Contents lists available at ScienceDirect

Discrete Optimization

journal homepage: www.elsevier.com/locate/disopt

A General VNS heuristic for the traveling salesman problem with time windows

Rodrigo Ferreira da Silva[*], Sebastián Urrutia

Department of Computer Science, Universidade Federal de Minas Gerais, Av. Antônio Carlos 6627, Belo Horizonte, MG, 31270-010, Brazil

ARTICLE INFO

Article history:
Received 1 June 2009
Received in revised form 21 April 2010
Accepted 22 April 2010
Available online 4 June 2010

Keywords:
Traveling salesman
Time windows
Heuristics
Variable neighborhood search
Variable neighborhood descent

ABSTRACT

This paper presents a General Variable Neighborhood Search (GVNS) heuristic for the Traveling Salesman Problem with Time Windows (TSPTW). The heuristic is composed by both constructive and optimization stages. In the first stage, the heuristic constructs a feasible solution using VNS, and in the optimization stage the heuristic improves the feasible solution with a General VNS heuristic. Both constructive and optimization stages take advantage of elimination tests, partial neighbor evaluation and neighborhood partitioning techniques. Experimental results show that this approach is efficient, reducing significantly the computation time and improving some best known results from the literature.

1. Introduction

The *Traveling Salesman Problem with Time Windows* (TSPTW) consists in finding a minimum cost tour starting and ending on a given depot and visiting a set of customers. Each customer has a service time (i.e. the time that must be spent at the customer) and a time window defining its ready time and due date. Every customer must be visited before its due date, therefore every tour visiting a customer after its due date is considered infeasible. If the vehicle arrives before the customer ready time, it must wait. The cost of a tour is the total distance traveled.

The TSPTW can be formulated on a graph $G = (V, A)$, where $V = \{1, 2, \ldots, n\}$ is the set of customers, 0 represents the depot, and $A = \{(i, j) : i, j \in V \cup \{0\}, i \neq j\}$ is the set of arcs between the customers. The cost of traveling from i to j is represented by c_{ij} that includes both the service time of customer i and the time needed to travel from i to j. Each customer i has an associated time window $[a_i, b_i]$, where a_i and b_i represent the ready time and due date, respectively. The problem consists in finding a Hamiltonian tour starting from the depot, that satisfies all time windows and minimizes the total distance traveled.

Let x_{ij} be equal to 1 if and only if customer j (or the depot in case $j = 0$) is visited immediately after visiting customer i (or the depot in case $i = 0$) and 0 otherwise. Let β_i be the time customer i is visited and let M be a big constant. The TSPTW may be formulated as an integer programming problem as follows:

$$\text{Minimize} \quad \sum_{i \in V \cup \{0\}} \sum_{j \in V \cup \{0\}} c_{ij} x_{ij}$$

subject to:

$$\sum_{i \in V \cup \{0\}, i \neq j} x_{ij} = 1 \quad \forall j \in V \cup \{0\} \tag{1}$$

$$\sum_{j \in V \cup \{0\}, j \neq i} x_{ij} = 1 \quad \forall i \in V \cup \{0\} \tag{2}$$

[*] Corresponding author.
E-mail address: rfsilva@dcc.ufmg.br (R.F. da Silva).

1572-5286/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.disopt.2010.04.002

$$\beta_j \geq \beta_i + c_{ij} - M(1 - x_{ij}) \quad \forall i, j \in V \cup \{0\},\ j \neq 0 \tag{3}$$
$$a_i \leq \beta_i \leq b_i \quad \forall i \in V \tag{4}$$
$$x_{ij} \in \{0, 1\} \quad \forall i, j \in V \cup \{0\}$$
$$\beta_i \in \mathbb{R}_{\geq 0} \quad \forall i \in V \cup \{0\},$$

The objective function minimizes the cost of the tour. Constraints (1) and (2) state that every customer and the depot must be visited exactly once. Constraints (3) ensure that the arrival time to a given customer cannot be smaller than the arrival time to the customer visited immediately before plus the cost of traveling between the two customers. Since all costs are positive constraints (3) avoid the formation of sub-cycles. Finally, constraints (4) impose the time windows.

The TSPTW has diverse practical applications in routing and scheduling problems. In particular, the TSPTW can be viewed as a subproblem of the *Vehicle Routing Problem with Time Windows* (VRPTW), considering the relaxation of the capacity constraints.

The literature contains some exact algorithms to the TSPTW. Langevin et al. [12] introduce a two-commodity flow formulation which can be extended to the makespan problem. Dumas et al. [4] use a dynamic programming approach with elimination tests, enhancing the performance and reducing the search space, both a priori and during its execution. Pesant et al. [15] propose a branch-and-bound algorithm using a constraint programming model. Similarly, Focacci et al. [5] propose a hybrid approach based on constraint programming and optimization techniques.

