the two depots are the same in practice) and R is the set of arcs. A single vehicle must start from the origin depot, visit each of the n customers exactly once, and finish its route at the destination depot. Besides, each customer i has to be visited within its time window $[a_i, b_i]$, which is known in advance. If the vehicle arrives at customer i before a_i, it must wait until the start of the customer's time window. However, customer i cannot be served if the vehicle arrives after the end of the time window, i.e, after b_i. Furthermore, every customer requires a certain time to be served. T is the travel time (or distance) matrix, and t_{ij} is the travel time from node i to node j. This travel time also includes the service time at node i (the service time is null for the depot). The travel times are assumed to be positive and to satisfy the triangle inequality.

The introduction of time windows can significantly reduce the number of feasible solutions, but it also complicates the task of finding a feasible solution (if such a solution exists). In particular, Savelsbergh (1985) proves that the problem of finding a feasible tour for the TSPTW is NP-complete in the strong sense, even in the case where the distance (or time) matrix is symmetric and satisfies the triangle inequality.

The aim of the TSPTW is to find an optimal tour with respect to some objective function, which can vary depending on the application and the goals of the decision-makers. We can target, among others, the minimization of the total distance (TSPTW-T); the minimization of the total waiting time; and the minimization of the total duration (TSPTW-D), that is to say, the arrival time at the destination depot minus the departure time from the origin depot. In this variant, the departure time from the origin depot is not known in advance, and it is a decision variable. Another possible objective is the minimization of the completion time (TSPTW-C), that is to say, the minimization of the arrival time at the destination depot. It is this last objective function that we consider in this paper. TSPTW-C is also known as the makespan problem with time windows (MPTW). It is useful when the start time of the vehicles is fixed in advance, especially in the case of school transport and staff transportation of a company, when one

wants the vehicles to return as soon as possible so that the students or employees are not late. This objective is also relevant in the context of machine scheduling.

We present in this paper a two-phase heuristic based on variable neighborhood search (VNS) (Mladenović and Hansen, 1997; Hansen et al., 2008). In the first phase, named the constructive phase, we find a feasible solution through the use of a VNS heuristic presented in Da Silva and Urrutia (2010). In the second phase, we try to improve the feasible solution found in the constructive phase. We use a VNS variant named GVNS, a heuristic presented in Mladenović et al. (2013), which turned out to be robust and efficient in addressing the TSPTW in which the objective is to minimize the total distance. Hence, we adapt it for the TSPTW variant in which the objective is to minimize the arrival time at the depot. In particular, we introduce an efficient approach to verify the profitability of a move and we combine it with an existing method that checks the feasibility of a move efficiently. We also discuss the order of neighborhoods to be considered, and for each neighborhood, we explain how the order of exploration plays a crucial role in accelerating the evaluation of this neighborhood. We improve the best known solutions of at least 6 instances (see details in Section 4). In addition to the improved best known solutions, we provide, to the best of our knowledge, the first results on 45 instances when the objective is to minimize the arrival time at the depot. Another contribution of our paper is to serve as a basis of comparison for future works since it contains the results on many instances from the literature. The presented method is robust, and the results show that it is at least as good as the method currently considered as the state of the art.

The next section surveys the related literature. The algorithm that we propose and the acceleration techniques are presented in Section 3, which also gives an overview of the neighborhoods and the move operators of known local search procedures (LSPs) for the TSPTW, especially those that we use in our method. In Section 4, we present the computational results of our method for many known instances in the literature, and we compare these results, when possible, to existing ones.

2. Literature review

The TSPTW literature is so rich that we cannot make an exhaustive review. We underline that even if we focus on the TSPTW-C (minimizing the completion time), we also survey the literature on other variants, especially the TSPTW-T (minimizing the travel time or the total distance) and TSPTW-D (minimizing the route duration). This is justified by the fact that even if the objective functions are different,

the problem structure remains the same. We first present exact methods followed by heuristic ones.

2.1. Exact methods

Branch-and-bound and dynamic programming have been widely used to solve the TSPTW to optimality. Christofides et al. (1981) solve the TSPTW-C by combining branch-and-bound with dynamic programming. The authors report results on instances with up to 50 nodes. Baker (1983) solves the TSPTW-C by a branch-and-bound algorithm and uses simple preprocessing techniques to eliminate incompatible arcs (an arc is incompatible if it cannot belong to a feasible solution), solving instances with up to 50 customers. Langevin et al. (1993) present a two-commodity flow model which can solve the TSPTW-T and the TSPTW-C (it can be generalized to handle several variants of the TSP). They solve their problem by branch-and-bound. The authors also introduce several procedures to eliminate incompatible arcs and to reduce the width of the time windows. The authors present experimental results for instances containing up to 60 customers. Pesant et al. (1998) present a constraint programming model of the TSPTW-T and solve it by branch-and-bound. The use of incompatible arcs elimination techniques and the reduction of the time windows is not limited to the preprocessing phase; it is dynamically performed during the algorithm's execution. Focacci et al. (2002) consider the TSPTW-T and use constraint programming to handle time window constraints. The optimization is guided by a search tree similar to branch-and-bound. Dumas et al. (1995) consider the TSPTW-T and introduce new elimination tests allowing to improve the performance of a dynamic programming approach, based on a reformulation as a constrained shortest path problem. They report the results on instances with up to 200 customers. Baldacci et al. (2012) solve the TSPTW-T with a dynamic programming approach based on three relaxations of the problem. They report results on 270 instances from the literature. Their algorithm can solve to optimality all these instances but one.

2.2. Heuristic methods

Local-search-based procedures have been widely used to solve the TSPTW. Savelsbergh (1985) introduced a method based on local search to solve the TSPTW-T and the TSPTW-C. The method consists of two phases. The first one uses an insertion heuristic to find a feasible solution. The second one improves the initial solution by applying local search procedures. Savelsbergh (1992) considers the variant of the TSPTW where the objective is to minimize the tour duration (TSTPW-D). He introduces an efficient method to quantify the impact of a move on the value of the objective function. We note that

the algorithms of Savelsbergh (1985) and Savelsbergh (1992) do not use any procedure for eliminating incompatible arcs. Gendreau et al. (1998) present an insertion heuristic for the TSPTW-T. If the algorithm can construct a feasible solution, then a procedure is executed to improve this solution. Carlton and Barnes (1996) solve the TSPTW-C and the TSPTW-T with a reactive tabu search method, which adjusts the parameters during the execution of the algorithm. Their algorithm allows visiting infeasible solutions by introducing a static penalty in the objective function.positions Da Silva and Urrutia (2010) opt for a GVNS heuristic to solve the TSPTW-T. Their method is composed of two phases. The first one, called a constructive phase, finds a feasible solution using VNS. This new method is more efficient than those presented in Gendreau et al. (1998) and Calvo (2000) regarding finding a feasible solution. The second phase, called the optimization phase, uses a GVNS algorithm to improve the solution found in the first phase. Their algorithm uses a basic preprocessing to eliminate incompatible arcs. Mladenović et al. (2013) also opt for a GVNS heuristic to solve the TSPTW-T. Their method follows two steps. The first step uses the algorithm of the constructive phase presented in Da Silva and Urrutia (2010) to find a feasible solution. The second step tries, using a GVNS heuristic, to improve the feasible solution found in the first step. The main differences between the methods of Mladenović et al. (2013) and Da Silva and Urrutia (2010) are that the former uses a vector that allows verifying the feasibility of a move efficiently, it explores more neighborhoods, and it does not perform any elimination of incompatible arcs. Mladenović et al. (2013) improve the best known solutions for some instances.

Favaretto et al. (2006) propose an ant-colony-based procedure. Their method is applied to the TSPTW-T and to another variant. The first step of their algorithm tries to find a feasible solution using a variant of the nearest neighbor heuristic of Solomon (1987). However, this algorithm does not ensure that the initial solution found contains all the customers. In the second step, the ant colony algorithm is executed while prioritizing, during the comparison between different solutions, the one that contains the highest number of customers. López-Ibáñez and Blum (2010) propose to solve the TSPTW-T with a hybrid method that combines an ant colony optimization (ACO) algorithm with a beam search method. The resulting method is known under the name of Beam-ACO (Blum, 2005). The beam search can be seen as a heuristic version of branch-and-bound. The authors present stochastic sampling techniques to deduce information about the bounds.

Ohlmann and Thomas (2007) solve the TSPTW-T with a compressed-annealing heuristic (a variant of simulated annealing). For the compressed annealing, a parameter, named pressure, controls the prob-

ability of making a move to an infeasible solution while adding a variable penalty term to the objective function.

López-Ibáñez et al. (2013) solve the TSPTW-C by adapting two methods for the TSPTW-T, compressed annealing (Ohlmann and Thomas, 2007) and Beam-ACO (López-Ibáñez and Blum, 2010), that generated good results. For each of the two methods, the authors execute a tuning process to identify the best parameters and configurations for each set of instances presented in their article. The obtained results are good for the two methods, but Beam-ACO is significantly better than the compressed annealing method. The authors improve the best known solutions for some instances and present, for the first time in the literature on the TSPTW-C, the results on several known instances.

3. The proposed method

As mentioned before, we adapt and improve the method presented in Mladenović et al. (2013), which was initially used to solve the TSPTW-T. We introduce some modifications to solve the TSPTW-C efficiently. In particular, we use some techniques to check the profitability of a move efficiently, and to reduce the search space while ensuring that all feasible solutions are still reachable.

3.1. Notation

We now summarize the notation defined in the introduction and define the notation that we use thereafter:

n is the number of customers;

T is the travel time (or cost or distance) matrix and t_{ij} is the time needed to move from node i to node j, which also includes the service time at node i (the service time is null for the depot);

A_i is the arrival time at node i;

a_i is the start of the time window of node i;

b_i is the end of the time window of node i;

w_i is the waiting time before the service begins at node i (this waiting time is null for the depot).

We recall that waiting is permitted if the vehicle arrives at a customer i before a_i. Thus, one has the following equation: $w_i = \max\{0, a_i - A_i\}$.

