Journal of Combinatorial Optimization (2021) 41:794–816
https://doi.org/10.1007/s10878-021-00716-2

A metaheuristic for the delivery man problem with time windows

Ha-Bang Ban[1]

Accepted: 19 February 2021 / Published online: 12 March 2021

Abstract
The Delivery Man Problem with Time Windows (DMPTW) is an extension of the Delivery Man Problem. The objective of DMPTW is to minimize the sum of customers' arrival time while the deliveries are made during a specific time window given by the customers. Another close variant of objective is a travel duration. In the case, the problem minimizes the sum of travel durations between a depot and customer locations. It has many practical applications to network problems, e.g., whenever servers have to accommodate a set of requests to minimize clients' total (or average) waiting time. To solve medium to large-sized instances, a two-phase metaheuristic algorithm is proposed. A construction stage generates a feasible solution using Neighborhood Descent with Random neighborhood ordering (RVND), and the optimization stage improves the feasible solution with an Iterated Local Search. Moreover, Tabu Search (TS) is incorporated in the proposed algorithm to prevent it from getting trapped into cycles. Therefore, our algorithm is prevented from becoming stuck at local optima. The results of experimental simulations are compared with well-known and successful metaheuristic algorithms. These results show that the proposed algorithm reaches better solutions in many cases.

Keywords DMPTW · ILS · TS · and RVND

1 Introduction

Delivery Man Problem with Time Windows (DMPTW) is a variant of Delivery Man Problem (DMP) that is seen as a "customer centric" routing problem. However, in DMP, there is no restriction on the time the service begins. That means the service begins as soon as the deliveryman reaches a location. In practice, this assumption does

✉ Ha-Bang Ban
BangBH@soict.hust.edu.vn

[1] School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam

not always hold. If the deliveryman reaches locations before customers are available for service, he will have to wait for the beginning of the service. Informally, in DMPTW, there are a vehicle at the main depot s, and n customers. Each customer has an additional restriction that is an interval time so that it has to be supplied. The goal is to find a tour that minimizes the overall customers' arrival time while ensuring that all customers are served. This objective has received recent attention in the literature Ban and Nghia (2017b). Another close variant of objective is a travel duration. In the case, the problem minimizes the sum of travel durations between a depot and customer locations. The travel duration of a customer is the difference between the beginning of his service and the beginning of service at the depot. DMPTW with the travel duration objective has been studied in Heilporna (2010).

DMPTW has many practical applications for scheduling tasks and logistics, etc. In logistics, it aims at finding a tour that minimizes the sum of arrival times of all customers so that each customer must be visited within a given time window. In scheduling tasks, DMPTW can model the problem of scheduling jobs on a single machine where setup times are sequence-dependent, and each job has a release and due time. Let the latency of each job be the time that it has to wait to be started. In the scheduling problem, the sum of latency of all jobs is subject to optimization.

To the best of our knowledge, there are three works for DMPTW in the literature. Tsitsiklis Tsitsiklis (1992) presented a polynomial algorithm since the number of locations are bounded. Heilporn et al. Heilporna (2010) then provided heuristic algorithms in accordance with insertion heuristic schemes or with an insertion heuristic scheme. After that, Ban et al. Ban and Nghia (2017b) proposed a metaheuristic algorithm based on Variable Neighborhood Search (VNS). Their experimental results Ban and Nghia (2017b); Heilporna (2010) were quite promising. However, these algorithms have drawbacks. The algorithm Ban and Nghia (2017b) might become trapped into cycles, and they return to the points previously explored in the solution space. Consequently, the algorithms can get stuck in local optima. The heuristic algorithms in Heilporna (2010), on the other hand, are often too greedy, and the explored solution space is not large enough. Therefore, they can get trapped into local optimum and fail to obtain the global optimum solution in many cases.

The success of any heuristic approach depends on balance between right intensification and diversification. While the above-mentioned algorithms Ban and Nghia (2017b); Heilporna (2010) meet some drawbacks, the proposed algorithm is developed to tackle them. The main contributions of this work can be summarized as follows:

– From the algorithmic design, we develop a two-phase metaheuristic consisting of a constructive and optimization stage. The construction stage seeks a feasible solution, whereas the improvement stage tries to improve it. Our metaheuristic combines RVND with ILS, in which ILS Talbi (2009) ensures diversification while RVND maintains intensification. This combination keeps the simplicity spirit of RVND while it effectively explores the search space. Moreover, TS Talbi (2009) is incorporated into our algorithm to mainly prohibit from getting trapped into cycles.