Due to the problem's complexity (NP-Hard, since it has the *Traveling Salesman Problem* as a special case) and the limitations of exact approaches to find optimal solutions in a reasonable amount of time when considering realistic instances, there is a need for effective heuristics. Carlton and Barnes [3] use a tabu-search heuristic with a static penalty function, that considers infeasible solutions. Gendreau et al. [7] propose a insertion heuristic based on GENIUS [6] that gradually builds the route and improves it in a post-optimization phase based on successive removal and reinsertion of nodes. Calvo [2] solves an assignment problem with an ad hoc objective function and builds a feasible tour merging all found sub-tours into a main tour, then applies a 3-opt local search procedure to improve the initial feasible solution. Recently, Ohlmann and Thomas [14] obtained best known results for numerous instances using a variant of simulated annealing [11], called compressed annealing, that relaxes the time windows constraints by integrating a variable penalty method within a stochastic search procedure.

In this work, we developed a two phase heuristic consisting of a constructive and an optimization stage. In the first stage, a feasible solution is constructed using a *Variable Neighborhood Search* (VNS) [13,9] heuristic. In the second phase, the solution is optimized using a General VNS [9] (GVNS) heuristic, this is, a VNS heuristic with *Variable Neighborhood Descent* [13,9] as local search. Computational results show that the proposed approach is efficient, saving on average approximately 80% of the time in comparison with the state-of-the-art algorithms, and improving best known results for some instances. We also propose a new set of TSPTW instances with more customers and wider time windows, and a new method to generate instances that improves the existent commonly used method.

The rest of the paper is organized as follows. In Section 2, we introduce the whole GVNS heuristic. In Section 3, issues about implementation and elimination tests are discussed. In Section 4, we present the experimental results. Finally, in Section 5 we report some concluding remarks.

2. Two phase heuristic

In our proposed approach to the problem, instances are pre-processed before the heuristic is executed as in [7]. This step eliminates *incompatible arcs*: if $a_i + c_{ij} > b_j$ then j must be visited before i, and therefore, every tour that contains any sequence in which i is visited before j is trivially infeasible. The pre-processing consists of marking these incompatible arcs so that the rest of the algorithm is able to quickly avoid them.

After the pre-processing, the proposed algorithm works iteratively in two stages. The construction stage seeks a feasible solution, whereas the improvement stage tries to improve the solution using the feasible solution produced in the previous phase as a starting point. A VNS heuristic is used in the construction phase and a General VNS heuristic is used in the improvement phase. Algorithm 1 depicts the whole process.

```
Algorithm 1: Two Phase Heuristic
  Input: iterMax
  Output: $X^*$
1 $iter \leftarrow 0$ ;
2 while $iter < iterMax$ do
3     $X \leftarrow BuildFeasibleSolution()$ ;          // see algorithm 2
4     $X \leftarrow GVNS(X)$ ;
5     $X^* \leftarrow better(X, X^*)$ ;
6     $iter++$ ;
7 end
```

2.1. Constructive stage

Salvesbergh [17] argues that even building a feasible solution to the TSPTW is a NP-Hard problem. Gendreau et al. [7] try to build feasible solutions using an insertion heuristic with backtracking. The results show that the algorithm fails to obtain feasible solutions for some instances with a great number of customers and narrow time windows. Calvo [2] uses the solution of an assignment problem to create feasible sub-routes and merges these into a main route. The algorithm specified by Calvo [2] works better than the one of Gendreau et al. [7] on the instances proposed in [4], but nevertheless, feasible solutions were not found for some instances.

In this paper, we propose a VNS heuristic to build feasible solutions for TSPTW. VNS has already been used in [8] for finding feasible dual solution in solving the Simple Plant-Location Problem (SPLP).

```
Algorithm 2: VNS - Constructive phase
  Output: X
1 repeat
2     $level \leftarrow 1$ ;
3     $X \leftarrow RandomSolution()$ ;
4     $X \leftarrow Local1Shift(X)$ ;
5     while $X$ is infeasible and $level < levelMax$ do
6         $X' \leftarrow Perturbation(X, level)$ ;
7         $X' \leftarrow Local1Shift(X')$ ;
8         $X \leftarrow better(X, X')$ ;
9         if $X$ is equal to $X'$ then
10            $level \leftarrow 1$ ;
11        else
12            $level++$ ;
13        end
14    end
15 until $X$ is feasible;
```

Algorithm 2 shows the constructive procedure. The objective function used in this procedure is the sum of all positive differences between the time to reach each customer and its due date, that is, $\sum_{i=1}^{n} max(0, \beta_i - b_i)$. The algorithm works iteratively until it finds a feasible solution (one with objective function equal to zero). The variable *level*, used to control the strength of the perturbation procedure, is set to one in line 2. In line 3, the algorithm generates a random, possibly infeasible, solution. A local search procedure using the 1-shift neighborhood (a neighbor solution is obtained by relocating a given customer) is applied to the initial solution in line 4. The *levelMax* parameter is used to control the maximum level of perturbation, and the algorithm iterates through lines 6 to 13 until finding a feasible solution or *levelMax* is reached. In line 6 the solution is perturbed, by making *level* random 1-shift movements, to escape from the current local optimal solution. In line 7 a 1-shift local search procedure is applied to the perturbed solution and, in line 8 the solution obtained after the local search is set as the new current solution, if it is better than the current best solution. At the end of each iteration, variable *level* is increased by one if the current solution is not improved, or reset to 1, otherwise.

