Information Sciences 279 (2014) 383–395

Contents lists available at ScienceDirect

Information Sciences

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

# A variable iterated greedy algorithm for the traveling salesman problem with time windows

Korhan Karabulut [a], M. Fatih Tasgetiren [b,*]

[a] Software Engineering Department, Yasar University, Selcuk Yasar Campus, Izmir, Turkey
[b] Industrial Engineering Department, Yasar University, Selcuk Yasar Campus, Izmir, Turkey

ARTICLE INFO

*Article history:*
Received 10 April 2012
Received in revised form 24 February 2014
Accepted 29 March 2014
Available online 5 April 2014

*Keywords:*
Traveling salesman problem with time windows
Iterated greedy algorithm
Variable neighborhood search
Heuristic optimization

ABSTRACT

This paper presents a variable iterated greedy algorithm for solving the traveling salesman problem with time windows (TSPTW) to identify a tour minimizing the total travel cost or the makespan, separately. The TSPTW has several practical applications in both production scheduling and logistic operations. The proposed algorithm basically relies on a greedy algorithm generating an increasing number of neighboring solutions through the use of the idea of neighborhood change in variable neighborhood search (VNS) algorithms. In other words, neighboring solutions are generated by destructing a solution component and re-constructing the solution again with variable destruction sizes. In addition, the proposed algorithm is hybridized with a VNS algorithm employing backward and forward 1_*Opt* local searches to further enhance the solution quality. The performance of the proposed algorithm was tested on several benchmark suites from the literature. Experimental results confirm that the proposed algorithm is either competitive to or even better than the best performing algorithms from the literature. Ultimately, new best-known solutions are obtained for 38 out of 125 problem instances of the recently proposed benchmark suite, whereas 15 out of 31 problem instances are also further improved for the makespan criterion.



## 1. Introduction

The traveling salesman problem with time windows (TSPTW) addresses finding a minimum cost-tour that starts and ends at a given depot and features exactly one visit to a given set of customers. Each customer has a service time and a time window defining its ready time and due date. Each customer must be visited before its due date minus service time for a feasible tour. Otherwise, the tour is considered infeasible. On the other hand, if the vehicle arrives before the customer ready time, it must wait. The TSPTW can be modeled as a routing or a scheduling problem. For routing tasks, the objective is to find a route to visit a number of customers, starting and ending at a depot with the constraint that each customer must be visited in a time window. Typically, in this case, the objective function is the cost of a tour, which is, in fact, the total distance traveled. In addition to above, the TSPTW is equivalent to modeling the problem of scheduling jobs on a single machine where setup times are sequence dependent, and each job has a release and due date. In this case, the objective function is to minimize the tour-completion time, the so-called makespan.

* Corresponding author.
*E-mail addresses:* korhan.karabulut@yasar.edu.tr (K. Karabulut), fatih.tasgetiren@yasar.edu.tr (M. Fatih Tasgetiren).

http://dx.doi.org/10.1016/j.ins.2014.03.127
0020-0255/ 

The TSPTW has been demonstrated to be NP-Hard. In addition, even finding a feasible solution is an *NP*-complete problem [1]. Solution methods for the TSPTW can be categorized into exact and heuristic approaches. The first exact algorithms for the TSPTW, developed by Christofides et al. [2] and Baker [3], were branch-and-bound algorithms, and the main focus was on the makespan minimization for solving instances up to 50 nodes. However, they were restricted to tight time windows or mostly overlapping windows. Both makespan and travel-cost optimization were considered in Langevin et al. [4], where a two-commodity How formulation was presented within a branch-and-bound procedure that was able to solve instances up to 40 nodes. Dumas et al. [5] extended earlier dynamic programming approaches through the use of a state-space-reduction technique that was able to solve instances up to 200 customers. In addition to the above, Ascheuer et al. [6] considered a branch-and-cut algorithm for solving the asymmetric TSPTW, and Balas and Simonetti [7] developed a dynamic programming algorithm for various TSP variants with precedence constraints, including the TSPTW. Constraint programming has also been employed to develop exact and heuristic algorithms in Pesant et al. [8] and Focacci et al. [9].

As the problem is NP-Hard, heuristic approaches have attracted attention for solving the TSPTW. Carlton and Barnes [10] presented a tabu search approach that considers infeasible solutions in its search neighborhood by using a static penalty function. Gendreau et al. [11] offered a construction and post-optimization heuristic based on a near-optimal TSP heuristic proposed by Gendreau et al. [12]. Calvo [13] introduced a construction heuristic where an initial solution was constructed by a novel assignment relaxation, which then improves upon this tour through a local search. Ohlmann and Thomas [14] developed an excellent compressed annealing (CA) algorithm employing a variable penalty function. An ant colony algorithm was also proposed by Cheng and Mao [15] where the makespan criterion was considered.

Very recently, two excellent papers were presented to solve the TSPTW. A beam-ant colony (Beam-ACO) approach was presented by Ibanez and Blum [16] where most of the best-known solutions reported in [14] were further improved, and the Beam-ACO algorithm was regarded as the state-of-art. However, a general variable neighborhood search (GVNS) heuristic was presented by da Silva and Urrutia [17] where further improvements were also reported, and the results for the instances compared with those of Dumas et al. [5], Gendreau et al. [11] and Ohlmann and Thomas [14] were better than those in Ibanez and Blum [16]. In addition, da Silva and Urrutia [17] have also presented new instances varying from 200 nodes to 400 nodes with time windows varying from 100 to 500.

In fact, the GVNS algorithm is a variable iterated local search where the perturbation level is determined by the idea of neighborhood change in variable neighborhood search algorithms (VNS). In other words, the perturbation level is fixed at $level = 1$ at the beginning. Then, a local search is applied to the perturbed solution. If the solution after the local search is further improved, the perturbation level remains at $level = 1$. Otherwise, the perturbation level is increased by one, i.e., $level = level + 1$. The maximum perturbation and maximum iteration levels are fixed at $LevelMax = 8$ and $IterMax = 30$, respectively. The GVNS algorithm has two phases. In the first phase, a feasible solution is first established by a VNS algorithm, which is called the constructive phase. Next, the GVNS algorithm is iteratively applied in a VNS loop again by using the variable neighborhood descend (VND) algorithm with two neighborhood structures of *Local*1*Shift* and *Local*2*Opt*. In this paper, we took inspiration from the GVNS algorithm by replacing the perturbation phase with the destruction and construction procedure of iterated greedy algorithms in [18,19]. As the state-of-art results are presented in da Silva and Urrutia [17], we mainly compare the performance of the proposed algorithm to the GVNS algorithm presented in da Silva and Urrutia [17]. 將擾動階段替換為迭代貪婪算法的破壞和構造過程

The IG algorithm is presented in Ruiz and Stützle [18], which has successful applications in discrete/combinatorial optimization problems such as those in [19–32]. The IG algorithm is fascinating in terms of its conceptual simplicity, which makes it easily tunable and extendible to any combinatorial optimization problem. In an IG algorithm, there are two central procedures consisting of the destruction and the construction phases. The algorithm starts from some initial solution and then iterates through a main loop where a partial candidate solution is first obtained by removing a number of solution components from a complete candidate solution. This is called the[1] destruction phase. Next, a complete solution is reconstructed with a constructive insertion heuristic by inserting each job in the partial candidate solution. Before continuing with the next loop, an acceptance criterion is then used to decide whether the re-constructed solution will replace the incumbent one. This is called the[2] construction phase. These simple steps are iterated until some predetermined termination criterion, such as a maximum number of iterations or a computation time limit, is met. For the details of the IG algorithm, we refer to Ruiz and Stützle [18].

In addition to the above, a variable IG algorithm (VIG_FL) is also presented and implemented to solve the permutation flowshop scheduling with the tardiness criterion in Framinan and Leisten [19]. The VIG_FL algorithm is inspired by the VNS algorithm in [33]. The idea of neighborhood change of the VNS algorithm is used to determine variable destruction sizes. The maximum destruction size is fixed at $d_{max} = n - 1$. The destruction size is initially set to $d = 1$. The current solution is destructed with a variable size of $d$ and re-constructed again. Then, a local search is applied to the reconstructed solution. The destruction size is incremented by one, that is, $d = d + 1$, if the solution is not improved until $d_{max} = n - 1$. Whenever a solution improves in any destruction size, the destruction size is again set to $d = 1$. Hence, the search starts from scratch again. For the details of the VIG_FL algorithm, we refer to Framinan and Leisten [19]. 沒有進步d++，有進步d=1

For scheduling problems in general, the IG and VIG_FL algorithms begin with a problem specific heuristic, which is usually the NEH heuristic [34]. Both in the destruction and construction phases, the well-known insertion scheme of the NEH heuristic is used. Insertion based local search algorithms are employed to further enhance the solution quality. However, the above IG and VIG_FL algorithms are specifically designed for scheduling problems. In the following section, we propose a simple variable iterated greedy algorithm (VIG_VNS) algorithm, which is tailored for solving the TSPTW problem. We

demonstrate in this paper that the VIG_VNS algorithm is able to yield the state-of-art results for the TSPTW. To the best of our knowledge, this paper is the first to employ the iterated greedy algorithm (IG) to solve the TSPTW. Ultimately, 38 out of 125 instances are further improved for the new benchmark instances proposed in da Silva and Urrutia [17], whereas 15 out 31 problem instances are also further improved for the makespan criterion.