3.2. Neighborhoods for the TSPTW

We briefly present hereafter some neighborhoods for the TSPTW. Several papers discuss neighborhoods for the TSPTW in more depth, e.g., Lin (1965), Lin and Kernighan (1973) and Funke et al. (2005).

The k-opt move for the TSPTW removes k arcs from the tour and replaces them with k other arcs so that the resulting solution is still a tour. The move is well known because it generalizes, as we will see, several other moves known by different names. In our method, we use the 2-opt neighborhood.

Moves that involve moving a chain of m consecutive customers are also widely used. In general, the value of m does not exceed three. We call these moves Or-opt-m. In this move, a chain of m customers is moved a number of positions forward or backward. We consider in our method the moves Or-opt-1 and Or-opt-2. These moves can be seen as special cases of 3-opt.

We also use a neighborhood called 1-opt (we use the same names as in Mladenović et al. (2013)) which consists in exchanging the order of two successive customers. This move can be seen as a special case of the Or-opt-1 and 2-opt moves.

3.3. General algorithm

Our method, summarized in Algorithm 1, proceeds in two phases. The first phase (Line 1 of the algorithm) is called the constructive phase. Its goal is to find a feasible solution for the TSPTW-C using a VNS-based heuristic presented in Da Silva and Urrutia (2010). VNS is a metaheuristic based upon systematic changes of neighborhoods combined with local search. These changes of neighborhoods may be used both in the intensification (descent) and the diversification (perturbation) phase. For more details about the variable neighborhood search and the related procedures and variants, see Hansen et al. (2008).

Algorithm 1: $GVNS-TSPTW-C$

```
1  x ← ConstructivePhase(maxLevel, t_max);
2  repeat
3      k ← 1;
4      repeat
5          x' ← Shake(x, k);
6          x'' ← VND(x', 6);
7          NeighborhoodChange(x, x'', k);
8      until k = k_max;
9      t ← CpuTime();
10 until t > t_max;
```

The second phase, called the optimization phase, is based on GVNS, which is a variant of VNS in which the local search procedure is replaced by a variable neighborhood descent (VND). The latter is shown in Algorithm 3. In the VND, the change of the neighborhoods is performed in a predetermined

order, and the obtained solution is a local minimum with respect to all the k'_{max} chosen neighborhoods. Note that the number of neighborhoods used in our method is 6 (Line 6 of Algorithm 1). These neighborhoods correspond to the following moves, in this order: Or-opt-2 forward, Or-opt-2 backward, 1-opt, Or-opt-1 backward, Or-opt-1 forward and 2-opt (see Section 3.2). The aim of the optimization phase is to improve the feasible solution found in the constructive phase. The local search procedures used in the second phase are of the best-improvement type. In addition, our method considers only feasible solutions in the optimization phase. Here, k_{max} is a parameter representing a limit on the degree of shaking (perturbation) to be performed. In our method, a shake (line 5 of Algorithm 1) corresponds to performing k feasible Or-opt-1 moves. Finally, t_{max} is the time limit set for our algorithm.

Algorithm 2 shows the *NeighborhoodChange* procedure, where $f(x)$ is the value of the objective function for the solution x. This procedure compares the value of a new solution x' to the value of the incumbent solution x. If the value of x' is better, i.e., $f(x') < f(x)$ in a minimization problem, x' becomes the incumbent solution (Line 2 of Algorithm 2) and k is reset to its initial value (Line 3 of Algorithm 2); otherwise, the value of k is incremented, i.e., we either increase the degree of perturbation or check the next neighborhood.

Algorithm 2: $NeighbourhoodChange(x, x', k)$

```
1 if f(x') < f(x) then
2 |   x ← x';
3 |   k ← 1;
4 else
5 |   k ← k + 1;
6 end
```

Algorithm 3 presents the *VND* procedure. In Line 4, a local search procedure finds a local minimum of the k^{th} neighborhood V_k of the incumbent solution x. We recall that the output of Algorithm 3 is a local minimum with respect to all the k'_{max} neighborhoods considered.

3.4. Constructive phase

In order to build a feasible solution, we follow the broad outline of the constructive phase presented in Da Silva and Urrutia (2010).

When solving the TSPTW with a heuristic, we can represent a solution as an arrangement of all the customers. Besides, we can include the origin depot at the beginning of the list and the destination

Algorithm 3: $VND(x, k'_{max})$

```
repeat
    k ← 1;
    repeat
        Find the best neighbor in V_k(x) : x' ← arg min_{y∈V_k(x)} f(y);
        NeighborhoodChange(x, x', k);
    until k = k'_max;
until no improvement is obtained;
```

depot at the end of the list. Thus, the order of the customers in the list corresponds to the order of visit of the customers. This representation ensures that the solution is a tour that starts at the origin depot and ends at the destination depot (which is the same depot as the origin) by visiting all the n customers exactly once. Therefore, the constraints that we must satisfy to find a feasible solution are only the constraints concerning the time windows. Da Silva and Urrutia (2010) propose a VNS algorithm which aims to minimize the objective function $g(x) = \sum_{i=1}^{n+1} \max\{0, A_i - b_i\}$. This equation represents the sum of the violations of the constraints related to the time windows for solution x. Thus, finding a feasible solution for the TSPTW consists in finding a solution for which the value of the objective function g is equal to zero.

Algorithm 4 shows the constructive phase. For each TSPTW-C instance, the *RandomSolution* procedure generates a random solution x (probably infeasible, i.e., $g(x) > 0$). The *Shake(x,level)* procedure executes randomly chosen *level* Or-opt-1 moves (i.e., the customer is randomly chosen and its new position is also randomly chosen). The *maxLevel* parameter, which defines a limit on the degree of perturbation, is chosen to be equal to 8 (we use the same value as Da Silva and Urrutia (2010)). *OrOpt1ConstructLS(x)* is the most important procedure in the constructive phase. It is a local search procedure of the first-improvement type. The only move used in this procedure is the Or-opt-1. The purpose of a local search procedure of the first-improvement type is to find a local minimum of a neighborhood. Having an incumbent solution x, the LSP evaluates the neighbors of x and makes a change of the incumbent solution as soon as it finds a better solution x^M which becomes the new incumbent solution. This process is repeated until there is no possible improvement, i.e., a local minimum is identified. The particularity of the procedure *OrOpt1ConstructLS(x)* is that it begins by exploring the moves that are more likely to yield a feasible solution. We can partition the set of customers into two subsets: the set of customers visited after the end of their time windows (we call them infeasible customers), and the set of

customers visited during or before their time windows (we call them feasible customers). The Or-opt-1 moves can be as well divided into two types: moving a customer forward, and moving a customer backward. Therefore, we can derive four types of moves: moving an infeasible customer forward, moving an infeasible customer backward, moving a feasible customer forward, and moving a feasible customer backward. Hence, the *OrOpt1ConstructLS* procedure explores the neighborhood corresponding to the Or-opt-1 move in this order: moving an infeasible customer backward; moving a feasible customer forward; moving a feasible customer backward and moving an infeasible customer forward. Da Silva and Urrutia (2010) present a comparison showing that sorting the moves in this order leads to finding a feasible solution more quickly than when the *OrOpt1ConstructLS* procedure executes the Or-opt-1 moves for all the customers without a previous partitioning of the moves.

Algorithm 4: $ConstructivePhase(maxLevel, t_{max})$

```
repeat
    level ← 1;
    x ← RandomSolution();
    x ← OrOpt1ConstructLS(x);
    while x is infeasible and level < maxLevel do
        x' ← Shake(x, level);
        x' ← OrOpt1ConstructLS(x');
        NeighborhoodChange(x, x', level);
    end
    t ← CpuTime();
until x is feasible or t > t_max;
```

After presenting the constructive phase, which allows finding a feasible solution, we present hereafter the optimization phase during which we try to improve that feasible solution. We first present the method that we adapt, due to Mladenović et al. (2013), then our method for the optimization phase.

3.5. The method of Mladenović et al. (2013)

Mladenović et al. (2013) solve the TSPTW-T with a GVNS heuristic. The method consists of two phases. The first phase uses the algorithm of the constructive phase presented in Section 3.4. For the second phase, an algorithm tries to improve the feasible solution found in the first phase, using a GVNS heuristic. The neighborhoods used for the VND are, in the following order: 1-opt, Or-opt-2 backward, Or-opt-2 forward, Or-opt-1 backward, Or-opt-1 forward, and 2-opt. The authors use a vector, called *gap*, that indicates, for each node i of the tour, by how much we can postpone the arrival time at node

i so that the part of the tour (the path) that begins at node i and ends at the destination depot remains feasible. The vector *gap* is defined as follows: $e_i = \min\{e_{i+1} + \max\{0, a_i - A_i\}, b_i - A_i\}$ for $i = 0, ..., n$ with $e_{n+1} = b_{n+1} - A_{n+1}$, and i represents the customer in position i of the tour ($i = 0$ represents the origin depot and $i = n+1$ represents the destination depot). The value e_i is called the *gap* at node i. The local search procedures, which are used to find the local minimum of each considered neighborhood, are of the best-improvement type. For each LSP, the components of the vector *gap* are updated only when there is a change in the incumbent solution.

In the following, we consider that a tour is represented by an ordered sequence $(0, 1, 2, ..., i, ..., n, n+1)$, where i represents the customer in position i. Nevertheless, we use only i to refer to that customer. Having a feasible solution, we explain hereafter how the vector *gap* allows checking the feasibility of a neighbor (that is to say, a solution obtained after performing a move among those mentioned above). The structure of the resulting solution when a move is applied to a feasible solution is shown in Figure 1.

0	1	2	.	.	p-1	p	.	.	q	c*	.	.	n+1

Figure 1: Structure of a solution after a move

The part that contains nodes between 0 and $p-1$ (included) remains unchanged: the positions are occupied by the same customers and the arrival times do not change. The part that contains nodes between p and q is modified: the positions are occupied by other customers and the new arrival times are different. The last part contains nodes whose positions are between c^* and $n+1$, the customers occupying these nodes (positions) are the same, but the arrival times to these customers may have changed. After a move is applied, position c^* (as mentioned above, we also use c^* to designate the customer in position c^*) may be seen as the smallest position j such that the path $(j, j+1, ..., n+1)$ is unchanged with respect to the customers that occupy positions j to $n+1$ (the arrival times to these customers may have changed). Therefore, considering a feasible solution and after a move is applied to this solution, we can check the feasibility of the resulting solution as follows:

- Verify the feasibility of the customers between positions p and q. If at least one customer is infeasible, then the resulting solution is infeasible, and we stop checking feasibility. If all customers between positions p and q are feasible, then we go to the next step;
- Update the arrival time at customer c^* (the old arrival time is denoted by A_{c^*} and the new arrival

time by $A^N_{c^*}$). If $A^N_{c^*} - A_{c^*} > e_{c^*}$, then the resulting solution is infeasible; otherwise, it is feasible.