Concerning the random solution used by this procedure, empirical tests showed that Algorithm 2 is able to quickly find a feasible solution when using an initial solution in where the customers are sorted by due date or by ready time. In contrast, when the algorithm is started with a random initial solution, it takes more time to find a feasible solution. The time spent in the constructive phase is small when compared with the total time spent by the whole two phase heuristic. Therefore, in order to obtain solutions with great diversity at the end of the constructive phase, we choose to work with random initial solutions. The ability to construct diverse initial solutions is essential in a multi-start algorithm such as the one proposed in this work.

2.2. Optimization phase

After the construction of a feasible initial solution, the heuristic tries to improve it. The optimization phase applies a General VNS [13,9] (GVNS) heuristic (i.e. VNS with VND as local search). The Algorithm 3 describes the optimization phase.

```
Algorithm 3: GVNS
  Input: X, levelMax
  Output: X
1 $level \leftarrow 1$ ;
2 $X \leftarrow VND(X)$ ;
3 while $level <= levelMax$ do
4     $X' \leftarrow Perturbation(X, level)$ ;
5     $X' \leftarrow VND(X')$ ;
6     $X \leftarrow better(X, X')$ ;
7     if $X$ is equal to $X'$ then
8         $level \leftarrow 1$ ;
9     else
10        $level++$ ;
11    end
12 end
```

In this phase, the objective function is the total cost of the tour, that is, $\sum_{i \in V \cup \{0\}} \sum_{j \in V \cup \{0\}} c_{ij} x_{ij}$. The algorithm starts with a feasible solution X provided by the constructive phase. The variable *level*, used to control the strength of the perturbation procedure, is set to one in line 1. In line 2, the local search VND is applied on X. The algorithm iterates from line 4 to 11 until *level* reaches *levelMax*, the maximum level of perturbation allowed. Similarly to the constructive phase, the *Perturbation* procedure makes *level* random 1-shift movements disturbing the current solution. In line 5, the local search VND is applied and, in line 6, the current and best solutions are compared and the best solution is updated if necessary. In lines 7 to 11, variable *level* is increased by one if the current solution is not improved or otherwise set to one.

VND is a variation of *Variable Neighborhood Search* (VNS) [13] that systematically changes the neighborhood within the search in a deterministic way. In this implementation, we combine the 1-shift local search with the classical 2-opt local search [10] (in the 2-opt local search a 2-exchange neighbor is used, a neighbor of the current solution is obtained removing two arcs from the solution and reconnecting the remainder two paths with different arcs). Algorithm 4 show the proposed VND procedure.

```
Algorithm 4: VND
  Input: X
  Output: X
1 $X' \leftarrow Null$ ;
2 repeat
3     $X \leftarrow Local1Shift(X)$ ;
4     $X' \leftarrow X$ ;
5     $X \leftarrow Local2Opt(X)$ ;
6 until $X = X'$;
```

The 1-shift local search is well suited to address the feasibility issue posed by the time windows constraints. This fact can be verified by the results obtained in the construction phase where the only aim is to build a feasible solution. The 2-opt local search, even restricted by the time windows constraints, is able to address the optimization issue of the problem. We observe that it is difficult to find feasible 2-exchange movements but, on the other hand, when this movements are available, they are often worthy.

3. Implementation issues

The efficient implementation of local search procedures is the key for success of several heuristic algorithms. Two implementation issues are determinant in the efficiency of local search procedures: the number of neighbor solutions evaluated and the fast computation of the neighbor solutions costs, see for example [16].

In our local search implementation, no movements that lead to solutions using incompatible arcs are evaluated, as proposed in [7]. This means that if a customer i must be visited before a customer j in every feasible solution, no movement that locates customer j before customer i is ever evaluated. This fact helps to reduce the size of the neighborhoods being investigated.

In this section we denote the current solution as Π and the customer being visited in the kth position as π_k. All movements consider the customer in a given position i and try to relocate it in other position j, preserving feasibility and improving the cost of the objective function. If $i < j$ and the arc (π_j, π_i) is incompatible, any movement that places π_i in a position $h > j$ does not need to be evaluated since it will lead to an infeasible solution. Analogously, if $i > j$ and the arc (π_i, π_j) is incompatible, any movement that places π_i in a position $h < j$ does not need to be evaluated.