The remaining part of the paper is organized as follows. Section 2 provides the problem formulation. Section 3 is devoted to the details of the proposed VIG_VNS algorithm. The computational results for the benchmark suite are discussed in Section 4. Section 5 presents the conclusions.

## 2. Problem formulation

We have an undirected complete graph $G = \{N, A\}$ where $N = \{0, 1, \ldots, n\}$ is a set of nodes representing the depot (node 0) and $n$ customers, and $A = N \times N$ is a set of edges connecting the nodes where a solution to the TSPTW is a tour visiting each node once, starting and ending at the depot. Therefore, a tour is represented as $\pi = \{\pi_0, \pi_1, \ldots, \pi_n, \pi_{n+1}\}$ where $\pi_0 = \pi_{n+1} = 0$, and the sub-sequence $\pi = \{\pi_1, \ldots, \pi_k, \ldots, \pi_n\}$ is a permutation of the nodes in $N\backslash\{0\}$, and $\pi_k$ denotes the index of the customer at the $k^{th}$ position of the tour. Two additional elements, $\pi_0$ and $\pi_{n+1}$, represent the depot where each tour must start and end.

For every edge $(i,j) \in A$ between two nodes $i$ and $j$, there is an associated cost $c(i,j)$ that represents the travel time between customers $i$ and $j$ plus a service time at customer $i$. In addition, there is a time window $[e_i, l_i]$ related to each node $i \in N$ that specifies that customer $i$ cannot be serviced before $e_i$ or visited later than $l_i$. Furthermore, waiting times are permitted, that is, a node $i$ can be reached before the start of its time window $e_i$ but cannot be left before $e_i$. Therefore, given a particular tour $\pi$, the departure time from customer $\pi_k$ is calculated as $D_{\pi_k} = \max(A_{\pi_k}, e_{\pi_k})$ where $A_{\pi_k} = D_{\pi_{k-1}} + c(\pi_{k-1}, \pi_k)$ is the arrival time at customer $\pi_k$.

In general, two primary TSPTW objective functions are considered in the literature. The first function is used to minimize the sum of arc-traversal costs along the tour, and the second function is used to minimize the tour-completion time, i.e., the time to return to the depot, the so-called makespan. In this paper, we focus on the minimization of both objectives. The TSPTW can be formulated under the objective of the total travel cost as follows:

$$\min \quad f(\pi) = \sum_{k=0}^{n} c(\pi_k, \pi_{k+1}) \tag{1}$$

$$\text{st}: \quad v(\pi) = \sum_{k=0}^{n+1} \omega(\pi_k) = 0 \tag{2}$$

where

$$\omega(\pi_k) = \begin{cases} 1 & \text{if} \quad A_{\pi_k} > l_{\pi_k} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

In the above definition, $v(\pi)$ denotes the total number of time window constraints that are violated by tour $\pi$, which must be zero for feasible solutions. Note that in the case of minimization of the makespan, we keep track of the waiting time of the vehicle at each position of the tour as $W_{\pi_k} = D_{\pi_k} - A_{\pi_k}$. The total waiting time, $\sum_{k=0}^{n+1} W_{\pi_k}$ will be added to the objective function in (1).

It should be noted that the TSPTW is a constrained optimization problem; thus, search operators may generate infeasible solutions. In this case, care must be taken in regards to them violating the constraints. Different approaches exist that can handle the constraints [35]. In this paper, two very sophisticated approaches are employed to handle the constraints. These approaches are summarized below.

Deb [36] proposed the superiority of feasible solutions (SF) for constrained optimization based on lexicographic ordering, where constraint violation and objective function value are distinguished. The aim is to optimize both the constraint violation and objective function by a lexicographic order where constraint violation precedes the objective function value. In SF, when two solutions $\pi_a$ and $\pi_b$ are evaluated, $\pi_a$ is deemed to be superior to $\pi_b$ under the following conditions for a minimization problem:

- $\pi_a$ is feasible, and $\pi_b$ is not.
- $\pi_a$ and $\pi_b$ are both feasible, and $\pi_a$ has a smaller objective function value than $\pi_b$.
- $\pi_a$ and $\pi_b$ are both infeasible, but $\pi_a$ has a smaller overall constraint violation $v(\pi_a)$ as computed by using the Eq. (2).

The adaptive penalty approach is presented in [37] where the notion of a "*near feasibility threshold*" (NFT) corresponds to a "*promising region*" beyond the feasible region. The NFT is defined as a threshold distance from a feasible region such that the search within feasible region and the NFT-neighborhood of the feasible region is encouraged, whereas it is discouraged beyond that threshold. In addition, an adaptive term is added to the penalty function to consider the gap between the best feasible value and best infeasible value found so far. Then, the adaptive penalty function is defined for the $m$ number of constraints as follows:

NFT 被定義為距可行區域的閾值距離，從而鼓勵在可行區域內和可行區域的 NFT 鄰域內進行搜索，但不鼓勵超過該閾值。

$$f_p(\pi) = f(\pi) + (f_{feas} - f_{all}) \sum_{i=1}^{m} \left( \frac{v_i(\pi)}{NFT_i} \right)^{\alpha} \tag{4}$$

where $f_{all}$ denotes the un-penalized value of the best solution found yet, and $f_{feas}$ denotes the value of the best feasible solution yet found. As noted in [38], the adaptive term may lead to zero or over-penalty. For this reason, only the dynamic part of the above penalty function with NFT threshold is used in this paper as follows:

$$f_p(\pi) = \sum_{k=0}^{n} c(\pi_k, \pi_{k+1}) + \left( \frac{v(\pi)}{NFT} \right)^{\alpha} \tag{5}$$

本文僅使用上述帶NFT閾值的懲罰函數的動態部分

Note that we have only a single constraint in the TSPTW problem. The general form of the NFT is given by $NFT = \frac{NFT_0}{1+\lambda \times t}$, where $NFT_0$ is an upper bound for NFT; $\lambda$ is a user-defined positive parameter; and $t$ is the iteration counter.

## 3. Variable iterated greedy algorithm VIG_VNS algorithm

A permutation-based representation is employed in the VIG_VNS algorithm, and a solution is represented by the permutation of nodes as $\pi = \{\pi_0, \pi_1, \ldots, \pi_n, \pi_{n+1}\}$. The initial solution is constructed randomly. Then, a VNS algorithm based on forward and backward 1_*Opt* local searches is applied to the initial solution, which is denoted as the *VNS*_1_*Opt* local search from now on. Subsequently, a loop is started and continues until a predetermined stopping criterion is satisfied. Inside the loop, the current solution is destructed and constructed. Again, the *VNS*_1_*Opt* local search is applied to the solution generated after the destruction and construction phase. Note that we use a variant of VIG_FL algorithm where the determination of the destruction size is somewhat different. In other words, we start with $k = 1$ and increase the neighborhood counter by $k = k + 1$, which is the same as in the VIG_FL algorithm. However, we determine the destruction by $d = k \times 5$ to avoid large CPU time requirements. In other words, $d$ is increased by 5, 10, 15, ..., until $d = k_{\max} \times 5$, where $k_{\max}$ is fixed at $k_{\max} = \lfloor n - 1/5 \rfloor$. The outline of the VIG_VNS algorithm is given in Fig. 1.

Regarding the destruction and construction procedure denoted as *DestructConstruct*(), a given number $d$ of nodes are randomly chosen and removed from the solution without repetition in the destruction step. This generates two partial solutions.

```
Procedure VIG_VNS( )
k_max = ⌊n − 1/5⌋
π_0 = GenerateInitialSolution( )   隨機初始化排列
π = VNS_1_Opt(π_0)
π_best = π
while(NotTermination){
    k = 1
    do{
        d = k × 5
        π_1 = DestructConstruct(π, d)   摧毀建構
        π_2 = VNS_1_Opt(π_1)   Local search
        if f(π_2) <_lex f(π) then   有進步
            k = 1
            π = π_2
            if f(π) <_lex f(π_best) then
                π_best = π   更新最好
            endif
        else   沒有進步
            k = k + 1
        endif
    }while(k ≤ k_max)
}endwhile
return π_best
Endprocedure
```

**Fig. 1.** The proposed VIG_VNS algorithm.

The first one with the size $d$ of nodes is denoted as $\pi^R$, including the removed nodes in the order in which they are removed. The second one with the size $n-d$ of nodes is the original solution without the removed nodes, which is denoted as $\pi^D$. Finally, in the construction phase, the second phase of the NEH insertion heuristic is used to complete the solution. To do so, the first node $\pi_1^R$ is inserted into all possible $n-d+1$ positions in the destructed solution $\pi^D$, generating $n-d+1$ partial solutions. Among these $n-d+1$ partial solutions, the best partial solution with the minimum tour length or makespan is chosen and kept for the next iteration. Then, the second node $\pi_2^R$ is considered and so on until $\pi^R$ is empty or a final solution is obtained. Hence, $\pi^D$ is again of size $n$.

As the TSPTW problem is a constrained optimization problem, constraint handling methods play an important role. In the construction step, we employ an adaptive penalty function, the NFT, which is explained in Section 2 in detail. In other words, when a tour is being constructed after the destruction phase, the penalized fitness values are compared, and feasibility conditions are not considered. When applying the *VNS*_1_*Opt* local search to the destructed and constructed solution, we use the superiority of feasible solution (SF) method of Deb [35] when comparing two solutions. It should be noted that "$<_{lex}$" indicates the use of the SF method, which is based on lexicographic ordering where constraint violation and objective function value are distinguished. In addition, we also always use SF when we update the $\pi_{best}$ solution.