Thus, the vector *gap* allows verifying the feasibility of a solution without moving through all the tour, i.e, without having to check the feasibility of each of the customers in positions $c^* + 1$ to $n + 1$.

Having an efficient tool for checking the feasibility of a move, the only concern is to verify efficiently the profitability of a move (a move is said to be profitable if it improves the current feasible solution). This task is easy for the TSPTW-T (minimization of the distance or the travel time of the route), but it is not the case for the TSPTW-C. In fact, for the TSPTW-T, one only needs to compare the sum of distances (travel times) of the deleted arcs to the sum of distances (travel times) of the new arcs in order to check the profitability of a move and to compute the new value of the objective function corresponding to the resulting solution. For the case of a 2-opt move, if the distance matrix is not symmetric, then one must be careful about the inverted arcs while verifying the profitability of a move even for the TSPPTW-T variant. On the contrary, checking the profitability of a move is time-consuming for the TSPTW-C. In fact, one needs, in theory, to update the arrival times at all the customers between positions p and $n+1$ to determine if a move is profitable. Yet, we present below, in Section 3.6.3, an efficient method that allows the verification of the profitability of a move for the TSPTW-C without having to update the arrival times at the customers between positions $c^* + 1$ and $n + 1$. In addition, we integrate a basic preprocessing (the method of Mladenović et al. (2013) does not use any preprocessing) that allows removing incompatible arcs. By identifying these incompatible arcs and exploring each neighborhood in a determined order, we can reduce the number of solutions to be visited while ensuring that all the feasible ones are considered.

3.6. The proposed method for the optimization phase

The proposed method for the optimization phase is presented in Algorithm 1 (without considering Line 1). We recall that the procedure *Shake(x,k)* performs k randomly chosen Or-opt-1 (the customer is chosen randomly, and it is moved to a random position) feasible moves in the optimization phase.

3.6.1. Basic preprocessing

We perform a basic preprocessing step that allows identifying incompatible arcs, i.e., arcs that cannot be present in any feasible solution. An arc (i, j) is incompatible if $a_i + t_{ij} > b_j$ (for the notation, see Section 3.1). Our algorithm avoids verifying many infeasible solutions by marking the incompatible arcs. Moreover, since the times t_{ij} are positive and the travel time matrix satisfies the triangle inequality, we can deduce that if an arc (i, j) is incompatible, then every solution in which customer j is visited after

(not necessarily exactly after) customer i is an infeasible solution. We see later that the order of exploration of a neighborhood is crucial to exploit the information derived from incompatible arcs in order to reduce the search space without eliminating feasible solutions.

3.6.2. The order of the neighborhoods

The neighborhoods that we consider in the VND are, in the following order: Or-opt-2 forward, Or-opt-2 backward, 1-opt, Or-opt-1 backward, Or-opt-1 forward and 2-opt. We tested other orders, especially the order proposed in Mladenović et al. (2013): 1-opt, Or-opt-2 backward, Or-opt-2 forward, Or-opt-1 backward, Or-opt-1 forward and 2-opt, but the first order gave the best results.

3.6.3. Checking the profitability

In Section 3.5, we saw that the vector *gap* offers an efficient tool to verify the feasibility of a move. In fact, one only needs to know the new arrival time at customer c^* (the customer that is in the smallest position j such that the path $(j, j+1, ..., n+1)$ is unchanged with respect to the customers occupying positions j to $n+1$). For further explanation, see Figure 1 and Section 3.5. We present hereafter an efficient way to check the profitability of a move. To achieve this, we define a vector, named maximum gain, that is calculated at the same time as the vector *gap*, using the following recurrence: $maxGain_i = \min\{maxGain_{i+1}, A_i - a_i\}$ for $i = 0, ..., n$ with $maxGain_{n+1} = A_{n+1} - a_{n+1}$, where a_i is the start of the time window of customer i, A_i is the arrival time at customer i, and i represents the customer in position i of the tour ($i = 0$ represents the origin depot and $i = n+1$ represents the destination depot). The maximum gain vector can be also expressed, for each node, as follows: $maxGain_i = \min_{j \in \{i, i+1, ..., n+1\}} \{A_j - a_j\}$.

We recall that the LSPs are of the best-improvement type. For each LSP, in the same way as the vector *gap*, the components of the vector *maxGain* are updated only when there is a change in the incumbent solution.

Considering a solution x and applying a move to it, we obtain a resulting solution x^N. In the case of the TSPTW-C, the applied move is profitable if $A_{n+1}(x^N) < A_{n+1}(x)$, where $A_{n+1}(y)$ designates the arrival time at the depot for a solution y.

For the solution x, the arrival time at customer c^* is A_{c^*}. Assuming that we know the new arrival time at customer c^*, i.e., $A^N_{c^*}$, we note that:

- If $A^N_{c^*} \geq A_{c^*}$, then $A_{n+1}(x^N) \geq A_{n+1}(x)$.

- If $A^N_{c^*} < A_{c^*}$ and $maxGain_{c^*} \leq 0$, then $A_{n+1}(x^N) = A_{n+1}(x)$. In fact, all the profit regarding the arrival time at customer c^* is absorbed by the first (with respect to the order of visit that starts at c^* and ends at $n+1$) customer k for which $A_k \leq a_k$.

- If $A^N_{c^*} < A_{c^*}$ and $maxGain_{c^*} > 0$, then $A_{n+1}(x^N) < A_{n+1}(x)$ and we have $A_{n+1}(x^N) = A_{n+1}(x) - \min\{maxGain_{c^*}, A_{c^*} - A^N_{c^*}\}$. In fact, if $maxGain_{c^*} \geq A_{c^*} - A^N_{c^*}$, then all the profit obtained for customer c^* is spread until the depot at $n+1$. However, if $maxGain_{c^*} < A_{c^*} - A^N_{c^*}$, then a part of the profit is absorbed.

Thus, the vector *maxGain* allows checking the profitability of a move efficiently. Besides, when a move is profitable, we can even predict the arrival time at the destination depot $n+1$ without browsing the path $(c^*, ..., n+1)$, and this is important for best-improvement type local search procedures. Note that the method that we propose to verify the profitability of a move does not depend on the move itself. In fact, for any given move, it is sufficient to determine customer c^* (i.e., the customer that is at the smallest position j such that the path $(j, j+1, ..., n+1)$ is unchanged with regard to the customers occupying positions j to $n+1$), to calculate the new arrival time at customer c^*, then to determine the profitability of the move without browsing the path $(c^*, ..., n+1)$. We recall that the vector *gap* (presented in Section 3.5) allows checking the feasibility of the path $(c^*, ..., n+1)$ without browsing it.

3.6.4. The exploration of the neighborhoods

The order in which we explore a neighborhood is of crucial importance for the number of evaluated neighbors and the necessary time to explore all the neighborhood. We explore every neighborhood in a lexicographical order (see Savelsbergh (1992)). The purpose of a local search procedure of the best-improvement type is to find a local minimum of the neighborhood. Having a current feasible solution x, the LSP must evaluate all the neighbors of x. If there are one or more (feasible) neighbors that improve solution x, then the LSP chooses the best solution x^M- in fact, one among the solutions which are equally better - and the solution x^M becomes the current solution. This process is repeated until there is no possible improvement, i.e., a local minimum is identified.

Forward Or-opt-m neighborhoods

As an example, take the neighborhood corresponding to the move forward Or-opt-1, which consists of moving a customer from a position j to a position k such that $j < k$. The LSP must test all forward Or-opt-1 moves for each customer. To do this, several orders are possible. We choose the lexicographical

order (Figure 2), in which the customer in position p is moved to position $p+1$, then to position $p+2$ and so on until position n. This order allows reducing the search space using the information deduced from the incompatible arcs and from the properties of the travel time matrix (the travel times are positive and satisfy the triangle inequality).

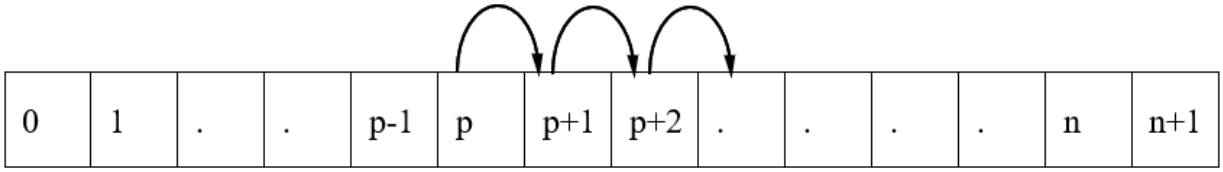

Figure 2: Lexicographical order for forward Or-opt-1

In what follows, we use the notation C_i to indicate customer i and the notation CP_i for the customer in position i. Customer C_i is initially in position p. In the first line of Algorithm 5, j represents the current position of C_i. In Line 2, we identify the customer *SuccessorCi* that is currently the successor of customer C_i. In Line 3, we identify the customer in position c^*. In Line 4, we perform a test concerning incompatible arcs: if the arc *(SuccessorCi,*C_i*)* is incompatible then customer C_i cannot be at a position higher than the position of customer *SuccessorCi* in any feasible solution. In this case, we stop the algorithm without evaluating all the next positions and we then evaluate the moves for another customer (we recall that Algorithm 5 concerns the forward Or-opt-1 moves for only one customer). If the arc *(SuccessorCi,*C_i*)* is compatible, then, in Line 7, we update the new arrival time at *SuccessorCi* (i.e., $A^N_{SuccessorCi}$), which is performed in $\mathcal{O}(1)$. We note that it is useless to check the feasibility of customer *SuccessorCi* because it was feasible before the move and the triangle inequality ensures that it cannot be visited later than in the current feasible solution (in fact, *SuccessorCi* is moved backward). In Line 8, we update, in constant time, the new arrival time at C_i (i.e., $A^N_{C_i}$) and we verify, in Line 9, if it satisfies the constraint of the time window of this customer. If $A^N_{C_i} > b_{C_i}$ (where b_{C_i} is the end of the time window of customer C_i), then we stop the algorithm (and we move on to another customer) because the triangle inequality ensures that the value of $A^N_{C_i}$ cannot decrease in the next positions. If customer C_i is feasible, then Line 12 updates the new arrival time at customer C_{c^*} (i.e., $A^N_{c^*}$). This is performed in $\mathcal{O}(1)$ as well. We can check, in constant time, whether the new solution is feasible or not through the vector *gap* and by having information about the new and the old arrival times to customer C_{c^*}. If the solution turns out to be infeasible, then the algorithm continues to check the next position (Line 1). If the solution is feasible, then we check, in constant time, if the move is profitable through the information about $A^N_{C_{c^*}}$