In the following subsections we describe the implementation issues that accelerate our local search procedures.

3.1. 1-shift local search in the constructive phase

In the constructive phase of the algorithm, we aim to obtain a feasible solution. The procedure starts with an initial (possibly infeasible) solution and applies 1-shift local search procedures until obtaining a feasible solution.

Our local search procedure implementation is of first improvement type. This means that the incumbent solution is updated, as soon as an improvement in the neighborhood of the current one is found. In order to evaluate as few neighbor solutions as possible, the algorithm first investigates the portion of the neighborhood more likely to contain improving solutions.

We divide 1-shift movements into two subsets (see Fig. 1). Backward movements shift a given customer to an earlier position in the sequence. After the movement, the customer will be visited earlier than in the current solution. On the other hand, forward movements shift a given customer to a later position in the sequence. After the movement, the customer will be visited later than in the current solution.

We also divide the customers into two subsets. Violated customers are those that in the current solution are visited after their due dates. The rest of the customers are non-violated.

These subset definitions partition the current neighborhood into four parts:

- backward movements of violated customers,
- forward movements of non-violated customers,

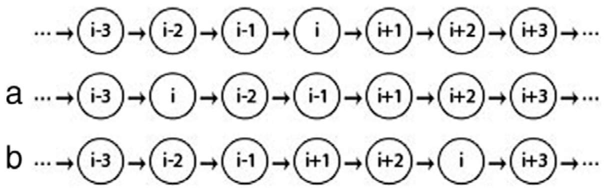

Fig. 1. Local search 1-shift backward (A) and forward (B) movements.

- backward movements of non-violated customers,
- forward movements of violated customers.

We analyze movements in each of these parts to improve the efficiency of the algorithm. The first part of the neighborhood corresponds to backward movements of violated customers. In this case, the customer being moved may turn into a non-violated customer since it will be visited earlier. Customers between the current and the new position of the customer being moved may turn into violated customers since they may be visited later than in the current solution.

The second part of the neighborhood corresponds to forward movements of non-violated customers. In this case, the customer being moved may turn into a violated customer since it will be visited later. Customers between the current and the new position of the customer being moved may turn into non-violated customers since they may be visited earlier than in the current solution.

The third part of the neighborhood corresponds to backward movements of non-violated customers. In this case, the customer being moved remains non-violated. Customers between the current and the new position of the customer being moved may turn into violated customers since they may be visited later than in the current solution.

The fourth part of the neighborhood corresponds to forward movements of violated customers. In this case, the customer being moved remains violated. Customers between the current and the new position of the customer being moved may turn into non-violated customers since they may be visited earlier than in the current solution.

In any case, customers that are visited after customers in the positions involved in the movement may be visited earlier or later depending on the movement, therefore these customers may turn into violated or non-violated. Customers visited before the customers in the positions involved in the movement are visited exactly at the same time they are visited in the current solution and their violated/non-violated state does not need to be reevaluated.

The first part of the neighborhood is the most promising in terms of the chance to find an improving solution followed by the second, third and the fourth part. Therefore the neighborhood of the current solutions is evaluated in this order. We note that since the objective function considers the positive difference between arrival time and due date a solution could be improved even if the number of violated customers is not decreased.

3.2. 1-shift local search in the optimization phase

The optimization phase starts with a feasible solution provided by the construction phase and aims to improve the solution until the stopping criterion is met.

In this stage, a neighbor solution has to be evaluated regarding both its distance traveled and its feasibility. The evaluation of the distance traveled can be done in $O(1)$ since it implies the removal of three arcs and the insertion of another three arcs (see Fig. 1). The evaluation of the solution feasibility is more complex. Therefore, we evaluate the distance traveled before evaluating solution feasibility and, if the cost of the neighbor solution is not better than the cost of the current solution, then we discard the neighbor solution without evaluating its feasibility.

Since the current solution is feasible, the customers in the sequence beginning with the first customer and ending at the last customer before the first customer affected by the movement remain feasible. Moreover, arrival times to these customers are unaffected by the movement.

The customers that are visited in between the former and new position of the customer being shifted may turn infeasible and their arrival time may change. Therefore the arrival time for these customers has to be reevaluated.

The arrival time of the customers visited after the last position affected by the movement may also change and so, they may turn infeasible. Therefore, their arrival time must be reevaluated. However, as soon as a given customer is visited no later than in the current solution, the neighbor solution is determined to be feasible. Since the cost of the neighbor solution was determined to be better than the cost of the current solution before testing feasibility, the new feasible solution is an improving one and is accepted as the new current solution.

3.3. 2-opt local search in the optimization phase

As it happens in the evaluation of the distance traveled for 1-shift movements, the distance traveled in neighbor solutions considering the 2-opt neighborhood can also be reevaluated in $O(1)$. It simply consists in the removal of two arcs and the insertion of another two arcs. The evaluation of feasibility is more complex. Therefore, as in 1-shift movements, the distance traveled is evaluated before evaluating solution feasibility.