As mentioned before, the local search employed in this work is based on the 1_*Opt* neighborhood. In the 1_*Opt* local search, a single node is removed from the tour and is re-inserted in different positions in a forward or backward manner. In both procedures, a node is removed and re-inserted into $n-1$ positions if the insertion point *is feasible*. In other words, the local search phase considers only the *feasible* movements. When inserting node $i$ into position $j$, feasibility is defined as follows:

檢查前項 檢查後項
$$e_{\pi_j} + c(\pi_j, \pi_i) \leqslant l_{\pi_i} \quad \text{and} \quad e_{\pi_i} + c(\pi_i, \pi_{j+1}) \leqslant l_{\pi_{j+1}} \tag{6}$$
起始 中止

Given that the 1_*Opt* local search can be conducted in a backward and forward manner, we developed a simple *VNS*_1_*Opt* local search algorithm using backward and forward insertion schemes as two neighborhood structures. These two neighborhoods are chosen as $N_1(\pi)$ = *Backward*_1_*Opt* and $N_2(\pi)$ = *Forward*_1_*Opt*, respectively. The outline of the *VNS*_1_*Opt* local search is provided in Fig. 2.

In both procedures of the *VNS*_1_*Opt* local search, we consider the "*gain*" and "*feasibility*" at the same time to avoid moving into infeasible solutions. If there is a gain in the move and if it is feasible, we insert node $i$ into position $j$. Then, we compare two solutions with respect to the SF constraint handling method. The outlines of the backward and forward insertion methods are given in Figs. 3 and 4. In addition, an example illustrates the "*gain*" and "*feasibility*" in Appendix A.

## 4. Computational results

We implemented the VIG_VNS algorithm in C++ and conducted all experiments on a computer with an Intel Core i5 processor at 2.53 GHz. We considered five available sets of benchmark instances, where $n$ represents the number of customers. $w$ is the time window width, and *Best* is the best known or optimal value.

1. The first benchmark set consists of 27 classes of five instances each. All instances were proposed and solved to optimality by Dumas et al. [5]. The instance size ranges from 20 to 200 customers.
2. The second benchmark set was proposed by Gendreau et al. [11] and consists of 140 instances grouped into 28 classes with an equal number of customers and time window width. These instances were obtained from the instances proposed

```
Procedure  VNS_1_Opt(π)
k_max = 2
k = 1
do{
    π_1 = N_k(π)   先試後，再試前
    if  f(π1) <_lex f(π)  then  有進步
        k = 1
        π = π_1
    else  無進步
        k = k + 1
}while(k ≤ k_max)
return  π
endprocedure
```

**Fig. 2.** The VNS_1_Opt algorithm.

```
Procedure N1(π) = Backward_1_Opt
    for i = n to 2 do   從後面開始更新
        remove = c(π_{i-1}, π_i) + c(π_i, π_{i+1}) − c(π_{i-1}, π_{i+1})
        for j = i − 1 to 1 do
            add = c(π_{j-1}, π_i) + c(π_i, π_j) − c(π_{j-1}, π_j)
            gain = add − remove
            if (gain < 0) and (isfeasible(π_i, π_j))
                π1 = insert(π, i, j)
                if f(π1) <_lex f(π) then
                    π = π1   有變好就更新
                endif
            endif
        endfor
    endfor
    return π
endprocedure
```

**Fig. 3.** The backward 1_Opt algorithm.

```
Procedure N2(π) = Forward_1_Opt
    for i = 1 to n − 1 do   從前面開始更新
        remove = c(π_{i-1}, π_i) + c(π_i, π_{i+1}) − c(π_{i-1}, π_{i+1})
        for j = i + 1 to n do
            add = c(π_i, π_{j+1}) + c(π_j, π_i) − c(π_j, π_{j+1})
            gain = add − remove
            if (gain < 0) and (isfeasible(π_i, π_j))
                π1 = insert(π, i, j)
                if f(π1) <_lex f(π) then
                    π = π1   有變好就更新
                endif
            endif
        endfor
    endfor
    return π
endprocedure
```

**Fig. 4.** The forward 1_Opt algorithm.

by Dumas et al. [5] by extending the time windows by 100 units, resulting in time windows in the range from 120 to 200 time units.

3. The third benchmark set was proposed by Ohlmann and Thomas [14] and consists of 25 instances grouped into 5 classes. These instances were obtained from the instances with 150 and 200 customers proposed by Dumas et al. [5] by extending the time windows by 100 time units.
4. The fourth benchmark set was proposed by da Silva and Urrutia [17] and consists of 125 instances grouped into 25 classes with 200–400 customers per instance and time windows between 100 and 500 units.
5. The fifth benchmark set was proposed by Potvin and Bengio [39] and consists of 31 problem instances derived from Solomon's RC2 VRPTW [40].

alpha = 2

As for the parameters of the NFT constraint handling method, they are carefully and experimentally fixed at $NFT_0 = 0.001$ and $\lambda = 0.04$, respectively. We performed 25 independent runs for our algorithm, and the statistics over 25 runs for each instance are averaged over the five instances of each instance class. Note that bolded values in all statistics in all tables indicate the better results than comparison algorithms.

### 4.1. Computational results for Dumas's instances

The computational results of the VIG_VNS algorithm for Dumas's instances are given in Table 1. In terms of solution quality, the VIG_VNS and GVNS algorithms were able to find all the best or optimal solutions and achieve a feasible solution in all runs except for instance *n150w60*. For instances *n40w80*, *n60w80* and *n200w20*, the VIG_VNS algorithm generated slightly better average values than the GVNS algorithm. In addition, the average $\sigma$ values indicate that the VIG_VNS algorithm was more robust than the GVNS algorithm. However, we performed a nonparametric Mann–Whitney test with a confidence level $\alpha = 0.05$ to determine the equality of population medians in terms of *Avg* values. The test statistic $W = 741.5$ has a $p$-value of 0.9931. As the $p$-value is not less than the chosen $\alpha$ level of 0.05, we can conclude that there is no difference between the population medians. In other words, the two algorithms were equivalent.

### 4.2. Computational results for Gendreau's instances

The computational results of the VIG_VNS algorithm for Gendreau's instances are given in Table 2, where the VIG_VNS algorithm is compared to the best-known results in the literature because optimal solutions are not known for these instances. These best-known solutions include the results improved by the Beam_ACO in [16]. However, both the VIG_VNS and GVNS algorithms were able to further improve on the best-known results. In particular, new best known solutions were found by the VIG_VNS and GVNS algorithms for the instances *n80w100*, *n80w140*, *n100w120*, *n100w140*, *n100w160* and *n100w200*. When comparing VIG_VNS with GVNS, the VIG_VNS algorithm generated better average results for 13 out of 28 instance classes. The smaller standard deviation of the VIG_VNS algorithm compared with that of the GVNS algorithm indicates the former is a more robust algorithm than the latter. Briefly, both the VIG_VNS and GVNS algorithms provide new best known solutions for 6 out of 28 instance classes. In addition, VIG_VNS provides new best solutions for *n100w100* instances. Again, we performed a nonparametric Mann–Whitney test with a confidence level $\alpha = 0.05$ to determine the equality of population medians in terms of *Avg* values. The test statistic $W = 792.5$ has a $p$-value of 0.9347. As