– From the computational perspective, extensive numerical experiments on benchmark instances show that our algorithm reaches better solutions than the-state-of-art metaheuristics in many cases.

The rest of this paper is organized as follows. Sections 2, 3 present the literature and problem definition, respectively. Section 4 describes the proposed algorithm. Computational evaluations are reported in Section 5. Sections 6, 7 discuss and conclude the paper, respectively.

2 Literature

DMPTW is NP-hard because it is a generalization case. Despite the similarities between DMP and DMPTW, solving DMPTW is more challenging since its solution must satisfy the time windows' constraint. M. W. Salvesbergh Salvesbergh (1985) argues that even building a feasible solution to the problem is a NP-hard problem. For NP-hard problems, there are three common approaches to solve the problem, namely, 1) exact algorithms, 2) approximation algorithms, 3) heuristic (or metaheuristic) algorithms. Firstly, the exact algorithms guarantee to find the optimal solution and take exponential time in the worst case, but they often run much faster in practice. In the literature, there is only work Heilporna (2010) to DMPTW. However, this exact algorithm only solves the problem with small sizes. Secondly, an α-approximation algorithm produces a solution within some factor of α of the optimal solution. Currently, no approximation work is known for DMPTW. Thirdly, heuristic algorithms perform well in practice and validate their empirical performance on an experimental benchmark of interesting instances. The metaheuristic algorithm falls into this approach. For DMPTW, there are two metaheuristic algorithms that are the algorithms of Ban et al. Ban and Nghia (2017b), and Heilporn Heilporna (2010).

Though only three works have been known for DMPTW, several classes of problems closely related to the problem were introduced in the literature: Delivery Man Problem (DMP), DMP with Profits (DMPP), Dynamic Traveling Repairman Problem (DTRP), Minimizing latency in Post-disaster Road Clearance Operations (ML-RCP), and Bi-objective Minimum Latency Problem with Profit Collection (BO-MLPPC).

– DMP is a particular case of DMPTW without time windows. Numerous works for DMP can be found in Ban and Nguyen (2010); Ban et al. (2013); Ban and Nghia (2017a); Blum et al. (1994); Chaudhuri et al. (2003); Goemans and Kleinberg (1996). Ban et al. (2013) presented an exact algorithm to solve instances with up to 40 vertices. Several approximation algorithms Blum et al. (1994); Chaudhuri et al. (2003); Goemans and Kleinberg (1996) were proposed to solve DMP with the best approximation ratio of 3.59. In the metaheuristic approach, we can found several algorithms Ban and Nguyen (2010); Ban and Nghia (2017a); Mladenovic et al. (2012); Salehipour et al. (2011); Silva et al. (2012); Tsitsiklis (1992). In Ban and Nguyen (2010), Ban et al. used a Genetic Algorithm to solve the problem while the others Ban and Nghia (2017a); Mladenovic et al. (2012); Salehipour

et al. (2011); Tsitsiklis (1992) were mainly based on Variable Neighborhood Search (VNS). Specifically, Salehipour et al. Salehipour et al. (2011) successfully combined Greedy Randomized Adaptive Search Procedure (GRASP) and VNS. Mladenovic et al. Mladenovic et al. (2012) then provided a General-VNS to improve solution quality. Later on, Silva et al. (2012) presented a multi-start approach, including a GRASP and RVNS. The algorithm finds good-quality solutions for large instances. Finally, Ban and Nghia (2017a) proposed a two-phase algorithm consisting of GRASP Feo and Resende (1995) in the construction phase and VNS+TS in the improvement phase. Their algorithm found the new best solutions in several small instances.