The 2-exchange movement, performed in the 2-opt local search, inverts part of the solution, trying to minimize the objective function. In every 2-exchange movement, a given customer is shifted forward a given number of positions and the customers in between are inverted. We note that if one of the customers being moved becomes a violated customer then it will also be violated in any movement involving the same customer being shifted a greater number of positions. Therefore, as soon as any customer involved in the movement becomes violated, no movement involving the same customer, being shifted a greater number positions is evaluated since it will certainty generate an infeasible solution.

As in 1-shift local search, the feasibility of the solution is not completely evaluated for each movement. The algorithm considers the same rules of 1-shift in optimization phase to cut the feasibility evaluation as soon as possible.

4. Results

The proposed algorithm was coded in C++ and run on a Pentium 4 2.40 GHz processor with 1 GB of RAM. We choose this test environment since it is similar to the one used in [14] (Pentium 4 2.66 GHz processor with 1 GB of RAM), the state-of-the-art heuristic. On all experiments, parameters *levelMax* and *iterMax* were respectively set to 8 and 30. These parameters were chosen through empirical tests and, with them, the algorithm seems to produce good solutions in a reasonable amount of time in comparison with other parameter values tested. The parameter — O3 was used in compilation to optimize the code.

4.1. Empirical tests planning

We tested two aspects of the algorithm. The first aspect tested was the constructive stage, where we compare the implementation of this stage with the same implementation without the neighborhood partitioning improvement made in code. To compare these two implementations we use a time-to-target plot (TTTPlot) [1].

We also tested the performance of the whole implementation against state-of-the-art algorithms. We compare both the numerical results and computational time.

4.2. Instances

We tested the algorithm in four sets of instances. The first set was proposed by Dumas et al. [4] and contains 135 instances grouped in 27 test cases. Each group has five Euclidean instances, coordinates between 0 and 50, with the same number of customers and the same maximum range of time windows. To illustrate this, the instances n40w60.001, n40w60.002, n40w60.003, n40w60.004 and n40w60.005 have 40 customers and the time window for each customer is uniformly random, between 0 and 60. The final result of each group is the average of the result on the five instances, however, the time computed considers the time to process all five instances.

The second group of instances was proposed by Gendreau et al. [7] and contains 140 instances grouped in 28 test cases. The third group of instances was proposed by Ohlmann and Thomas [14] and contains 25 instances grouped in 5 test cases. The second and third groups instances, in majority, are the instances proposed by Dumas et al. [4] with wider time windows, systematically extended by 100.

The fourth group is proposed in this paper to overcome the limitations of the other test sets. Considering the currently approaches to TSPTW, we need instances with more than 200 customers and wider time windows to benchmark and compare algorithms. Another reason to propose this set, is the limitation of the heuristic used to generate the instances on previous works. Dumas et al. [4] build the instances with customers at random position, x and y between 0 and 50, and construct the second-nearest neighbor tour. Then, the algorithm traverses the tour randomly choosing the size of the time window for each customer, so the ready time is the arrival time minus random value between 0 and $w/2$, and the due date is the arrival time plus a random value between 0 and $w/2$, where w is the maximum time window length.

We believe that this method is a biased way to generate instances since it does not reflect the real cases. Moreover, there is no challenge to build a feasible solution because the second-nearest neighbor tour already is a feasible solution.

In the method being proposed here, we increase the x and y maximum range to 100 and build a random path to visit all customers, as opposed to the second-nearest neighbor tour. In the same manner as Dumas et al., we randomly generate the time window width for each customer in the tour. So, we have 125 instances grouped in 25 test cases, with 200 to 400 customers per instance, and time windows between 100 and 500 units.

4.3. Numerical results

Fig. 2 shows a time-to-target plot [1] comparing the performance of the constructive stage with and without partitioning the neighborhood in the local search, as explained in Section 3.1, using the instance n80w60 proposed by Dumas et al. [4].

The plot shows that the partitioning works well to reduce the amount of time needed to construct a feasible solution. With 0.05 s the probability of the algorithm without partitioning construct a feasible solution is close to zero. On the other hand, the probability of the algorithm with partitioning constructs a feasible solution is approximately 45%, with the same amount of time. The same behavior is noted in all benchmark instances.

Table 1
Comparison between results obtained by Ohlmann and Thomas [14], and by General heuristic, on instances proposed by Dumas et al. [4].