**Table 1**
Comparison between VIG_VNS and GVNS on instances by Dumas et al. [5]. $T_{max} = 60$ s.

| Instance | | | GVNS | | | | | VIG_VNS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $w$ | Optimal | Best | Avg | Avg $\sigma$ | Avg Sec | Avg $\sigma$ Sec | Best | Avg | Avg $\sigma$ | Avg Sec | Avg $\sigma$ Sec |
| 20 | 20 | 361.2 | 361.2 | 361.2 | 0.0 | 0.2 | 0.0 | 361.2 | 361.2 | 0.0 | 0.0 | 0.0 |
| | 40 | 316.0 | 316.0 | 316.0 | 0.0 | 0.2 | 0.0 | 316.0 | 316.0 | 0.0 | 0.0 | 0.0 |
| | 60 | 309.8 | 309.8 | 309.8 | 0.0 | 0.2 | 0.0 | 309.8 | 309.8 | 0.0 | 0.0 | 0.0 |
| | 80 | 311.0 | 311.0 | 311.0 | 0.0 | 0.3 | 0.0 | 311.0 | 311.0 | 0.0 | 0.0 | 0.0 |
| | 100 | 275.2 | 275.2 | 275.2 | 0.0 | 0.3 | 0.0 | 275.2 | 275.2 | 0.0 | 0.0 | 0.0 |
| 40 | 20 | 486.6 | 486.6 | 486.6 | 0.0 | 0.3 | 0.0 | 486.6 | 486.6 | 0.0 | 0.0 | 0.0 |
| | 40 | 461.0 | 461.0 | 461.0 | 0.0 | 0.4 | 0.0 | 461.0 | 461.0 | 0.0 | 0.0 | 0.0 |
| | 60 | 416.4 | 416.4 | 416.4 | 0.0 | 0.5 | 0.0 | 416.4 | 416.4 | 0.0 | 0.0 | 0.0 |
| | 80 | 399.8 | 399.8 | 399.9 | 0.4 | 0.5 | 0.0 | 399.8 | **399.8** | **0.0** | 0.0 | 0.0 |
| | 100 | 377.0 | 377.0 | 377.0 | 0.2 | 0.6 | 0.0 | 377.0 | 377.0 | **0.0** | 0.0 | 0.0 |
| 60 | 20 | 581.6 | 581.6 | 581.6 | 0.0 | 0.6 | 0.0 | 581.6 | 581.6 | 0.0 | 0.0 | 0.0 |
| | 40 | 590.2 | 590.2 | 590.2 | 0.0 | 0.8 | 0.0 | 590.2 | 590.2 | 0.0 | 0.0 | 0.0 |
| | 60 | 560.0 | 560.0 | 560.0 | 0.0 | 0.9 | 0.0 | 560.0 | 560.0 | 0.0 | 0.0 | 0.0 |
| | 80 | 508.0 | 508.0 | 508.1 | 0.2 | 1.2 | 0.0 | 508.0 | **508.0** | **0.0** | 0.0 | 0.0 |
| | 100 | 514.8 | 514.8 | 514.8 | 0.0 | 1.3 | 0.0 | 514.8 | 514.8 | 0.0 | 0.1 | 0.3 |
| 80 | 20 | 676.6 | 676.6 | 676.6 | 0.0 | 0.9 | 0.0 | 676.6 | 676.6 | 0.0 | 0.0 | 0.0 |
| | 40 | 630.0 | 630.0 | 630.0 | 0.0 | 1.3 | 0.0 | 630.0 | 630.0 | 0.0 | 0.3 | 0.4 |
| | 60 | 606.4 | 606.4 | 606.4 | 0.1 | 1.8 | 0.1 | 606.4 | 606.4 | **0.0** | 0.1 | 0.2 |
| | 80 | 593.8 | 593.8 | 593.8 | 0.1 | 2.1 | 0.1 | 593.8 | 593.8 | **0.0** | 1.6 | 2.0 |
| 100 | 20 | 757.6 | 757.6 | 757.6 | 0.0 | 1.4 | 0.0 | 757.6 | 757.6 | 0.0 | 0.2 | 0.7 |
| | 40 | 701.8 | 701.8 | 701.8 | 0.0 | 1.9 | 0.1 | 701.8 | 701.8 | 0.0 | 0.0 | 0.1 |
| | 60 | 696.6 | 696.6 | 696.6 | 0.0 | 2.7 | 0.1 | 696.6 | 696.6 | 0.0 | 0.1 | 0.2 |
| 150 | 20 | 868.4 | 868.4 | 868.4 | 0.0 | 3.6 | 0.3 | 868.4 | 868.4 | 0.0 | 0.7 | 1.0 |
| | 40 | 834.8 | 834.8 | 834.8 | 0.0 | 5.3 | 0.3 | 834.8 | 834.8 | 0.0 | 1.7 | 2.0 |
| | 60 | 805.0 | 818.6 | 818.6 | 0.1 | 7.4 | 0.7 | 818.6 | 818.6 | **0.0** | 6.2 | 6.2 |
| 200 | 20 | 1009.0 | 1009.0 | 1009.1 | 0.1 | 8.5 | 0.5 | 1009.0 | **1009.0** | 0.1 | 6.0 | 5.7 |
| | 40 | 984.2 | 984.2 | **984.2** | **0.1** | 12.6 | 0.8 | 984.2 | 984.3 | 0.3 | 9.6 | 9.7 |

**Table 2**
Comparison between VIG_VNS and GVNS on instances by Gendreau et al. [11]. $T_{max}$ = 60 s.

| Instance | | | GVNS | | | | | VIG_VNS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *n* | *w* | Best known | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec |
| 20 | 120 | 265.6 | 265.6 | 265.6 | 0.0 | 0.3 | 0.0 | 265.6 | 265.6 | 0.0 | 0.0 | 0.0 |
| | 140 | 232.8 | 232.8 | 232.8 | 0.0 | 0.3 | 0.0 | 232.8 | 232.8 | 0.0 | 0.0 | 0.0 |
| | 160 | 218.2 | 218.2 | 218.2 | 0.0 | 0.3 | 0.0 | 218.2 | 218.2 | 0.0 | 0.0 | 0.0 |
| | 180 | 236.6 | 236.6 | 236.6 | 0.0 | 0.4 | 0.0 | 236.6 | 236.6 | 0.0 | 0.0 | 0.0 |
| | 200 | 241.0 | 241.0 | 241.0 | 0.0 | 0.4 | 0.0 | 241.0 | 241.0 | 0.0 | 0.0 | 0.0 |
| 40 | 120 | 360.0 | 377.8 | 377.8 | 0.0 | 0.8 | 0.0 | 377.8 | 377.8 | 0.0 | 0.0 | 0.0 |
| | 140 | 348.4 | 364.4 | 364.4 | 0.0 | 0.8 | 0.0 | 364.4 | 364.4 | 0.0 | 0.0 | 0.0 |
| | 160 | 326.8 | 326.8 | 326.8 | 0.0 | 0.9 | 0.0 | 326.8 | 326.8 | 0.0 | 0.0 | 0.0 |
| | 180 | 326.8 | 330.4 | 331.3 | 0.8 | 1.0 | 0.0 | 330.4 | **330.4** | **0.0** | 0.0 | 0.0 |
| | 200 | 313.8 | 313.8 | 314.3 | 0.4 | 1.0 | 0.1 | 313.8 | **313.8** | **0.0** | 0.0 | 0.0 |
| 60 | 120 | 451.0 | 451.0 | 451.0 | 0.1 | 1.5 | 0.1 | 451.0 | 451.0 | 0.0 | 0.1 | 0.2 |
| | 140 | **452.0**[a] | 452.0 | 452.1 | 0.2 | 1.7 | 0.1 | 452.0 | **452.0** | **0.0** | 0.3 | 0.5 |
| | 160 | 448.6 | 464.0 | 464.5 | 0.2 | 1.7 | 0.0 | 464.0 | **464.0** | **0.0** | 0.6 | 1.0 |
| | 180 | **421.2**[a] | 421.2 | 421.2 | 0.1 | 2.2 | 0.1 | 421.2 | 421.2 | **0.0** | 0.1 | 0.3 |
| | 200 | 427.4 | 427.4 | 427.4 | 0.0 | 2.4 | 0.1 | 427.4 | 427.4 | 0.0 | 0.0 | 0.2 |
| 80 | 100 | **578.8**[a] | 578.6 | 578.7 | 0.2 | 2.3 | 0.1 | 578.6 | **578.6** | **0.0** | 0.8 | 1.0 |
| | 120 | 541.4 | 541.4 | 541.4 | 0.0 | 2.7 | 0.1 | 541.4 | 541.4 | 0.0 | 2.8 | 3.7 |
| | 140 | **506.8**[a] | 506.0 | 506.3 | 0.2 | 3.2 | 0.3 | 506.0 | **506.0** | **0.1** | 2.5 | 3.5 |
| | 160 | 502.8 | 504.8 | 505.5 | 0.7 | 3.3 | 0.1 | 504.8 | **504.8** | **0.0** | 0.5 | 0.9 |
| | 180 | 489.0 | 500.6 | 501.2 | 0.9 | 3.7 | 0.1 | 500.6 | **500.6** | **0.0** | 3.7 | 4.8 |
| | 200 | 481.4 | 481.4 | 481.8 | **0.1** | 4.2 | 0.2 | 481.4 | **481.7** | 0.2 | 3.2 | 5.7 |
| 100 | 80 | 666.4 | 666.4 | 666.6 | 0.2 | 3.1 | 0.2 | 666.4 | **666.4** | **0.0** | 0.2 | 0.4 |
| | 100 | 642.0 | 642.0 | 642.1 | 0.1 | 3.7 | 0.1 | **640.6** | **640.6** | **0.1** | 1.7 | 3.5 |
| | 120 | **599.4**[a] | 597.2 | 597.5 | 0.3 | 4.1 | 0.2 | 597.2 | **597.2** | **0.1** | 8.1 | 8.0 |
| | 140 | **550.2**[a] | 548.4 | 548.4 | 0.0 | 4.4 | 0.2 | 548.4 | 548.4 | 0.0 | 1.9 | 3.0 |
| | 160 | **556.6**[a] | 555.0 | **555.0** | 0.1 | 5.1 | 0.2 | 555.0 | 555.2 | 0.1 | 1.3 | 1.4 |
| | 180 | 561.6 | 561.6 | **561.6** | **0.0** | 6.3 | 0.3 | 561.6 | 561.7 | 0.1 | 2.3 | 4.7 |
| | 200 | 555.4 | 550.2 | 551.0 | 1.2 | 6.8 | 0.3 | 550.2 | **550.2** | **0.0** | 5.6 | 6.8 |

[a] Best known results obtained in [16].

the *p*-value is not less than the chosen $\alpha$ level of 0.05, we can conclude that there is no difference between the population medians. In other words, the two algorithms were equivalent.

For the remaining sets of benchmark instances, we fixed the maximum CPU time to 100 s because they were more difficult instances than the previous two benchmark sets.

### 4.3. Computational results for Ohlmann and Tomas's instances

The computational results of the VIG_VNS algorithm for Ohlmann and Tomas's instances are given in Table 3, where the VIG_VNS algorithm is compared to the GVNS algorithm. Note that these best-known solutions include the results improved by the Beam_ACO algorithm in [16]. As seen in Table 3, the VIG_VNS and GVNS algorithms were able to further improve all the results and present new best-known solutions for these classes of instances. When comparing the two best-performing algorithms, the GVNS algorithm was slightly better than the VIG_VNS algorithm in terms of average values, indicating that when the problem size gets larger, the performance of the VIG_VNS algorithm degrades slightly. Again, we performed a non-parametric Mann–Whitney test with a confidence level $\alpha = 0.05$ to determine the equality of population medians in terms of *Avg* values. The test statistic $W = 29$ has a *p*-value of 0.8345. Given that the *p*-value is not less than the chosen $\alpha$ level of 0.05, we can conclude that there is no difference between the population medians. In other words, the two algorithms were equivalent.