and the vector *maxGain*. If the move is not profitable, then we pass to the evaluation of the next position (Line 1). If the move turns out to be beneficial, then, before moving to the following position (Line 1) we calculate, in $\mathcal{O}(1)$, the arrival time at the destination depot A_{n+1} and we compare it to the current best arrival time at the depot found by the local search procedure corresponding to forward Or-opt-1 move (we recall that Algorithm 5 is actually a part of a local search procedure of the best-improvement type).

Algorithm 5: Forward Or-opt-1 LSP for one customer

```
1  for j ← p to n − 1 do
2      SuccessorCi ← CP_{j+1};
3      C_{c*} ← CP_{j+2};
4      if (SuccessorCi, C_i) is incompatible then
5          stop;
6      else
7          update A^N_{SuccessorCi};
8          update A^N_{C_i};
9          if C_i is infeasible then
10             stop;
11         else
12             update A^N_{C_{c*}};
13             if the tour is feasible then
14                 check the profitability;
15             end
16         end
17     end
18 end
```

Hence, we deduce that the forward Or-opt-1 neighborhood is explored efficiently. We presented in details the neighborhood exploration corresponding to the forward Or-opt-1 move. Below, we show only the broad lines for the other neighborhoods.

The neighborhoods corresponding to the moves forward Or-opt-2 and 1-opt are explored almost in the same way as the neighborhood corresponding to the forward Or-opt-1 move.

Backward Or-opt-m neighborhoods

The neighborhood corresponding to the backward Or-opt-1 move is explored, for a customer C_i initially in position p, by moving this customer to position $p-1$, then to position $p-2$ and so on until position 1 (Figure 3). Therefore, we notice that position c^* (the smallest position k such that the path

$(k, k+1, ..., n+1)$ is unchanged with respect to the customers occupying the positions k to $n+1$) remains unchanged when we consider customer C_i by using this order. In fact, position c^* is equal to position $p+1$ (Figure 3). Concerning the use of the information deduced from the incompatible arcs, we stop moving customer C_i backward when an incompatible arc appears in the resulting solution. Finally, we note that when the customer is moved, in a given step, to position q, the verification of the feasibility and the profitability of the resulting solution is not performed in $\mathcal{O}(1)$. This verification requires: updating the arrival times of customers that are in positions q to $p+1$; checking the feasibility of the customers that are in positions $q+1$ to p, and the application of the processes of checking the feasibility and the profitability. We note that the triangle inequality ensures the feasibility of the customer that occupies position q - i.e., customer C_i - because it was feasible when it was occupying position p, and it was moved backward. The feasibility of the customer that fills position $p+1$ (that is, position c^*) is checked through the process of the verification of the feasibility of the path $(c^*, ..., n+1)$ described in Section 3.5.

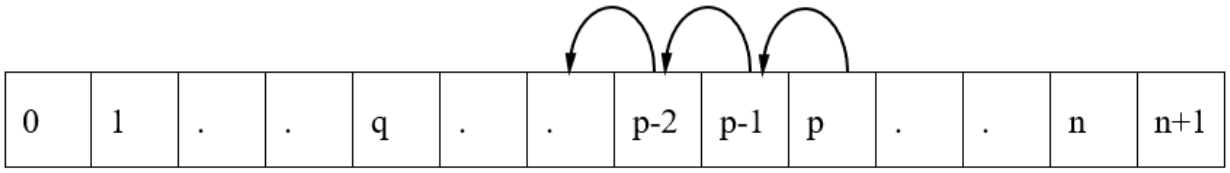

Figure 3: Lexicographical order for backward Or-opt-1

The reasoning concerning the neighborhood corresponding to the backward Or-opt-2 move is almost identical to the reasoning corresponding to the backward Or-opt-1 move.

The 2-opt neighborhood

For the 2-opt neighborhood, we consider in our work the move shown in Figure 4. In the example of Figure 4, we choose to represent the depot only with node 0 to insist on the fact that the resulting solution after a 2-opt move must be a tour. In this Figure, the considered 2-opt move consists in deleting the two arcs $(4,5)$ and $(3,7)$, replacing them by the two arcs $(4,3)$ and $(5,7)$, then reversing the direction of the arcs $(5,1)$ and $(1,3)$. We choose to explore the neighborhood corresponding to the 2-opt move in the lexicographical order explained in Figure 5 (inspired from Savelsbergh (1992)).

In Figure 6, we present the structure of a solution after the application of a 2-opt move. The notation CP_i designates the customer that occupied position i before the application of a 2-opt move. The presented 2-opt move in Figure 6 is the one that deletes the arcs (CP_p, CP_{p+1}) and (CP_q, CP_{q+1}), replaces them by (CP_p, CP_q) and (CP_{p+1}, CP_{q+1}), then reverses the path between the customers CP_{p+1} and CP_q.

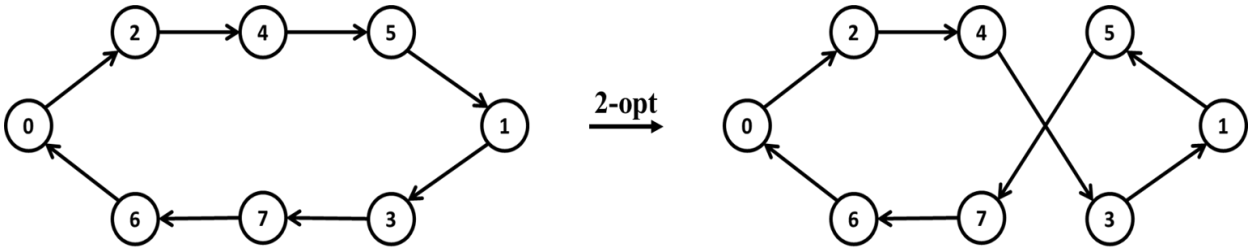

Figure 4: 2-opt move

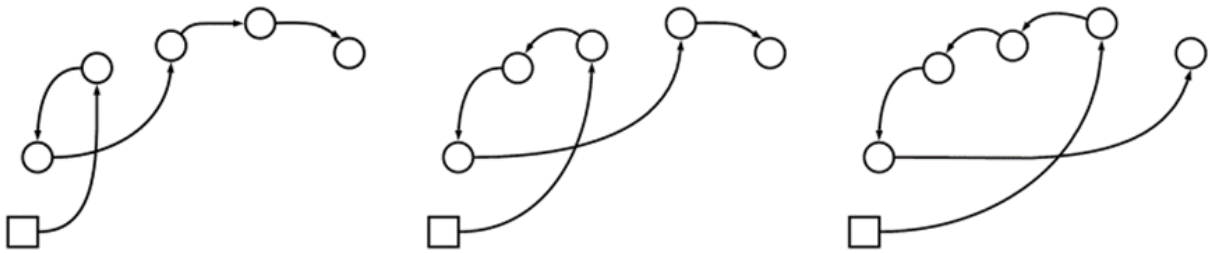

Figure 5: Lexicographical order for 2-opt

Exploring the neighborhood corresponding to the 2-opt move in the lexicographical order (presented in Figure 5) enables to profit from some tests that allow eliminating a set of infeasible solutions without evaluating them (search space reduction). In fact, the lexicographical order can be seen as follows: for each customer CP_p, we want to explore the solutions obtained by the application of the 2-opt move presented in Figure 6, where the position q ranges from $p+2$ to n. Hence, for a given customer CP_p, after a 2-opt move, we must update the arrival times at the customers that occupy the positions $p+1$ to $q+1$ (note that position $q+1$ corresponds to position c^* in every step), then verify the feasibility of the customers occupying positions $p+2$ to q. The customer in position $p+1$ is feasible because the time matrix T satisfies the triangle inequality, and the verification of the feasibility of the customer that occupies position $q+1$ (i.e., position c^*) is included in the process of the verification of the feasibility presented in detail in Section 3.5. If one of the customers occupying the positions $p+2$ to q is infeasible, then we stop the process of browsing for customer CP_p (and we move on to the next customer) because all the following 2-opt moves, that is to say, the next positions q, will lead to infeasible solutions. If all the customers occupying the positions $p+2$ to q are feasible, then we perform the verification of the feasibility and the profitability.