Instance			Compressed annealing			General VNS heuristic				
n	w	Optimal	Best value	Avg. value	Avg. sec.	Best value	Avg. value	Avg. σ value	Avg. sec.	Avg. σ sec.
	20	361.2	361.2	361.2	2.0	361.2	361.2	0.0	0.2	0.0
	40	316.0	316.0	316.0	2.7	316.0	316.0	0.0	0.2	0.0
20	60	309.8	309.8	309.8	2.5	309.8	309.8	0.0	0.2	0.0
	80	311.0	311.0	311.0	3.0	311.0	311.0	0.0	0.3	0.0
	100	275.2	275.2	275.2	3.2	275.2	275.2	0.0	0.3	0.0
	20	486.6	486.6	486.6	3.8	486.6	486.6	0.0	0.3	0.0
	40	461.0	461.0	461.0	5.1	461.0	461.0	0.0	0.4	0.0
40	60	416.4	416.4	416.5	6.0	416.4	416.4	0.0	0.5	0.0
	80	399.8	399.8	399.8	6.2	399.8	399.9	0.4	0.5	0.0
	100	377.0	377.0	377.5	6.6	377.0	377.0	0.2	0.6	0.0
	20	581.6	581.6	581.6	7.2	581.6	581.6	0.0	0.6	0.0
	40	590.2	590.2	590.7	8.2	590.2	590.2	0.0	0.8	0.0
60	60	560.0	560.0	560.0	8.5	560.0	560.0	0.0	0.9	0.0
	80	508.0	508.0	509.3	8.6	508.0	508.1	0.2	1.2	0.0
	100	514.8	514.8	516.5	8.8	514.8	514.8	0.0	1.3	0.0
	20	676.6	676.6	676.6	11.3	676.6	676.6	0.0	0.9	0.0
80	40	630.0	630.0	630.2	11.5	630.0	630.0	0.0	1.3	0.0
	60	606.4	606.4	607.0	12.0	606.4	606.4	0.1	1.8	0.1
	80	593.8	593.8	594.1	11.5	593.8	593.8	0.1	2.1	0.1
	20	757.6	757.6	757.8	15.4	757.6	757.6	0.0	1.4	0.0
100	40	701.8	701.8	702.4	15.7	701.8	701.8	0.0	1.9	0.1
	60	696.6	696.6	697.2	15.9	696.6	696.6	0.0	2.7	0.1
	20	868.4	868.4	869.2	24.7	868.4	868.4	0.0	3.6	0.3
150	40	834.8	834.8	836.2	25.2	834.8	834.8	0.0	5.3	0.3
	60	805.0	818.8	820.2	25.6	**818.6**	818.6	0.1	7.4	0.7
200	20	1009.0	1009.0	1010.0	35.1	1009.0	1009.1	0.1	8.5	0.5
	40	984.2	984.6	986.1	35.2	**984.2**	984.2	0.1	12.6	0.8

Table 2
Comparison between results obtained by Ohlmann and Thomas [14], and by General VNS heuristic, on instances proposed by Gendreau et al. [7].

Instance			Compressed annealing			General VNS heuristic				
n	w	Best known	Best value	Avg. value	Avg. sec.	Best value	Avg. value	Avg. σ value	Avg. sec.	Avg. σ sec.
	120	265.6	265.6	265.6	3.1	265.6	265.6	0.0	0.3	0.0
	140	232.8	232.8	232.8	3.9	232.8	232.8	0.0	0.3	0.0
20	160	218.2	218.2	218.2	4.0	218.2	218.2	0.0	0.3	0.0
	180	236.6	236.6	236.0	4.0	236.6	236.6	0.0	0.4	0.0
	200	241.0	241.0	241.0	4.1	241.0	241.0	0.0	0.4	0.0
	120	360.0[a]	377.8	378.1	6.0	377.8	377.8	0.0	0.8	0.0
	140	348.4[a]	364.4	364.7	6.0	364.4	364.4	0.0	0.8	0.0
40	160	326.8	326.8	327.1	6.0	326.8	326.8	0.0	0.9	0.0
	180	326.8[a]	332.0	333.9	6.2	330.4	331.3	0.8	1.0	0.0
	200	313.8	313.8	315.0	6.3	313.8	314.3	0.4	1.0	0.1
	120	451.0	451.0	452.9	8.3	451.0	451.0	0.1	1.5	0.1
	140	452.4	452.4	454.0	8.6	**452.0**	452.1	0.2	1.7	0.1
60	160	448.6[a]	464.6	465.4	8.4	464.0	464.5	0.2	1.7	0.0
	180	421.6	421.6	425.2	8.6	**421.2**	421.2	0.1	2.2	0.1
	200	427.4	427.4	430.8	8.4	427.4	427.4	0.0	2.4	0.1
	100	579.2	579.2	581.6	11.5	**578.6**	578.7	0.2	2.3	0.1
	120	541.4	541.4	544.0	11.5	541.4	541.4	0.0	2.7	0.1
	140	509.8	509.8	513.6	11.3	**506.0**	506.3	0.2	3.2	0.3
80	160	502.8[a]	505.4	511.7	11.2	504.8	505.5	0.7	3.3	0.1
	180	489.0[a]	502.0	505.9	11.4	500.6	501.2	0.9	3.7	0.1
	200	481.8	481.8	486.4	11.1	**481.4**	481.8	0.1	4.2	0.2
	80	666.4	666.4	668.1	15.9	666.4	666.6	0.2	3.1	0.2
	100	642.2	642.2	645.0	14.6	**642.0**	642.1	0.1	3.7	0.1
	120	601.2	601.2	603.7	15.0	**597.2**	597.5	0.3	4.1	0.2
100	140	579.2	579.2	582.5	14.9	**548.4**	548.4	0.0	4.4	0.2
	160	570.4[a]	584.0	588.8	15.0	**555.0**	555.0	0.1	5.1	0.2
	180	561.6	561.6	566.9	14.9	561.6	561.6	0.0	6.3	0.3
	200	555.4	555.4	562.3	14.9	**550.2**	551.0	1.2	6.8	0.3