**Table 3**
Comparison between VIG_VNS and GVNS on instances by Ohlmann and Thomas [14]. $T_{max}$ = 100 s.

| Instance | | | GVNS | | | | | VIG_VNS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *n* | *w* | Best known | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec |
| 150 | 120 | 724.0[a] | 722.0 | **722.3** | **0.4** | 11.8 | 0.3 | 722.0 | 722.5 | 0.8 | 29.3 | 35.3 |
| | 140 | 697.2[a] | 693.8 | 694.8 | **0.5** | 13.3 | 0.5 | 693.8 | **694.6** | 1.0 | 27.2 | 34.7 |
| | 160 | 672.6[a] | 671.0 | **671.2** | **0.3** | 15.0 | 0.8 | 671.0 | 672.0 | 1.5 | 22.9 | 29.4 |
| 200 | 120 | 806.4[a] | 803.6 | **803.9** | **0.1** | 30.3 | 2.0 | 803.6 | 804.8 | 1.8 | 48.6 | 48.6 |
| | 140 | 802.4[a] | 798.0 | **799.5** | **1.1** | 38.0 | 1.8 | 798.0 | 802.0 | 4.6 | 48.8 | 49.3 |

[a] Best known results obtained in [16].

### 4.4. Computational results for Silva and Urrutia's instances

The computational results of the VIG_VNS algorithm for da Silva and Urrutia's instances are given in Table 4, where the VIG_VNS algorithm is compared to the GVNS algorithm. The VIG_VNS algorithm was not able to generate feasible solutions in 28 out of 3125 runs. In other words, each problem class has 5 instances, and there are 25 classes, which makes a total of 125 problem instances. As we had 25 replications for each instance, we conducted 3125 runs for the 25 classes in total. Among them, only 28 replications ended up with infeasible solutions as indicated by "a" in Table 4. In terms of generating feasible solutions, the GVNS algorithm was better than the VIG_VNS algorithm. However, the VIG_VNS algorithm was also able to further improve 12 out of 25 instance classes and find the same results for 10 instance classes. The GVNS algorithm outperformed the other algorithm in only 3 out of 25 instance classes. However, we performed a nonparametric Mann–Whitney test with a confidence level $\alpha = 0.05$ to determine the equality of population medians in terms of *Avg* values. The test statistic $W = 639.5$ has a $p$-value of 0.9768. As the $p$-value is not less than the chosen $\alpha$ level of 0.05, we can conclude that there is no difference between the population medians. In other words, the two algorithms were equivalent.

To determine the peak performance of the VIG_VNS algorithm for these new benchmark set, we fix the maximum CPU time to 300 s for each run. The computational results are given in Table 5 where new best known solutions are provided for 14 out of 25 instance classes. Note that the number of infeasible solutions are decreased considerably at the expense of increased CPU times. Again, we carried out a nonparametric Mann–Whitney test with a confidence level $\alpha = 0.05$ to see the equality of population medians in terms of *Avg* values. The test statistic $W = 631.5$ has a $p$-value of 0.9150. Because the $p$-value is not less than the chosen $\alpha$ level of 0.05, we can conclude that there is no difference between the population medians. In other words, two algorithms were equivalent.

Regarding the CPU times in general, it is clear that the GVNS algorithm was superior to the VIG_VNS algorithm as it used the neighborhood partitioning improvement method in their local search. In addition, the parameter −03 was used in compilation to optimize their code. Because our aim is to demonstrate that a simple variable iterated greedy algorithm can solve the TSPTW, we focused on the solution quality rather than the computational time. All permutations of new best known solutions can be found at http://kkarabulut.yasar.edu.tr/tsptw.

In addition to the above, one might wonder about the performance of pure VIG_VNS algorithm without local search. For this purpose, we fixed the maximum CPU time to 300 s and ran the n400w500 instance class with and without local search. The computational results are given in Table 6 where the performance of the VIG_VNS algorithm using only the destruction and construction procedure was quite remarkable. When considering the overall average values, the relative percent deviation was only 0.86 percent from the best average value with local search (i.e., $(12724.6 - 12833.4) * 100/12724.6 = -0.86$).

**Table 4**
Comparison between VIG_VNS and GVNS on instances by da Silva and Urrutia [17]. $T_{max} = 100$ s.

| Instance | | GVNS | | | | | VIG_VNS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $w$ | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec |
| 200 | 100 | 10019.6 | 10019.6 | 0.1 | 4.8 | 0.3 | 10019.6 | 10019.6 | **0.0** | 0.1 | 0.3 |
| | 200 | 9252.0 | 9254.1 | 7.2 | 5.8 | 0.2 | 9252.0 | **9252.0** | **0.0** | 6.7 | 8.0 |
| | 300 | 8026.4 | 8034.3 | 4.5 | 7.2 | 0.2 | **8022.8** | **8024.3** | **2.7** | 25.6 | 23.0 |
| | 400 | 7067.2 | 7079.3 | **4.4** | 8.7 | 0.4 | **7062.4** | **7069.3** | 6.0 | 27.0 | 20.4 |
| | 500 | 6466.4 | **6474.0** | 5.1 | 10.0 | 0.3 | **6466.2** | 6472.4 | 9.1 | 32.3 | 23.4 |
| 250 | 100 | 12633.0 | 12633.0 | 0.0 | 9.9 | 0.2 | 12633.0 | 12633.0 | 0.0 | 0.5 | 1.0 |
| | 200 | 11310.4 | **11310.7** | **0.7** | 11.9 | 0.4 | 11310.4 | 11311.7 | 5.6 | 18.2 | 15.8 |
| | 300 | 10230.4 | 10235.1 | 2.8 | 14.9 | 0.6 | 10230.4 | **10233.5** | **1.7** | 26.0 | 20.5 |
| | 400 | 8899.2 | 8908.5 | 4.1 | 18.9 | 0.7 | **8896.2** | **8903.6** | 7.5[a] | 48.5 | 25.3 |
| | 500 | 8082.4 | **6474.0**[A] | **6.7** | 20.7 | 0.9 | **8066.4** | **8088.5** | 14.8 | 52.3 | 24.6 |
| 300 | 100 | 15041.2 | 15041.2 | 0.0 | 21.2 | 0.7 | 15041.2 | 15041.2 | 0.0 | 0.7 | 0.9 |
| | 200 | **13846.8** | 13853.1 | 2.3 | 23.7 | 0.6 | 13851.4 | **13852.0** | **1.5** | 21.4 | 14.3 |
| | 300 | 11477.6 | 11488.5 | **5.2** | 37.0 | 3.8 | **11477.2** | **11481.0** | 10.1 | 47.5 | 23.3 |
| | 400 | 10413.0 | 10437.4 | 12.9 | 31.7 | 1.2 | **10402.8** | **10413.3** | **12.3** | 59.8 | 21.1 |
| | 500 | 9861.8 | 9876.7 | **8.9** | 35.4 | 1.1 | **9844.2** | **9872.5** | 20.5 | 63.5 | 20.8 |
| 350 | 100 | 17494.0 | 17494.0 | 0.0 | 41.0 | 2.5 | 17494.0 | 17494.0 | 0.0 | 5.3 | 6.4 |
| | 200 | 15672.0 | **15672.2** | **0.6** | 47.3 | 2.1 | 15672.0 | 15674.6 | 5.5[a] | 41.9 | 21.6 |
| | 300 | 13650.2 | **13654.1** | **1.7** | 54.9 | 2.2 | **13648.8** | 13658.8 | 13.4[a] | 60.7 | 26.0 |
| | 400 | 12099.0 | **12119.6** | **8.9** | 60.2 | 2.8 | **12090.2** | 12143.4 | 42.2[a] | 76.1 | 21.5 |
| | 500 | 11365.8 | **11388.2** | **12.0** | 57.8 | 1.2 | **11350.6** | 11399.4 | 36.9 | 75.5 | 20.7 |
| 400 | 100 | 19454.8 | 19454.8 | 0.0 | 57.1 | 0.6 | 19454.8 | 19454.8 | 0.0 | 4.9 | 5.7 |
| | 200 | 18439.8 | **18439.9** | **0.6** | 66.9 | 1.9 | 18439.8 | 18440.5 | 1.8[a] | 38.8 | 22.3 |
| | 300 | **15873.4** | **15879.1** | **3.0** | 93.6 | 7.9 | 15876.4 | 15891.6 | 16.0 | 69.3 | 19.3 |
| | 400 | **14115.4** | **14145.5** | 12.9 | 96.2 | 3.9 | 14125.6 | 14184.9 | 50.8[a] | 80.6 | 20.7 |
| | 500 | 12747.6 | **12766.2** | **9.7** | 109.3 | 4.4 | **12736.0** | 12802.0 | 42.5[a] | 85.8 | 16.2 |

[A] Must be mistyped in [17] because average cannot be smaller than best value reported.
[a] Averaged after removing infeasible solutions (total of 28 out of 3125 runs).