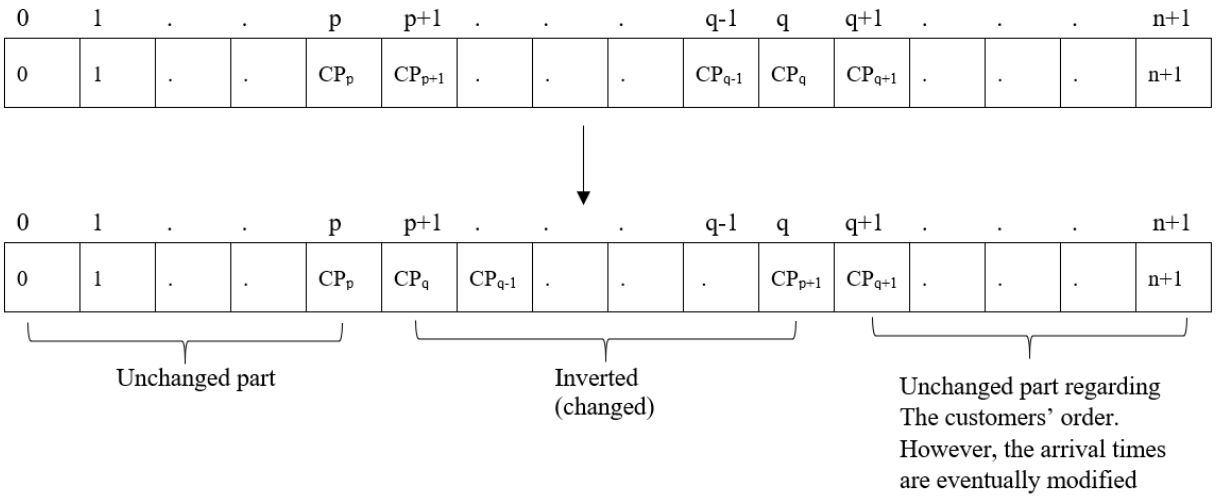

Figure 6: Structure of a solution after a 2-opt move

4. Computational results

In this section, we present the results of the application of our method to well-known instances from the TSPTW literature.

4.1. Brief description of the instances

Multiple sets of instances exist for the TSPTW. These instances and their description can be found on the web page `http://lopez-ibanez.eu/tsptw-instances`. We consider the following sets here:

1. The set of AFG instances consists of 50 asymmetric instances. These are real-world instances introduced in Ascheuer (1995). The number of customers ranges from 10 to 233.
2. The set of DUMAS instances consists of 135 instances grouped into 27 classes (five instances each). For example, the class *n20w40* consists of five instances that contain 20 customers each, and for each customer, the maximum time window width is 40 units. These instances were proposed in Dumas et al. (1995). The number of customers ranges from 20 to 200 customers.
3. The set of GENDREAU instances consists of 130 instances grouped into 26 classes. These instances were presented in Gendreau et al. (1998), and they were obtained from the instances of DUMAS by extending the time windows widths. The number of customers ranges from 20 to 100.
4. The set of OHLMANN instances consists of 25 instances. These instances were presented in Ohlmann and Thomas (2007), and they were also obtained from the instances of DUMAS by extending the time windows widths. The number of customers ranges from 150 to 200.

5. The set of PESANT instances consists of 27 instances that were presented in Pesant et al. (1998). These instances were derived from Solomon's RC2 VRPTW instances presented in Solomon (1987). The number of customers ranges from 19 to 44.
6. The set of POTVIN instances consists of 30 instances that were proposed in Potvin and Bengio (1996). They were also derived from Solomon's RC2 VRPTW instances. The number of customers ranges from 3 to 45.
7. The set of LANGEVIN instances consists of 70 instances grouped into seven classes. They were proposed in Langevin et al. (1993). The number of customers ranges from 20 to 60.

4.2. Results

For all the presented results, the parameter k_{max} (see Algorithm 1) was set to 60. In fact, we tested for the parameter k_{max} the values 30, 45, 60 and 75. After preliminary experiments, we noticed that the best results, in terms of solution quality, were achieved for the values 60 and 75, compared to the values 30 and 45. However, the value of 75 does not improve the results found with 60.

Our program is sequential and executed on an *Intel(R) Core(TM) i7-3770* processor with a 3.40 GHz CPU and 8 MB of cache. We used the C++ programming language. We compare the results obtained by our method to those obtained by the Beam-ACO method (López-Ibáñez et al., 2013) which is known as the state of the art for the TSPTW-C (minimization of the arrival time at the depot). The results presented in López-Ibáñez et al. (2013) were obtained on an *Intel Xeon E5410* processor with a 2.33 GHz CPU and 6 MB of cache. To compare the computational times of the two methods, we use the performance measurements presented by SPEC (Standard Performance Evaluation Corporation), see `http://spec.org/cpu2006/results/cint2006.html`. These measurements indicate that our processor is faster than the one used in López-Ibáñez et al. (2013) by a factor that ranges from 2.33 to 2.71. Hence, we decided to multiply the computational times of our method by 2.5. The time limit t_{max} (see Algorithm 1) was divided by the same coefficient 2.5. Therefore, in every run, the chosen parameter t_{max} is 24 s because the imposed time limit in the Beam-ACO method is 60 s. Note that in the comparison of the times reported in the following tables, we consider that the times are equal for the two methods if the difference does not exceed one second.

For the Beam-ACO method, López-Ibáñez et al. (2013) perform a rigorous tuning process on 20% of the instances of each set of instances (AFG, DUMAS, GENDREAU, OHLMANN, PESANT, POTVIN, and LANGEVIN) in order to find the parameters (or the configurations) that give the best results for that set

of instances. This allows them to determine five configurations. Then, the different configurations identified are tested on the remaining 80% instances. The authors report the results of the best configuration for each set of instances. Regarding our method, it is robust, and the parameter k_{max} is chosen to be equal to 60 for all sets of instances.

The presented results are in the same format as in López-Ibáñez et al. (2013). For each instance, the column ***BestKnown*** indicates the best known value of this instance (or the mean for a class of instances) when the objective is to minimize the arrival time at the destination depot. When not specified, the values of the column ***BestKnown*** are the ones found in López-Ibáñez et al. (2013); another column with the name ***Best*** indicates the best value found by our GVNS method for this instance (or the mean for a class of instances); the column ***%inf*** represents the percentage of times where the Beam-ACO method cannot identify a feasible solution for the instance. This column is not indicated for our GVNS method because it can find a feasible solution, in every run, for all of the tested instances; the columns ***RPDm*** and ***RPDsd*** represent, respectively, the mean and the standard deviation of the relative percent deviation RPD ($RPD = \frac{value - BestKnown}{BestKnown}$, where value represents the returned value by a given run); the columns ***Tm*** and ***Tsd*** are in seconds, and they represent, respectively, the mean and the standard deviation of the computational time that was needed to find the best returned value in a given run. For each instance, we run our method **15 times** as in López-Ibáñez et al. (2013). The notation NR indicates that the results were not reported in López-Ibáñez et al. (2013). The values in bold indicate that, to the best of our knowledge, the results of the given instance for the TSPTW-C are not reported in the literature, and they match, then, the results of our GVNS method (as indicated in López-Ibáñez et al. (2013), this is in itself a contribution to the literature of the TSPTW-C). The values in bold and underlined indicate that our method improved the best known solution for the instance.

Table 1 presents the results of the Beam-ACO and GVNS methods for the AFG instances. As mentioned before, the results reported for Beam-ACO correspond to the best results for this set of instances. Beam-ACO finds the best known solution, in each run, for every instance (the instances that were reported in López-Ibáñez et al. (2013)) except for the instances *rbg193.tw* and *rbg233.tw* where it does not find a feasible solution in, respectively, 6.67% and 20% of the cases. In the other cases, the method finds the best known solutions. The results of ten instances (20% of the AFG instances) were not reported because these instances were used in the tuning process. These results are also not reported on the web page `http://lopezibanez.eu/tsptw-instances` which contains the instances and which is

supposed to be maintained up-to-date by two authors of López-Ibáñez et al. (2013) in order to indicate the best known solutions for the TSPTW-T and the TSPTW-C. We provide the results obtained by our method for these instances. The GVNS method is able to find the best known solution for each instance in each of the 15 runs, except for the instance *rbg172a.tw*, where it finds the best known solution in 13 out of 15 runs. GVNS is also faster than Beam-ACO for the AFG instances (see the results of the instances *rbg021.6.tw*, *rbg172a.tw*, *rbg193.tw*, *rbg201a.tw*, and *rbg233.tw*).

Table 2 shows the results of the Beam-ACO and GVNS methods for the DUMAS instances. The results are grouped by classes of instances (each class contains five instances). Beam-ACO finds the best known solution for all classes (the ones that are reported in López-Ibáñez et al. (2013)) in every run, except for the classes of instances *n150w60* and *n200w40*. We think that there is a typographic error in López-Ibáñez et al. (2013) in the result of the class of instances *n150w60*. The best known value is reported to be equal to 981.4, whereas on the web page maintained by two authors of the article, the value is to 988.6. Note that the best solution found by our method and by Carlton and Barnes (1996) is 988.6. Hence, it is possible that Beam-ACO found the best known solution 988.6 but the authors indicate that Beam-ACO did not find the best known solution. For the class *n200w40*, Beam-ACO does not find a feasible solution in 8% of the runs (which corresponds to 6 out of 75 runs, because each one of the five instances is executed 15 times; the authors did not give information about which instances BEAM-ACO could not solve). Note that, for Beam-ACO, the reported results include the results of the instances used in the tuning process (in that process, the authors used an instance from each class). Our GVNS method is able to find the best known solution for all the instances in every run except for the instance *n60w100.004* for which it fails to find the best known solution. Moreover, GVNS is able to improve the best known solutions for the two classes *n80w40* and *n80w80*. The previous best known values (reported in Carlton and Barnes (1996)) were, respectively, 726 and 715.2. The new best known values identified by our method are 725.6 and 714.6, in other words, the number of instances for which we identified a new best known value is at least equal to two and at most equal to five (because each class contains five instances and the value of the solution of each instance should be integer). GVNS is faster than Beam-ACO for some classes of instances (*n150w40*, *n150w60*, *n200w20*, and *n200w40*).