[a] Best known results obtained by [2].

Table 3
Comparison between results obtained by Ohlmann and Thomas [14], and by General VNS heuristic, on instances proposed by Ohlmann and Thomas [14].

Instance			Compressed annealing			General VNS heuristic				
n	*w*	Best known	Best value	Avg. value	Avg. sec.	Best value	Avg. value	Avg. σ value	Avg. sec.	Avg. σ sec.
150	120	725.0	725.0	731.1	24.8	**722.0**	722.3	0.4	11.8	0.3
	140	697.6	697.6	705.4	24.9	**693.8**	694.8	0.5	13.3	0.5
	160	673.6	673.6	680.9	25.0	**671.0**	671.2	0.3	15.0	0.8
200	120	806.8	806.8	817.0	34.4	**803.6**	803.9	0.1	30.3	2.0
	140	804.6	804.6	812.6	35.2	**798.0**	799.5	1.1	38.0	1.8

Table 4
Results obtained by General VNS heuristic on instances proposed in this paper.

Instance		General VNS heuristic				
n	*w*	Best value	Avg. value	Avg. σ value	Avg. sec.	Avg. σ sec.
200	100	10 019.6	10 019.6	0.1	4.8	0.3
	200	9 252.0	9 254.1	7.2	5.8	0.2
	300	8 026.4	8 034.3	4.5	7.2	0.2
	400	7 067.2	7 079.3	4.4	8.7	0.4
	500	6 466.4	6 474.0	5.1	10.0	0.3
250	100	12 633.0	12 633.0	0.0	9.9	0.2
	200	11 310.4	11 310.7	0.7	11.9	0.4
	300	10 230.4	10 235.1	2.8	14.9	0.6
	400	8 899.2	8 908.5	4.1	18.9	0.7
	500	8 082.4	6 474.0	6.7	20.7	0.9
300	100	15 041.2	15 041.2	0.0	21.2	0.7
	200	13 846.8	13 853.1	2.3	23.7	0.6
	300	11 477.6	11 488.5	5.2	37.0	3.8
	400	10 413.0	10 437.4	12.9	31.7	1.2
	500	9 861.8	9 876.7	8.9	35.4	1.1
350	100	17 494.0	17 494.0	0.0	41.0	2.5
	200	15 672.0	15 672.2	0.6	47.3	2.1
	300	13 650.2	13 654.1	1.7	54.9	2.2
	400	12 099.0	12 119.6	8.9	60.2	2.8
	500	11 365.8	11 388.2	12.0	57.8	1.2
400	100	19 454.8	19 454.8	0.0	57.1	0.6
	200	18 439.8	18 439.9	0.6	66.9	1.9
	300	15 873.4	15 879.1	3.0	93.6	7.9
	400	14 115.4	14 145.5	12.9	96.2	3.9
	500	12 747.6	12 766.2	9.7	109.3	4.4

Table 1 shows the results on the Dumas et al. [4] instances and compares them with the optimal solution and the results obtained by compressed annealing. The GVNS heuristic did not obtain the optimal just for instance n150w60. The average result is slightly better (0.07%) than the average result obtained by compressed annealing, but the run time was reduced by 86.80% on average. This is a substantial gain considering the similarity of the computational environment with the one used in this work.

The results on the instances proposed by Gendreau et al. are presented in Table 2. The GVNS results are compared to compressed annealing and best known results obtained in the literature, since no optimal solutions are known for these instances. The GVNS heuristic is 0.92% better than compressed annealing on the average result, and also obtained best known results for 10 out of 27 instances. Moreover, the computational time was reduced an average of 77.68% in comparison with compressed annealing.

Table 3 shows the results on the instances proposed by Ohlmann and Thomas [14]. Best known results were found for all benchmark instances and the average time was reduced by 29.59% in mean.

Finally, Table 4 shows the results for the instances proposed in this paper. Although we cannot compare the results to other approaches, we succeed in producing feasible solutions for instances with 300 customers and time windows with a maximum width of 500 units in less than 40 s executing the algorithm with the same parameters used on previous data sets.