**Table 5**
Peak performance of VIG_VNS compared to GVNS on instances by da Silva and Urrutia [17]. $T_{max}$ = 300 s.

| Instance | | GVNS | | | | | VIG_VNS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $w$ | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec |
| 200 | 100 | 10019.6 | 10019.6 | 0.1 | 4.8 | 0.3 | 10019.6 | 10019.6 | **0.0** | 0.2 | 0.3 |
| | 200 | 9252.0 | 9254.1 | 7.2 | 5.8 | 0.2 | 9252.0 | **9252.0** | **0.0** | 6.5 | 7.9 |
| | 300 | 8026.4 | 8034.3 | 4.5 | 7.2 | 0.2 | **8022.8** | **8023.5** | **1.8** | 44.5 | 45.8 |
| | 400 | 7067.2 | 7079.3 | 4.4 | 8.7 | 0.4 | **7062.4** | **7066.0** | **3.9** | 66.6 | 64.8 |
| | 500 | 6466.4 | 6474.0 | 5.1 | 10.0 | 0.3 | **6466.2** | **6467.8** | **4.7** | 70.3 | 66.2 |
| 250 | 100 | 12633.0 | 12633.0 | 0.0 | 9.9 | 0.2 | 12633.0 | 12633.0 | 0.0 | 0.5 | 0.9 |
| | 200 | 11310.4 | 11310.7 | 0.7 | 11.9 | 0.4 | 11310.4 | **11310.4** | **0.1** | 24.6 | 24.0 |
| | 300 | 10230.4 | 10235.1 | 2.8 | 14.9 | 0.6 | 10230.4 | **10231.6** | **1.8** | 50.0 | 39.7 |
| | 400 | 8899.2 | 8908.5 | **4.1** | 18.9 | 0.7 | **8896.2** | **8899.3** | 4.6 | 97.0 | 67.3 |
| | 500 | 8082.4 | **6474.0**[A] | **6.7** | 20.7 | 0.9 | **8066.4** | **8078.8** | 10.6 | 126.3 | 79.7 |
| 300 | 100 | 15041.2 | 15041.2 | 0.0 | 21.2 | 0.7 | 15041.2 | 15041.2 | 0.0 | 0.7 | 0.9 |
| | 200 | **13846.8** | 13853.1 | 2.3 | 23.7 | 0.6 | 13851.4 | **13851.4** | **0.0** | 24.5 | 18.6 |
| | 300 | 11477.6 | 11488.5 | 5.2 | 37.0 | 3.8 | **11477.2** | **11477.2** | **0.2** | 79.9 | 55.1 |
| | 400 | 10413.0 | 10437.4 | 12.9 | 31.7 | 1.2 | **10402.8** | **10405.2** | **5.9** | 104.3 | 61.8 |
| | 500 | 9861.8 | 9876.7 | **8.9** | 35.4 | 1.1 | **9842.2** | **9854.9** | 13.2 | 161.4 | 74.7 |
| 350 | 100 | 17494.0 | 17494.0 | 0.0 | 41.0 | 2.5 | 17494.0 | 17494.0 | 0.0 | 5.2 | 6.4 |
| | 200 | 15672.0 | **15672.2** | **0.6** | 47.3 | 2.1 | 15672.0 | 15672.4 | 1.2[a] | 65.1 | 50.1 |
| | 300 | 13650.2 | 13654.1 | **1.7** | 54.9 | 2.2 | **13648.8** | **13652.9** | 3.7 | 121.4 | 74.9 |
| | 400 | 12099.0 | 12119.6 | **8.9** | 60.2 | 2.8 | **12083.2** | **12103.2** | 18.0 | 193.2 | 62.8 |
| | 500 | 11365.8 | 11388.2 | **12.0** | 57.8 | 1.2 | **11347.8** | **11366.7** | 19.9 | 177.9 | 69.7 |
| 400 | 100 | 19454.8 | 19454.8 | 0.0 | 57.1 | 0.6 | 19454.8 | 19454.8 | 0.0 | 4.9 | 5.7 |
| | 200 | 18439.8 | 18439.9 | 0.6 | 66.9 | 1.9 | 18439.8 | **18439.8** | **0.0** | 45.3 | 31.3 |
| | 300 | 15873.4 | 15879.1 | **3.0** | 93.6 | 7.9 | **15872.8** | **15878.6** | 5.9 | 134.2 | 60.7 |
| | 400 | 14155.4 | 14145.5 | **12.9** | 96.2 | 3.9 | **14086.6** | **14130.1** | 32.4[a] | 213.3 | 68.7 |
| | 500 | 12747.6 | 12766.2 | **9.7** | 109.3 | 4.4 | **12722.4** | **12753.5** | 24.3[a] | 221.6 | 58.1 |

[A] Must be mistyped in [17] because average cannot be smaller than best value reported.
[a] Averaged after removing infeasible solutions (total of 4 out of 3125 runs).

This indicates the fact that a simple IG algorithm was able to generate results no worse than 0.86% from the best results obtained by the local search. Again, we performed a nonparametric Mann–Whitney test with a confidence level $\alpha = 0.05$ to determine the equality of population medians in terms of *Avg* values. The test statistic $W = 25.0$ has a $p$-value of 0.6765. As the $p$-value is not less than the chosen $\alpha$ level of 0.05, we can conclude that there is no difference between the population medians. In other words, VIG without a VNS local search was statistically equivalent to the VIG_VNS algorithm.

Again, one might wonder about the performance of the VIG_FL algorithm as compared with the VIG_VNS algorithm. For this purpose, we fixed the maximum CPU time to 300 s and ran the n400w500 instance class with the local search. The computational results are given in Table 7, which indicates that the performance of the VIG_FL algorithm was slightly worse than the VIG_VNS algorithm in overall average. The problem with the VIG_FL algorithm is because it considers each increment of the destruction size until $d = n - 1$. As it is inside the VNS loop, it consumes more CPU time than the VIG_VNS algorithm, as seen in Avg run sec column indicating the total CPU time spent. It is also interesting to see that the VIG_FL algorithm was able to carry out only 2.5 iterations overall on average, whereas the VIG_VNS algorithm was able to iterate almost 60 times overall on average. However, Table 7 results indicate that the VIG_FL algorithm was very competitive with the VIG_VNS algorithm as indicated by the relative percent deviation being only 0.094 percent from the best average value with local search (i.e., $(12724.6 - 12736.6) * 100/12724.6 = -0.094$) when considering the overall average values. We performed a nonparametric Mann–Whitney test with a confidence level $\alpha = 0.05$ to determine the equality of population medians in terms of *Avg* values. The test statistic $W = 26.0$ has a $p$-value of 0.8345. As the $p$-value is not less than the chosen $\alpha$ level of 0.05, we can conclude that there is no difference between the population medians. In other words, VIG_FL with a VNS local search was statistically equivalent to the VIG_VNS algorithm.

**Table 6**
Impact of the VNS local search. $T_{max}$ = 300 s.

| VIG (without VNS local search) | | | | | | | VIG_VNS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec | Avg Iter | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec | Avg Iter |
| n400w500.001 | 12261 | 12322.2 | 32.4 | 247.1 | 41.8 | 65.3 | 12141 | 12169.6 | 23.1 | 223.7 | 58.6 | 59.0 |
| n400w500.002 | 13144 | 13195.9 | 37.7 | 219.2 | 72.0 | 66.3 | 13015 | 13043.3 | 22.1 | 232.7 | 51.4 | 61.0 |
| n400w500.003 | 13357 | 13445.3 | 67.2 | 249.8 | 53.8 | 67.4 | 13273 | 13293.5 | 13.2 | 234.5 | 52.4 | 58.4 |
| n400w500.004 | 12515 | 12665.3 | 88.5 | 232.1 | 50.0 | 68.6 | 12351 | 12381.6 | 30.3 | 244.9 | 35.3 | 57.6 |
| n400w500.005 | 12890 | 12990.3 | 92.1 | 252.0 | 50.5 | 67.6 | 12843 | 12897.6 | 46.1 | 184.8 | 64.2 | 62.0 |
| Overall Avg | 12833.4 | 12923.8 | 63.6 | 240.0 | 53.6 | 67.0 | **12724.6** | **12757.1** | **27.0** | **224.1** | **52.4** | **59.6** |

**Table 7**
Comparison between VIG_VNS and VIG_FL with local search. $T_{max}$ = 300 s.

| VIG_FL (with VNS local search) | | | | | | | VIG_VNS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec | Avg Iter | Best | Avg | Avg $\sigma$ | Avg sec | Avg $\sigma$ sec | Avg Iter |
| n400w500.001 | 12144 | 12204.9 | 42.8 | 220.5 | 339.9 | 2.2 | 12141 | 12169.6 | 23.1 | 223.7 | 301.4 | 59.0 |
| n400w500.002 | 13031 | 13073.4 | 28.0 | 228.1 | 366.0 | 2.8 | 13015 | 13043.3 | 22.1 | 232.7 | 301.5 | 61.0 |
| n400w500.003 | 13296 | 13329.3 | 37.6 | 233.0 | 351.4 | 2.5 | 13273 | 13293.5 | 13.2 | 234.5 | 301.5 | 58.4 |
| n400w500.004 | 12356 | 12417.1 | 40.3 | 228.3 | 346.0 | 2.2 | 12351 | 12381.6 | 30.3 | 244.9 | 302.0 | 57.6 |
| n400w500.005 | 12856 | 12886.4 | 36.3 | 171.1 | 358.2 | 2.9 | 12843 | 12897.6 | 46.1 | 184.8 | 300.9 | 62.0 |
| Overall Avg | 12736.6 | 12782.2 | 37.0 | **216.2** | 352.3 | 2.5 | **12724.6** | **12757.1** | **27.0** | 224.1 | **301.5** | **59.6** |