Table 3 presents the results of the Beam-ACO and GVNS methods for the GENDREAU instances. For the instances reported in López-Ibáñez et al. (2013), the best configuration of Beam-ACO is always able to find the best known solutions. GVNS can find the best known solution for each class of instances,

		Beam-ACO					GVNS				
Instance	**BestKnown**	**%inf**	**RPDm**	**RPDsd**	**Tm**	**Tsd**	**Best**	**RPDm**	**RPDsd**	**Tm**	**Tsd**
rbg010a.tw	-	NR	NR	NR	NR	NR	**3840**	0.00	0.00	0	0
rbg016a.tw	-	NR	NR	NR	NR	NR	**2596**	0.00	0.00	0	0
rbg016b.tw	2094	0.00	0.00	0.00	0	0	2094	0.00	0.00	0	0
rbg017.2.tw	2351	0.00	0.00	0.00	0	0	2351	0.00	0.00	0	0
rbg017a.tw	4296	0.00	0.00	0.00	0	0	4296	0.00	0.00	0	0
rbg017.tw	-	NR	NR	NR	NR	NR	**2351**	0.00	0.00	0	0
rbg019a.tw	2694	0.00	0.00	0.00	0	0	2694	0.00	0.00	0	0
rbg019b.tw	-	NR	NR	NR	NR	NR	**3840**	0.00	0.00	0	0
rbg019c.tw	-	NR	NR	NR	NR	NR	**4536**	0.00	0.00	0	0
rbg019d.tw	3479	0.00	0.00	0.00	0	0	3479	0.00	0.00	0	0
rbg020a.tw	4689	0.00	0.00	0.00	0	0	4689	0.00	0.00	0	0
rbg021.2.tw	4528	0.00	0.00	0.00	0	0	4528	0.00	0.00	0	0
rbg021.3.tw	4528	0.00	0.00	0.00	0	0	4528	0.00	0.00	0	0
rbg021.4.tw	-	NR	NR	NR	NR	NR	**4525**	0.00	0.00	0	0
rbg021.5.tw	4516	0.00	0.00	0.00	0	0	4516	0.00	0.00	0	0
rbg021.6.tw	4492	0.00	0.00	0.00	3	3	4492	0.00	0.00	0	0
rbg021.7.tw	4481	0.00	0.00	0.00	0	0	4481	0.00	0.00	0	0
rbg021.8.tw	4481	0.00	0.00	0.00	0	0	4481	0.00	0.00	0	0
rbg021.9.tw	4481	0.00	0.00	0.00	0	0	4481	0.00	0.00	0	0
rbg021.tw	-	NR	NR	NR	NR	NR	**4536**	0.00	0.00	0	0
rbg027a.tw	5093	0.00	0.00	0.00	0	0	5093	0.00	0.00	0	0
rbg031a.tw	3498	0.00	0.00	0.00	0	0	3498	0.00	0.00	0	0
rbg033a.tw	3757	0.00	0.00	0.00	0	0	3757	0.00	0.00	0	0
rbg034a.tw	3314	0.00	0.00	0.00	0	0	3314	0.00	0.00	0	0
rbg035a.2.tw	3325	0.00	0.00	0.00	0	0	3325	0.00	0.00	0	0
rbg035a.tw	3388	0.00	0.00	0.00	0	0	3388	0.00	0.00	0	0
rbg038a.tw	5699	0.00	0.00	0.00	0	0	5699	0.00	0.00	0	0
rbg040a.tw	5679	0.00	0.00	0.00	0	0	5679	0.00	0.00	0	0
rbg041a.tw	3793	0.00	0.00	0.00	0	0	3793	0.00	0.00	0	0
rbg042a.tw	3260	0.00	0.00	0.00	1	1	3260	0.00	0.00	1	1
rbg048a.tw	9799	0.00	0.00	0.00	0	0	9799	0.00	0.00	0	0
rbg049a.tw	13257	0.00	0.00	0.00	0	0	13257	0.00	0.00	0	0
rbg050a.tw	-	NR	NR	NR	NR	NR	**12050**	0.00	0.00	0	0
rbg050b.tw	11957	0.00	0.00	0.00	1	1	11957	0.00	0.00	0	0
rbg050c.tw	10985	0.00	0.00	0.00	0	0	10985	0.00	0.00	0	0
rbg055a.tw	6929	0.00	0.00	0.00	0	0	6929	0.00	0.00	0	0
rbg067a.tw	10331	0.00	0.00	0.00	0	0	10331	0.00	0.00	0	0
rbg086a.tw	16899	0.00	0.00	0.00	0	0	16899	0.00	0.00	0	0
rbg092a.tw	-	NR	NR	NR	NR	NR	**12501**	0.00	0.00	0	0
rbg125a.tw	14214	0.00	0.00	0.00	0	0	14214	0.00	0.00	0	0
rbg132.2.tw	18524	0.00	0.00	0.00	0	0	18524	0.00	0.00	0	0
rbg132.tw	18524	0.00	0.00	0.00	0	0	18524	0.00	0.00	0	0
rbg152.3.tw	-	NR	NR	NR	NR	NR	**17455**	0.00	0.00	0	0
rbg152.tw	17455	0.00	0.00	0.00	0	0	17455	0.00	0.00	0	0
rbg172a.tw	17783	0.00	0.00	0.00	18	19	17783	0.00	0.00	11	13
rbg193.2.tw	21401	0.00	0.00	0.00	0	0	21401	0.00	0.00	0	0
rbg193.tw	21401	6.67	0.00	0.00	16	14	21401	0.00	0.00	0	0
rbg201a.tw	21380	0.00	0.00	0.00	8	7	21380	0.00	0.00	0	0
rbg233.2.tw	26143	0.00	0.00	0.00	0	0	26143	0.00	0.00	0	0
rbg233.tw	26143	20.00	0.00	0.00	14	9	26143	0.00	0.00	0	0

Table 1: Results for AFG instances

whereas, for five groups of instances (for six instances exactly), it is able to find the best known value but not in every run. We also provide the results of five groups of instances for the first time, to the best of our knowledge, in the literature of the TSPTW-C. Note that the reported results for Beam-ACO contain the results of the instances that were used in the tuning process. The necessary time to identify the best

		Beam-ACO					GVNS				
Instance	**BestKnown**	**%inf**	**RPDm**	**RPDsd**	**Tm**	**Tsd**	**Best**	**RPDm**	**RPDsd**	**Tm**	**Tsd**
n20w20	370.4	0.00	0.00	0.00	0	0	370.4	0.00	0.00	0	0
n20w40	342.8	0.00	0.00	0.00	0	0	342.8	0.00	0.00	0	0
n20w60	362.0	0.00	0.00	0.00	0	0	362.0	0.00	0.00	0	0
n20w80	363.4[a]	NR	NR	NR	NR	NR	363.4	0.00	0.00	0	0
n20w100	331.6[a]	NR	NR	NR	NR	NR	331.6	0.00	0.00	0	0
n40w20	521.2	0.00	0.00	0.00	0	0	521.2	0.00	0.00	0	0
n40w40	512.2	0.00	0.00	0.00	0	0	512.2	0.00	0.00	0	0
n40w60	481.4	0.00	0.00	0.00	0	0	481.4	0.00	0.00	0	0
n40w80	486.6	0.00	0.00	0.00	0	0	486.6	0.00	0.00	0	0
n40w100	463.0	0.00	0.00	0.00	0	0	463.0	0.00	0.00	0	0
n60w20	626.8	0.00	0.00	0.00	0	0	626.8	0.00	0.00	0	0
n60w40	654.4[a]	NR	NR	NR	NR	NR	654.4	0.00	0.00	0	0
n60w60	672.8	0.00	0.00	0.00	0	0	672.8	0.00	0.00	0	0
n60w80	628.2	0.00	0.00	0.00	0	0	628.2	0.00	0.00	0	0
n60w100	620.2	0.00	0.00	0.00	0	0	620.6	0.06	0.00	0	0
n80w20	748.2	0.00	0.00	0.00	0	0	748.2	0.00	0.00	0	0
n80w40	726.0[a]	NR	NR	NR	NR	NR	**725.6**	-0.05	0.00	0	0
n80w60	712.6	0.00	0.00	0.00	0	0	712.6	0.00	0.00	0	0
n80w80	715.2[a]	NR	NR	NR	NR	NR	**714.6**	-0.08	0.00	0	0
n100w20	823.0	0.00	0.00	0.00	0	0	823.0	0.00	0.00	0	0
n100w40	821.0	0.00	0.00	0.00	0	0	821.0	0.00	0.00	0	0
n100w60	817.2	0.00	0.00	0.00	0	0	817.2	0.00	0.00	0	0
n150w20	978.4	0.00	0.00	0.00	1	1	978.4	0.00	0.00	0	0
n150w40	990.4	0.00	0.00	0.00	4	4	990.4	0.00	0.00	0	0
n150w60	988.6	0.00	0.73	0.00	4	4	988.6	0.00	0.00	1	0
n200w20	1137.8	0.00	0.00	0.00	7	8	1137.8	0.00	0.00	1	1
n200w40	1156.0	8.00	0.00	0.00	15	13	1156.0	0.00	0.00	1	1

[a]Reported in Carlton and Barnes (1996)

Table 2: Results for DUMAS instances

solutions for each method is almost the same (as mentioned above, the differences within one second are irrelevant).

Table 4 presents the results of the Beam-ACO and GVNS methods for the OHLMANN instances. Beam-ACO finds the best known solution for all the instances that were reported in López-Ibáñez et al. (2013) (the results of 20% of the instances were not published because these instances were used in the tuning process). The authors do not indicate if the best known solution was found in every run (the value of the average of the relative error is reported with a precision of 0.01 and, hence, we cannot ensure if the best known solutions were found in every run). Nevertheless, we notice that the results of Beam-ACO are good for the OHLMANN instances. Considering the reported instances in López-Ibáñez et al. (2013) for which we can compare the two methods, GVNS is able to find the best known solution for all instances in every run, except for the instance *n200w120.1* where GVNS can find the best known solution in 13 out of 15 runs. GVNS is faster than Beam-ACO for the instances *200w120.5*, *n200w140.3* and *n200w140.4*, whereas Beam-ACO is faster than GVNS for the instances *n150w120.1*, *n200w120.1*,