5. Conclusion

In this paper we proposed a General VNS (GVNS) heuristic for the Traveling Salesman Problem with Time Windows (TSPTW). This work shows that a good implementation plays an important role in heuristic optimization, since elimination tests and neighborhood reduction enables the algorithm to quickly obtain good results.

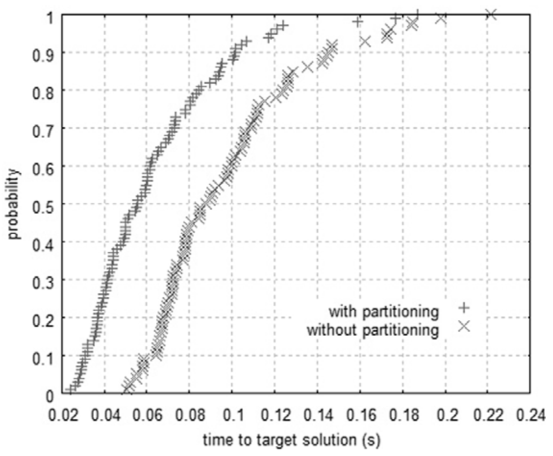

Fig. 2. TTTPlot comparing the local search in the constructive stage with and without partitioning the neighborhood on instance n80w60 proposed by [4].

With respect to the instance data set proposed by Dumas et al. [4], the algorithm found good quality solutions while reducing the computational time by 86.80% on average in comparison with the state-of-the-art. In the instances proposed by Gendreau et al. [7], the algorithm obtained best known results for 10 out of 27 instances, reducing the computational time by 77.68% on average in comparison with the state-of-the-art. The algorithm also produced best known results for all instances proposed by Ohlmann and Thomas [14]. We proposed a new set of instances with up to 400 customers and the algorithm is able to obtain feasible solutions for these instances in a reasonable amount of time.

References

[1] R.M. Aiex, M.G.C. Resende, C.C. Ribeiro, TTTPlots: a perl program to create time-to-target plots, Optimization Letters 1 (4) (2006) 355–366.
[2] R.W. Calvo, A new heuristic for the traveling salesman problem with time windows, Transportation Science 34 (1) (2000) 113–124.
[3] W.B. Carlton, J.W. Barnes, Solving the travelling salesman problem with time windows using tabu search, IEE Transactions 28 (1996) 617–629.
[4] Y. Dumas, J. Desrosiers, E. Gelinas, M. Solomon, An optimal algorithm for the travelling salesman problem with time windows, Operations Research 43 (2) (1995) 367–371.
[5] F. Focacci, A. Lodi, M. Milano, A hybrid exact algorithm for the TSPTW, INFORMS Journal on Computing 14 (4) (2002) 403–417.
[6] M. Gendreau, A. Hertz, G. Laporte, New insertion and postoptimization procedures for the travelling salesman problem, Operations Research 40 (1992) 1086–1094.
[7] M. Gendreau, A. Hertz, G. Laporte, M. Stan, A generalized insertion heuristic for the traveling salesman problem with time windows, Operations Research 46 (3) (1998) 330–335.
[8] P. Hansen, J. Brimberg, D. Urošević, N. Mladenović, Primal-dual variable neighborhood search for the simple plant-location problem, INFORMS Journal on Computing 19 (4) (2007) 552–564.
[9] P. Hansen, N. Mladenović, J.A.M. Pérez, Variable neighbourhood search: methods and applications, 4OR 6 (4) (2008) 319–360.
[10] D.S. Johnson, L.A. McGeoch, The traveling salesman problem: a case study in local optimization, in: E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, John Wiley and Sons, 1995, pp. 215–310.
[11] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[12] A. Langevin, M. Desrochers, J. Desrosiers, S. Gélinas, F. Soumis, A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows, Networks 23 (7) (1993) 631–640.
[13] N. Mladenović, P. Hansen, Variable neighborhood search, Computations & Operations Research 24 (11) (1997) 1097–1100.
[14] J.W. Ohlmann, B.W. Thomas, A compressed-annealing heuristic for the traveling salesman problem with time windows, INFORMS Journal on Computing 19 (1) (2007) 80–90.
[15] G. Pesant, M. Gendreau, J.-Y. Potvin, J.-M. Rousseau, An exact constraint logic programming algorithm for the travelling salesman problem with time windows, Transportation Science 32 (1998) 12–29.
[16] C.C. Ribeiro, D. Aloise, T.F. Noronha, C. Rocha, S. Urrutia, An efficient implementation of a VNS/ILS heuristic for a real-life car sequencing problem, European Journal of Operational Research 191 (3) (2008) 596–611.
[17] M.W. Salvesbergh, Local search in routing problems with time windows, Annals of Operations Research 4 (1985) 285–305.