### 4.5. Computational results for Makespan criterion

For the fifth benchmark set with the makespan criterion, we fixed the maximum CPU time to 100 s for each run. We compared to the ACO algorithm in Cheng and Mao in [15] for 31 problem instances proposed by Potvin and Bengio [39]. The computational results are given in Table 8, which indicates that substantial improvements are achieved. The VIG_VNS algorithm was able to further improve the best-known results provided by Cheng and Mao in [15] for 15 out of 31 problem instances, whereas the ACO algorithm was slightly better in 10 instances. The VIG_VNS algorithm was not able to generate a feasible solution for 3 out of 25 runs for instance rc204.1, whereas the ACO algorithm could not find any feasible solution for 3 problem instances denoted by $\infty$ in Table 8. In addition, the VIG_VNS algorithm was much faster than the ACO algorithm. We performed a nonparametric Mann–Whitney test with a confidence level $\alpha = 0.05$ to determine the equality of population medians in terms of *Avg* values. The test statistic $W = 880.5$ has a $p$-value of 0.1788. As the $p$-value is not less than the chosen $\alpha$ level of 0.05, we can conclude that there is no difference between the population medians. In other words, VIG_VNS was statistically equivalent to the ACO algorithm. Again, all permutations of new best known solutions can be found at http://kkarabulut.yasar.edu.tr/tsptw.

**Table 8**
Comparison between VIG_VNS and ACO [15] for makespan criterion. $T_{max}$ = 100 s.

| Instance | $n$ | VIG_VNS | | | ACO | | |
|---|---|---|---|---|---|---|---|
| | | Avg | Best | CPU | Avg | Best | CPU |
| rc201.1 | 20 | 592.06 | 592.06 | 0.00 | 592.06 | 592.06 | 100.94 |
| rc201.2 | 26 | **869.90** | 869.90 | 0.00 | 877.49 | **861.91** | 246.26 |
| rc201.3 | 32 | **854.12** | 854.12 | 0.00 | 867.61 | **853.71** | 464.30 |
| rc201.4 | 26 | **889.18** | **889.18** | 0.00 | 900.52 | 900.38 | 151.05 |
| rc202.1 | 33 | **850.48** | **850.48** | 1.96 | 880.74 | 871.11 | 241.77 |
| rc202.2 | 14 | 342.20 | 342.20 | 0.00 | **338.52** | **338.52** | 46.77 |
| rc202.3 | 29 | 904.48 | 904.48 | 3.00 | **892.18** | **847.31** | 190.24 |
| rc202.4 | 28 | **854.12** | **854.12** | 0.00 | $\infty$ | $\infty$ | – |
| rc203.1 | 19 | **488.42** | **488.42** | 0.00 | 673.07 | 663.66 | 78.83 |
| rc203.2 | 33 | **853.71** | **853.71** | 0.08 | 926.75 | 897.88 | 255.77 |
| rc203.3 | 37 | **956.92** | **956.92** | 2.12 | $\infty$ | $\infty$ | – |
| rc203.4 | 15 | **350.83** | **350.83** | 0.00 | 493.85 | 493.85 | 53.08 |
| rc204.1 | 46 | 950.36[a] | 950.36 | 29.92 | **949.68** | **949.22** | 438.25 |
| rc204.2 | 33 | **701.62** | **701.62** | **0.80** | 863.65 | 821.63 | 240.55 |
| rc204.3 | 24 | **455.03** | **455.03** | **0.00** | 642.06 | 635.36 | 127.27 |
| rc204.4 | 14 | **426.13** | 426.13 | 0.00 | 428.39 | 425.20 | 46.75 |
| rc205.1 | 14 | 455.94 | 455.94 | 0.00 | **422.24** | **417.81** | 46.89 |
| rc205.2 | 27 | 820.19 | 820.19 | 0.00 | 820.19 | 820.19 | 181.06 |
| rc205.3 | 35 | 950.05 | 950.05 | 1.92 | 950.05 | 950.05 | 274.55 |
| rc205.4 | 28 | **867.13** | 867.13 | 1.40 | 870.43 | **850.99** | 186.56 |
| rc206.1 | 4 | 117.85 | 117.85 | 0.00 | 117.85 | 117.85 | 13.27 |
| rc206.2 | 37 | 917.26 | 917.26 | 2.56 | **914.99** | **909.30** | 306.13 |
| rc206.3 | 25 | 661.07 | 661.07 | 0.00 | **650.59** | **650.59** | 140.72 |
| rc206.4 | 38 | **930.10** | **930.10** | 1.80 | 943.31 | 943.31 | 320.19 |
| rc207.1 | 34 | 865.07 | 865.07 | 1.04 | **860.98** | **851.06** | 258.44 |
| rc207.2 | 31 | **735.56** | **735.56** | 0.08 | $\infty$ | $\infty$ | – |
| rc207.3 | 33 | **800.39** | **800.39** | **0.08** | 955.7 | 944.52 | 241.70 |
| rc207.4 | 6 | 133.14 | 133.14 | 0.00 | 133.14 | 133.14 | 22.45 |
| rc208.1 | 38 | **841.28** | **841.06** | 12.20 | 934.8 | 925.36 | 334.72 |
| rc208.2 | 29 | **644.13** | **644.13** | 0.08 | 722.24 | 712.96 | 185.73 |
| rc208.3 | 36 | **747.15** | **747.15** | 0.08 | 795.03 | 774.72 | 291.98 |

[$\infty$] No feasible solution found.
[a] Average of 22 runs without 3 infeasible solutions.

## 5. Conclusions

This paper presents a VIG_VNS algorithm for the TSPTW. The outcomes of the paper can be summarized as follows. For the first benchmark set, both the VIG_VNS and GVNS algorithms were able to find all the optimal solutions. For the second benchmark set, both the VIG_VNS and GVNS algorithms provided new best-known solutions for 6 out of 28 instance classes. For the third benchmark set, both the VIG_VNS and GVNS algorithms provided new best-known solutions for all the instance classes. However, for the fourth benchmark set, the VIG_VNS algorithm was able to further improve 14 out of 25 instance classes at the expense of increased CPU times. In terms of solution quality, the VIG_VNS algorithm was statistically equivalent to the GVNS algorithm, whereas the GVNS algorithm was much faster than the VIG_VNS algorithm due to the use of neighborhood partitioning improvement method in their algorithm. Regarding the fifth benchmark set with the makespan criterion, the VIG_VNS algorithm generated the new best-known solutions for 15 out of 31 problem instances compared to Cheng and Mao in [15] on the instances in Potvin and Bengio [39].

For the future work, the proposed VIG_VNS algorithm can be extended to other variants of TSP problem, such as a team-orienteering problem with and without time windows. The algorithm can be extended to capacitated vehicle-routing problems. In addition, several new meta-heuristics such as [41,42] can also be applied to the TSPTW.

## Appendix A. Forward pass example

*Step 1. Insert node $i = 1$ into position $j = 2$*

| | | $j$ | 0 | 1 | 2 | 3 | 4 | 5 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| | | $\pi$ | 0 | 3 | 1 | 4 | 5 | 2 | 0 |
| $i$ | $j$ | | $c(0,3)$ | $c(3,1)$ | $c(1,4)$ | $c(4,5)$ | $c(5,2)$ | $c(2,0)$ | |
| 1 | 2 | | $Remove = c(\pi_0,\pi_1) + c(\pi_1,\pi_2) - c(\pi_0,\pi_2)$<br>$Remove = c(0,3) + c(3,1) - c(0,1)$<br>$Add = c(\pi_3,\pi_1) + c(\pi_1,\pi_2) - c(\pi_2,\pi_3)$<br>$Add = c(4,3) + c(3,1) - c(1,4)$ | | | | | | |
| After insertion | | $\pi$ | 0 | 1 | 3 | 4 | 5 | 2 | 0 |
| | | Cost | $c(0,1)$ | $c(1,3)$ | $c(3,4)$ | $c(4,5)$ | $c(5,2)$ | $c(2,0)$ | |
| | | | $Gain = Add - Remove$<br>$Gain = c(4,3) + c(3,1) - c(1,4) - (c(0,3) + c(3,1) - c(0,1))$<br>$Gain = c(4,3) + c(3,1) - c(1,4) - c(0,3) - c(3,1) + c(0,1)$<br>$Gain = (c(0,1) + c(3,1) + c(4,3)) - (c(0,3) + c(3,1) + c(1,4)) < 0$ | | | | | | |
| | | *is feasible* | $e_{\pi_j} + c(\pi_j,\pi_i) \leqslant l_{\pi_i}$ *and* $e_{\pi_i} + c(\pi_i,\pi_{j+1}) \leqslant l_{\pi_{j+1}}$<br>$e_{\pi_2} + c(\pi_2,\pi_1) \leqslant l_{\pi_1}$ *and* $e_{\pi_1} + c(\pi_1,\pi_3) \leqslant l_{\pi_3}$<br>$e_1 + c(3,1) \leqslant l_3$ *and* $e_3 + c(3,4) \leqslant l_4$ | | | | | | |