		Beam-ACO					GVNS				
Instance	**BestKnown**	**%inf**	**RPDm**	**RPDsd**	**Tm**	**Tsd**	**Best**	**RPDm**	**RPDsd**	**Tm**	**Tsd**
n20w120	319.6	0.00	0.00	0.00	0	0	319.6	0.00	0.00	0	0
n20w140	286.2	0.00	0.00	0.00	0	0	286.2	0.00	0.00	0	0
n20w160	311.4	0.00	0.00	0.00	0	0	311.4	0.00	0.00	0	0
n20w180	-	NR	NR	NR	NR	NR	**311.2**	0.00	0.00	0	0
n20w200	281.8	0.00	0.00	0.00	0	0	281.8	0.00	0.00	0	0
n40w120	470.6	0.00	0.00	0.00	0	0	470.6	0.00	0.00	0	0
n40w140	458.2	0.00	0.00	0.00	0	0	458.2	0.00	0.00	0	0
n40w160	-	NR	NR	NR	NR	NR	**426.8**	0.00	0.00	0	0
n40w180	427.4	0.00	0.00	0.00	0	0	427.4	0.00	0.00	0	0
n40w200	-	NR	NR	NR	NR	NR	**412.0**	0.00	0.00	0	0
n60w120	573.8	0.00	0.00	0.00	0	0	573.8	0.05	0.08	1	1
n60w140	600.0	0.00	0.00	0.00	0	0	600.0	0.00	0.00	0	0
n60w160	619.6	0.00	0.00	0.00	0	0	619.6	0.00	0.00	0	0
n60w180	576.0	0.00	0.00	0.00	0	0	576.0	0.00	0.00	0	0
n60w200	570.2	0.00	0.00	0.00	0	0	570.2	0.02	0.02	0	1
n80w100	-	NR	NR	NR	NR	NR	**711.2**	0.00	0.00	0	0
n80w120	-	NR	NR	NR	NR	NR	**697.4**	0.01	0.01	1	1
n80w140	672.8	0.00	0.00	0.00	0	0	672.8	0.00	0.00	1	1
n80w160	653.6	0.00	0.00	0.00	1	1	653.6	0.23	0.24	2	2
n80w180	656.4	0.00	0.00	0.00	1	1	656.4	0.05	0.09	1	1
n80w200	646.2	0.00	0.00	0.00	1	0	646.2	0.00	0.00	2	3
n100w80	805.8	0.00	0.00	0.00	0	0	805.8	0.00	0.00	0	0
n100w100	795.8	0.00	0.00	0.00	0	0	795.8	0.00	0.00	0	0
n100w120	895.4	0.00	0.00	0.00	0	0	895.4	0.00	0.00	0	0
n100w140	906.4	0.00	0.00	0.00	0	0	906.4	0.00	0.00	0	0
n100w160	865.0	0.00	0.00	0.00	0	0	865.0	0.00	0.00	0	0

Table 3: Results for GENDREAU instances

n200w120.3, and *n200w140.1*. For the five instances that are not reported in López-Ibáñez et al. (2013), we provide the results of the best known values for the first time in the literature. Note that GVNS finds the best known value for the instance *n150w160.2* in 11 out of 15 runs and fails to find the best known solution for the instance *n150w120.3*. Note that the best known value for the instance *n150w120.3* was identified by our method but with another order of the neighborhoods in the VND procedure.

Table 5 presents the results of the Beam-ACO and GVNS methods for the PESANT instances. Beam-ACO finds the best known solution, in each run, for every instance (the results of the instances that were used in the tuning process were not reported in López-Ibáñez et al. (2013)). The GVNS method can find the best known solution for each instance in each of the 15 runs, except for the instances *rc204.2*, *rc205.3* and *rc208.0*, where our method finds the best known solutions in, respectively, 6, 13 and 12 runs (out of 15 runs). We report the results of five instances for the first time, to the best of our knowledge, in the literature of the TSPTW-C. GVNS is faster than Beam-ACO for the two instances *rc208.0* (but, as mentioned before, GVNS does not find the best known solution in 3 out of 15 runs for this instance) and *rc208.1*.

Table 6 presents the results of the Beam-ACO and GVNS methods for the POTVIN instances. Beam-

		Beam-ACO					GVNS				
Instance	**BestKnown**	**%inf**	**RPDm**	**RPDsd**	**Tm**	**Tsd**	**Best**	**RPDm**	**RPDsd**	**Tm**	**Tsd**
n150w120.1	972	0.00	0.00	0.00	1	1	972	0.00	0.00	3	3
n150w120.2	917	0.00	0.00	0.00	1	2	917	0.00	0.00	1	0
n150w120.3	-	NR	NR	NR	NR	NR	**910**[a]	0.29	0.28	11	11
n150w120.4	925	0.00	0.00	0.00	1	1	925	0.00	0.00	2	2
n150w120.5	907	0.00	0.00	0.00	1	1	907	0.00	0.00	2	1
n150w140.1	-	NR	NR	NR	NR	NR	**1008**	0.00	0.00	1	0
n150w140.2	-	NR	NR	NR	NR	NR	**1020**	0.00	0.00	3	3
n150w140.3	844	0.00	0.00	0.00	0	0	844	0.00	0.00	1	0
n150w140.4	898	0.00	0.00	0.00	1	1	898	0.00	0.00	1	1
n150w140.5	926	0.00	0.00	0.00	0	0	926	0.00	0.00	1	0
n150w160.1	959	0.00	0.00	0.00	1	1	959	0.00	0.00	1	0
n150w160.2	-	NR	NR	NR	NR	NR	**890**	0.03	0.05	18	15
n150w160.3	934	0.00	0.00	0.00	1	1	934	0.00	0.00	1	0
n150w160.4	-	NR	NR	NR	NR	NR	**912**	0.00	0.00	1	0
n150w160.5	920	0.00	0.00	0.00	0	0	920	0.00	0.00	1	0
n200w120.1	1089	0.00	0.00	0.00	9	8	1089	0.28	1.04	13	11
n200w120.2	1072	0.00	0.00	0.00	1	2	1072	0.00	0.00	2	1
n200w120.3	1128	0.00	0.00	0.00	4	5	1128	0.00	0.00	7	6
n200w120.4	1072	0.00	0.00	0.00	4	5	1072	0.00	0.00	3	1
n200w120.5	1073	0.00	0.00	0.00	5	5	1073	0.00	0.00	2	0
n200w140.1	1138	0.00	0.00	0.00	12	12	1138	0.00	0.00	19	17
n200w140.2	1087	0.00	0.00	0.00	4	5	1087	0.00	0.00	3	0
n200w140.3	1083	0.00	0.00	0.00	12	11	1083	0.00	0.00	5	3
n200w140.4	1100	0.00	0.00	0.00	12	8	1100	0.00	0.00	10	8
n200w140.5	1121	0.00	0.00	0.00	5	9	1121	0.00	0.00	5	2

[a]Our method identified a best known solution with the value 909 but with a different order of the neighborhoods in the VND

Table 4: Results for OHLMANN instances

ACO finds the best known solution, in each run, for every instance reported (20% of the POTVIN instances were not reported in López-Ibáñez et al. (2013) because they were used in the tuning process). The GVNS method is able to find the best known solution for each instance in each of the 15 runs, except for the instances *rc204.1* and *rc208.1* for which our method finds the best known solution in 13 out of 15 runs and in 1 out of 15 runs respectively. Our method finds the best known solution of the instance *rc206.4* in 12 out of 15 runs, but this instance was not reported in López-Ibáñez et al. (2013), so we cannot compare the two methods for this instance. Note that for that instance, i.e., *rc206.4*, our method improves the best known solution in the literature which was found in Cheng and Mao (2007). In addition to the instance *rc206.4*, our method improves the best known solution for the instances *rc202.4*, *rc203.4*, and *rc204.2*. For the last two instances, the solutions identified by our method are optimal because the lower bounds, reported in Cheng and Mao (2007), are equal to 338.52 and 690.06, respectively. GVNS is faster than Beam-ACO for four instances and slower than Beam-ACO for one instance.

Table 7 presents the results of our GVNS method for the LANGEVIN instances. The results are grouped in classes of 10 instances each. López-Ibáñez et al. (2013) state that they tested their Beam-

		Beam-ACO					GVNS				
Instance	**BestKnown**	**%inf**	**RPDm**	**RPDsd**	**Tm**	**Tsd**	**Best**	**RPDm**	**RPDsd**	**Tm**	**Tsd**
rc201.0	853.71	0.00	0.00	0.00	0	0	853.71	0.00	0.00	0	0
rc201.1	850.48	0.00	0.00	0.00	0	0	850.48	0.00	0.00	0	0
rc201.2	883.97	0.00	0.00	0.00	0	0	883.97	0.00	0.00	0	0
rc201.3	722.43	0.00	0.00	0.00	0	0	722.43	0.00	0.00	0	0
rc202.0	850.48	0.00	0.00	0.00	0	0	850.48	0.00	0.00	0	0
rc202.1	702.28	0.00	0.00	0.00	0	0	702.28	0.00	0.00	0	0
rc202.2	853.71	0.00	0.00	0.00	0	0	853.71	0.00	0.00	0	0
rc202.3	883.97	0.00	0.00	0.00	0	0	883.97	0.00	0.00	0	0
rc203.0	-	NR	NR	NR	NR	NR	**870.52**	0.00	0.00	0	0
rc203.1	850.48	0.00	0.00	0.00	0	0	850.48	0.00	0.00	0	0
rc203.2	-	NR	NR	NR	NR	NR	**853.71**	0.00	0.00	0	0
rc204.0	839.24	0.00	0.00	0.00	0	0	839.24	0.00	0.00	0	0
rc204.1	492.60	0.00	0.00	0.00	0	0	492.60	0.00	0.00	0	0
rc204.2	870.52	0.00	0.00	0.00	1	1	870.52	2.65	2.24	0	1
rc205.0	834.62	0.00	0.00	0.00	0	0	834.62	0.00	0.00	0	0
rc205.1	899.24	0.00	0.00	0.00	0	0	899.24	0.00	0.00	0	0
rc205.2	908.79	0.00	0.00	0.00	0	0	908.79	0.00	0.00	0	0
rc205.3	684.21	0.00	0.00	0.00	0	0	684.21	0.31	0.81	0	0
rc206.0	-	NR	NR	NR	NR	NR	**893.21**	0.00	0.00	0	0
rc206.1	756.45	0.00	0.00	0.00	0	0	756.45	0.00	0.00	1	2
rc206.2	776.19	0.00	0.00	0.00	3	3	776.19	0.00	0.00	3	8
rc207.0	-	NR	NR	NR	NR	NR	**847.63**	0.00	0.00	0	0
rc207.1	785.37	0.00	0.00	0.00	0	0	785.37	0.00	0.00	0	0
rc207.2	-	NR	NR	NR	NR	NR	**650.80**	0.00	0.00	0	0
rc208.0	836.04	0.00	0.00	0.00	18	15	836.04	0.04	0.09	11	8
rc208.1	615.51	0.00	0.00	0.00	14	12	615.51	0.00	0.00	0	0
rc208.2	596.21	0.00	0.00	0.00	1	1	596.21	0.00	0.00	0	0

Table 5: Results for PESANT instances

ACO method on the LANGEVIN instances, but do not report the results in their article. Thus, we compare our method to the best known solutions reported either in Langevin et al. (1993) or on the web page `http://lopez-ibanez.eu/tsptw-instances` (visited June 17th, 2019). Our method finds the best known solution for all the 70 instances. In addition, there is no variability in the quality of the solutions obtained by our method, that is to say, GVNS finds the best known solution in each run for every instance. Besides, GVNS is able to find the best known solution in less than one second, in each run, for every instance. Note that we did not report the results of our method concerning the columns *RPDsd*, *Tm*, and *Tsd* because they are all equal to zero for every instance.