*Step 2. Insert node $i = 1$ into position $j = 3$*

| | | $j$ | 0 | 1 | 2 | 3 | 4 | 5 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| | | $\pi$ | 0 | 3 | 1 | 4 | 5 | 2 | 0 |
| $i$ | $j$ | | $c(0,3)$ | $c(3,1)$ | $c(1,4)$ | $c(4,5)$ | $c(5,2)$ | $c(2,0)$ | |
| 1 | 3 | | $Remove = c(\pi_0,\pi_1) + c(\pi_1,\pi_2) - c(\pi_0,\pi_2)$<br>$Remove = c(0,3) + c(3,1) - c(0,1)$<br>$Add = c(\pi_4,\pi_1) + c(\pi_1,\pi_3) - c(\pi_3,\pi_4)$<br>$Add = c(5,3) + c(3,4) - c(4,5)$ | | | | | | |
| After insertion | | $\pi$ | 0 | 1 | 4 | 3 | 5 | 2 | 0 |
| | | Distance | $c(0,1)$ | $c(1,4)$ | $c(4,3)$ | $c(3,5)$ | $c(5,2)$ | $c(2,0)$ | |
| | | | $Gain = Add - Remove$<br>$Gain = c(5,3) + c(3,4) - c(4,5) - (c(0,3) + c(3,1) - c(0,1))$<br>$Gain = c(5,3) + c(3,4) - c(4,5) - c(0,3) - c(3,1) + c(0,1)$<br>$Gain = (c(5,3) + c(3,4) + c(0,1)) - (c(4,5) + c(0,3) + c(3,1)) < 0$ | | | | | | |
| | | *is feasible* | $e_{\pi_j} + c(\pi_j,\pi_i) \leqslant l_{\pi_i}$ *and* $e_{\pi_i} + c(\pi_i,\pi_{j+1}) \leqslant l_{\pi_{j+1}}$<br>$e_{\pi_3} + c(\pi_3,\pi_1) \leqslant l_{\pi_1}$ *and* $e_{\pi_1} + c(\pi_1,\pi_4) \leqslant l_{\pi_4}$<br>$e_4 + c(4,3) \leqslant l_3$ *and* $e_3 + c(3,5) \leqslant l_5$ | | | | | | |

## References


[1] M.W.P. Savelsbergh, Local search in routing problems with time windows, Ann. Oper. Res. 4 (1) (1985) 285–305.
[2] N. Christofides, A. Mingozzi, P. Toth, State-space relaxation procedures for the computation of bounds to routing problems, Networks 11 (2) (1981) 145–164.
[3] E.K. Baker, An exact algorithm for the time-constrained traveling salesman problem, Oper. Res. 31 (5) (1983) 938–945.
[4] A. Langevin, M. Desrochers, J. Desrosiers, S. Gelinas, F. Soumis, A two-commodity flow formulation for the traveling salesman and makespan problems with time windows, Networks 23 (7) (1993) 631–640.
[5] Y. Dumas, J. Desrosiers, E. Gelinas, M.M. Solomon, An optimal algorithm for the traveling salesman problem with time windows, Oper. Res. 43 (2) (1995) 367–371.
[6] N. Ascheuer, M. Fischetti, M. Grotschel, Solving asymmetric travelling salesman problem with time windows by branch-and-cut, Math. Program. 90 (2001) 475–506.
[7] E. Balas, N. Simonetti, Linear time dynamic-programming algorithms for new classes of restricted TSPs: a computational study, INFORMS J. Comput. 13 (1) (2001) 56–75.
[8] G. Pesant, M. Gendreau, J.-Y. Potvin, J.-M. Rousseau, An exact constraint logic programming algorithm for the traveling salesman problem with time windows, Transp. Sci. 32 (1998) 12–29.
[9] F. Focacci, A. Lodi, M. Milano, A hybrid exact algorithm for the TSPTW, INFORMS J. Comput. 14 (2002) 403–417.
[10] W.B. Carlton, J.W. Barnes, Solving the traveling-salesman problem with time windows using tabu search, IIE Trans. 28 (1996) 617–629.
[11] M. Gendreau, A. Hertz, G. Laporte, M. Stan, A generalized insertion heuristic for the traveling salesman problem with time windows, Oper. Res. 46 (1998) 330–335.
[12] M. Gendreau, A. Hertz, G. Laporte, New insertion and post optimization procedures for the traveling salesman problem, Oper. Res. 40 (1992) 1086–1094.
[13] R.W. Calvo, A new heuristic for the traveling salesman problem with time windows, Transp. Sci. 34 (1) (2000) 113–124.
[14] J.W. Ohlmann, B.W. Thomas, A compressed-annealing heuristic for the traveling salesman problem with time windows, INFORMS J. Comput. 19 (1) (2007) 80–90.
[15] C.-B. Cheng, C.-P. Mao, A modified ant colony system for solving the travelling salesman problem with time windows, Math. Comput. Model. 46 (2007) 1225–1235.
[16] M.L. Ibanez, C. Blum, Beam-ACO for the travelling salesman problem with time windows, Comput. Oper. Res. 37 (2010) 1570–1583.
[17] R.F. Da Silva, S. Urrutia, A general VNS heuristic for the traveling salesman problem with time windows, Discr. Optim. 7 (2010) 203–211.
[18] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, Eur. J. Oper. Res. 177 (3) (2007) 2033–2049.
[19] J.M. Framinan, R. Leisten, Total tardiness minimization in permutation flow shops: a simple approach based on a variable greedy algorithm, Int. J. Prod. Res. 46 (22) (2008) 6479–6498.
[20] G. Gerardo Minella, R. Ruiz, M. Ciavotta, Restarted iterated Pareto greedy algorithm for multi-objective flowshop scheduling problems, Comput. Oper. Res. 38 (2011) 1521–1533.
[21] M. Fatih Tasgetiren, Q.-K. Pan, P.N. Suganthan, H.-L. Chen Angela, A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops, Inf. Sci. 181 (2011) 3459–3475.
[22] Q. Qinma Kanga, H. Heb, H. Song, Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm, J. Syst. Softw. 84 (2011) 985–992.
[23] I. Ribas, R. Ramon Companys, X. Tort-Martorell, An iterated greedy algorithm for the flowshop scheduling problem with blocking, Omega 39 (2011) 293–301.
[24] M. Fatih Tasgetiren, Q.-K. Pan, Y.-C. Liang, A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times, Comput. Oper. Res. 36 (2009) 1900–1915.
[25] Q.-K. Pan, M. Fatih Tasgetiren, Y.-C. Liang, A discrete differential evolution algorithm for the permutation flowshop scheduling problem, Comput. Ind. Eng. 55 (2008) 795–816.
[26] R. Ruiz, T. Stützle, An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives, Eur. J. Oper. Res. 187 (2008) 1143–1159.
[27] L. Fanjul-Peyro, R. Ruiz, Iterated greedy local search methods for unrelated parallel machine scheduling, Eur. J. Oper. Res. 207 (2010) 55–69.
[28] M. Lozano, D. Molina, C. Garcia-Martinez, Iterated greedy for the maximum diversity problem, Eur. J. Oper. Res. 214 (2011) 31–38.
[29] K.-C. Kuo-Ching Ying, S.-W. Lin, C.-Y. Huang, Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic, Expert Syst. Appl. 36 (2009) 7087–7092.
[30] K.-C. Ying, H.-M. Cheng, Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic, Expert Syst. Appl. 37 (2010) 2848–2852.
[31] C. Kahraman, O. Engin, I. Kaya, R.E. Ozturk, Multiprocessor task scheduling in multistage hybrid flow-shops: a parallel greedy algorithm approach, Appl. Soft Comput. 10 (2010) 1293–1300.
[32] S. Bouamama, C. Blum, A. Boukerram, A population-based iterated greedy algorithm for the minimum weight vertex cover problem, Appl. Soft Comput. J. (2012), http://dx.doi.org/10.1016/j.asoc.2012.02.013.
[33] N. Mladenovic, P. Hansen, Variable neighborhood search, Comput. Oper. Res. 24 (1997) 1097–1100.
[34] M. Nawaz, E.E. Enscore Jr., I.A. Ham, Heuristic algorithm for the m-machine, n-node flow shop sequencing problem, Omega 11 (1) (1983) 91–95.
[35] Coello Carlos A. Coello, Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art, Comput. Methods Appl. Mech. Eng. 191 (11–12) (2002) 1245–1287.
[36] K. Deb, An efficient constraint handling method for genetic algorithms, Comput. Methods Appl. Mech. Eng. 186 (2000) 311–338.
[37] D.W. Coit, A.E. Smith, Penalty guided genetic search for reliability design optimization, Comput. Ind. Eng. 30 (4) (1996) 895–904.
[38] Mitsuo Gen, R. Cheng, Genetic Algorithms and Engineering Optimization, John Wiley and Sons, 2000.
[39] J.-Y. Potvin, S. Bengio, The vehicle routing problem with time windows Part II: genetic search, INFORMS J. Comput. 8 (1996) 165–172.
[40] M.M. Solomon, Algorithms for the vehicle routing and scheduling problems with time windows constraints, Oper. Res. 35 (2) (1987) 254–265.
[41] Cai Xingjuan, Fan Shujing, Tan Ying, Light responsive curve selection for photosynthesis operator of APOA, Int. J. Bio-Inspired Comput. 4 (6) (2012) 373–379.
[42] Xie Liping, Zeng Jianchao, Richard A. Formato, Selection strategies for gravitational constant G in artificial physics optimisation based on analysis of convergence properties, Int. J. Bio-Inspired Comput. 4 (6) (2012) 380–391.