Summarizing the results of the comparison (based on the instances reported in López-Ibáñez et al. (2013)), regarding the quality of the obtained solutions, the frequency of finding feasible solutions and the necessary time to find the best known solutions: GVNS performs better than Beam-ACO for the group of instances AFG; GVNS and Beam-ACO are competitive for the sets of instances DUMAS, OHLMANN, and PESANT; GVNS performs worse than Beam-ACO for the sets of instances GENDREAU and POTVIN (regarding the percentage of runs in which we find the best known solution); GVNS is ro-

		Beam-ACO					GVNS				
Instance	**BestKnown**	**%inf**	**RPDm**	**RPDsd**	**Tm**	**Tsd**	**Best**	**RPDm**	**RPDsd**	**Tm**	**Tsd**
rc201.1	592.06	0.00	0.00	0.00	0	0	592.06	0.00	0.00	0	0
rc201.2	860.17	0.00	0.00	0.00	0	0	860.17	0.00	0.00	0	0
rc201.3	853.71[a]	NR	NR	NR	NR	NR	853.71	0.00	0.00	0	0
rc201.4	889.18	0.00	0.00	0.00	0	0	889.18	0.00	0.00	0	0
rc202.1	850.48	0.00	0.00	0.00	0	0	850.48	0.00	0.00	0	0
rc202.2	338.52	0.00	0.00	0.00	0	0	338.52	0.00	0.00	0	0
rc202.3	894.10	0.00	0.00	0.00	0	0	894.10	0.00	0.00	0	0
rc202.4	854.12[b]	NR	NR	NR	NR	NR	**853.71**	-0.05	0.00	0	0
rc203.1	488.42	0.00	0.00	0.00	0	0	488.42	0.00	0.00	0	0
rc203.2	853.71	0.00	0.00	0.00	0	0	853.71	0.00	0.00	0	0
rc203.3	921.44	0.00	0.00	0.00	0	0	921.44	0.00	0.00	0	0
rc203.4	350.83[b]	NR	NR	NR	NR	NR	**338.52**	-3.51	0.00	0	0
rc204.1	917.83	0.00	0.00	0.00	27	21	917.83	0.11	0.28	8	14
rc204.2	701.62[a]	NR	NR	NR	NR	NR	**690.06**	-1.65	0.00	0	0
rc204.3	455.03	0.00	0.00	0.00	0	0	455.03	0.00	0.00	0	0
rc205.1	417.81	0.00	0.00	0.00	0	0	417.81	0.00	0.00	0	0
rc205.2	820.19	0.00	0.00	0.00	0	0	820.19	0.00	0.00	0	0
rc205.3	950.05	0.00	0.00	0.00	0	0	950.05	0.00	0.00	0	0
rc205.4	837.71	0.00	0.00	0.00	0	0	837.71	0.00	0.00	0	0
rc206.1	117.85	0.00	0.00	0.00	0	0	117.85	0.00	0.00	0	0
rc206.2	870.49	0.00	0.00	0.00	0	0	870.49	0.00	0.00	4	9
rc206.3	650.59[a]	NR	NR	NR	NR	NR	650.59	0.00	0.00	0	0
rc206.4	930.10[b]	NR	NR	NR	NR	NR	**911.98**	-1.83	0.25	4	8
rc207.1	804.67	0.00	0.00	0.00	1	1	804.67	0.00	0.00	1	1
rc207.2	713.90	0.00	0.00	0.00	0	0	713.90	0.00	0.00	0	0
rc207.3	745.77	0.00	0.00	0.00	3	4	745.77	0.00	0.00	0	0
rc207.4	133.14	0.00	0.00	0.00	0	0	133.14	0.00	0.00	0	0
rc208.1	810.70	0.00	0.00	0.00	4	4	810.70	0.04	0.01	0	0
rc208.2	579.51	0.00	0.00	0.00	9	8	579.51	0.00	0.00	0	0
rc208.3	686.80	0.00	0.00	0.00	1	1	686.80	0.00	0.00	0	0

[a] Reported in Cheng and Mao (2007)
[b] Reported in Karabulut and Tasgetiren (2014)

Table 6: Results for POTVIN instances

bust, the same parameters were used for all sets of instances, while the results reported by Beam-ACO correspond to different configurations depending on the set of instances considered; and GVNS always finds a feasible solution, while Beam-ACO fails to find feasible solutions for some instances with relatively narrow time windows.

We have summarized the main differences between the Beam-ACO method and our GVNS method. Overall, we conclude that our method is at least as good as the state-of-the-art method for the TSPTW-C.

5. Conclusion

We adapted and improved an existing method, which gave good results for the TSPTW-T (minimizing the travel time), to solve the TSPTW-C (minimizing the completion time). Our method proceeds in

		GVNS	
Instance	**BestKnown**	**Best**	**RPDm**
N20ft30	730.8[a]	730.78	-
N20ft40	730.0[a]	730.00	-
N40ft20	999.2[a]	999.20	-
N40ft40	996.21[b]	996.21	0.00
N60ft20	1248.88[b]	1248.88	0.00
N60ft30	1247.31[b]	1247.31	0.00
N60ft40	1244.73[b]	1244.73	0.00

[a]Optimal value reported in Langevin et al. (1993). We don't report the value of the *RPDm* column because the optimal value was reported with a precision of only 0.1

[b]Reported in `http://lopez-ibanez.eu/tsptw-instances` (visited June 17th, 2019)

Table 7: Results for LANGEVIN instances

two phases. The first phase aims to find a feasible solution. In the second phase, we improve, using a GVNS-based heuristic, the solution found in the first phase. We introduced an efficient way to check the profitability of a move, and we explained how to reduce the search space without ignoring any feasible solution. We tested our method on 467 well-known instances from the literature on the TSPTW. We improved the best known solutions of, at least, 6 instances (and at most 9 instances). In addition to the new best known solutions that we found, we provided, to the best of our knowledge, the results of 45 instances for the first time in the literature of the TSPTW-C. Our method is robust and the results show that it is at least as good as the method considered as the state of the art.

References

Ascheuer, N., 1995. Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. Ph.D. thesis, Technische Universität Berlin, Germany.

Baker, E. K., 1983. Technical note-an exact algorithm for the time-constrained traveling salesman problem. Operations Research 31 (5), 938–945.

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.

Blum, C., 2005. Beam-ACO-Hybridizing ant colony optimization with beam search: An application to open shop scheduling. Computers & Operations Research 32 (6), 1565–1591.

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

Carlton, W. B., Barnes, J. W., 1996. Solving the traveling-salesman problem with time windows using tabu search. IIE transactions 28 (8), 617–630.

Cheng, C.-B., Mao, C.-P., 2007. A modified ant colony system for solving the travelling salesman problem with time windows. Mathematical and Computer Modelling 46 (9), 1225–1235.

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

Da Silva, R. F., Urrutia, S., 2010. A General VNS heuristic for the traveling salesman problem with time windows. Discrete Optimization 7 (4), 203–211.

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

Favaretto, D., Moretti, E., Pellegrini, P., 2006. An ant colony system approach for variants of the traveling salesman problem with time windows. Journal of Information and Optimization Sciences 27 (1), 35–54.

Focacci, F., Lodi, A., Milano, M., 2002. A hybrid exact algorithm for the TSPTW. INFORMS Journal on Computing 14 (4), 403–417.

Funke, B., Grünert, T., Irnich, S., 2005. Local search for vehicle routing and scheduling problems: Review and conceptual integration. Journal of Heuristics 11 (4), 267–306.

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

Hansen, P., Mladenović, N., Pérez, J. A. M., 2008. Variable neighbourhood search: methods and applications. 4OR 6 (4), 319–360.

Ilavarasi, K., Joseph, K. S., 2014. Variants of travelling salesman problem: A survey. In: Information Communication and Embedded Systems (ICICES), 2014 International Conference on. IEEE, pp. 1–7.

Karabulut, K., Tasgetiren, M. F., 2014. A variable iterated greedy algorithm for the traveling salesman problem with time windows. Information Sciences 279, 383–395.

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.

Lin, S., 1965. Computer solutions of the traveling salesman problem. The Bell System Technical Journal 44 (10), 2245–2269.

Lin, S., Kernighan, B. W., 1973. An effective heuristic algorithm for the traveling-salesman problem. Operations Research 21 (2), 498–516.

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.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Computers & Operations Research 24 (11), 1097–1100.

Mladenović, N., Todosijević, R., Urošević, D., 2013. An efficient general variable neighborhood search for large travelling salesman problem with time windows. Yugoslav Journal of Operations Research 23 (1), 19–30.

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.

Pesant, G., Gendreau, M., Potvin, J.-Y., Rousseau, J.-M., 1998. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. Transportation Science 32 (1), 12–29.

Potvin, J.-Y., Bengio, S., 1996. The vehicle routing problem with time windows part II: genetic search. INFORMS Journal on Computing 8 (2), 165–172.

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

Savelsbergh, M. W., 1992. The vehicle routing problem with time windows: Minimizing route duration. ORSA Journal on Com-

puting 4 (2), 146–154.

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