- In DMPP, the aim is to find a travel plan that maximizes the total revenue. However, in contrast to DMP, in DMPP, not all the customers need to be visited. In [14], Dewilde et al. combined VNS and Tabu Search to solve the problem. M. Avci then proposed a metaheuristic that brings together GRASP, and Iterated Local Search to improve solution quality. Their metaheuristic algorithms Avci and Avci (2017); Dewilde et al. (2013) can well solve the problem with up to 500 vertices. After that, Beraldi et al. Beraldi et al. (2018) have introduced a stochastic variant of DMPP under uncertain travel times. The difference between DMPP and stochastic DMPP is that the travel times in stochastic DMPP are uncertain. Stochastic DMPP is heuristically solved by means of a beam search heuristic. Finally, Bruni et al. extended stochastic DMPP with multiple vehicles Bruni et al. (2018, 2019). The effectiveness of their algorithms was very promising through extensive experimental results.
- In DTRP Bertsimas and Ryzin (1989), the objective function of TRP is stochastic in a dynamic environment. Currently, to the best of our knowledge, this is the only paper dealing with DTRP. However, neither metaheuristic nor exact algorithm is proposed for the problem.
- ML-RCP Ajama et al. (2019) finds a tour with minimum latency sum in post-disaster road clearance. Unlike TRP, in disaster situations, travel costs need to be added to debris removal times. Moreover, they divide vertices in the graph into two types: 1) critical vertices must be visited; 2) non-critical vertices may or may not be visited. They provided GRASP with VNS metaheuristic that obtained optimal or near-optimal solutions on Istanbul data within seconds.
- In BO-MLPPC Bruni (2020), routes must be constructed to maximize the collected profit and minimize the total latency. In this problem, not all the customers need to be visited and there is the presence of uncertain travel times. To address this problem, the authors proposed a Multi-Objective Iterated Local Search. The experimental results concluded that the algorithm found good-quality solutions for small and medium-size instances.

The above algorithms are the state-of-the-art algorithms for several variants of DPMTW. However, there is no time restriction in these problems, and they cannot be adapted directly to DMPTW. That means that we cannot use the above algorithms to solve DMPTW.

3 Problem definition

In this section, we provide a formulation of the DMPTW:
Given an complete graph $K_n = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is a set of vertices representing the depot and client places, and E is the set of edges connecting the vertices. Assume that, a tour is represented as $T = (v_1 = s, v_2 ..., v_n)$, and v_k denotes the index of the vertex at the k-th position of the tour. For each edge $(v_i, v_j) \in E$, which connects the two vertices v_i and v_j, there exists a cost $c(v_i, v_j)$. This cost describes the travel time between vertex v_i and v_j. A time window $[e_i, l_i]$ is associated to each vertex $v_i \in V$, which indicates when service time at vertex v_i can start. This implies that a vertex v_i may be reached before the start e_i, but service cannot start until e_i and no latter than l_i of its time window. In addition, the deliveryman spends time to serve each customer. Let $S(v_i)$ be the service time at vertex v_i. Therefore, given a particular tour T, let $D(v_k)$ be the time at which service begins at vertex v_k. It is calculated as follows: $D(v_k) = \max\{A(v_k), e_k\}$, where $A(v_k) = D(v_{k-1}) + S(v_{k-1}) + c(v_{k-1}, v_k)$ is the arrival time at vertex v_k in the tour. We also define the travel duration of vertex v_i as the difference between the beginning of service at v_i and the beginning of service at the depot. A tour is feasible, if and only if $A(v_k) \leq l_k$ for all vertices. DMPTW requires finding a feasible tour visiting all vertices to minimize the sum of arrival times or travel durations.

4 Two-Phase Algorithm

4.1 Neighborhoods

Several popular neighborhoods in the literature are used to explore the solution space of the problem [29]. Let *Tsol* be the time complexity of calculating solution's cost. The main operation in exploring these neighborhoods is the calculation of the cost function of neighboring solutions. In straightforward implementation in the worst case, this operation requires $Tsol = \Theta(n)$. Assume that T, and n are a tour, and its size, respectively. We describe in more details about nine neighborhoods as follows:

- **move-up** moves a vertex forward one position in T. The complexity of exploring the neighborhood is $O(Tsol \times n)$.
- **move-down** moves a vertex backward one position in T. The complexity of exploring the neighborhood is $O(Tsol \times n)$.
- **shift** relocates a vertex to another position in T. The complexity of exploring the neighborhood is $O(Tsol \times n)$.
- **swap-adjacent** attempts to swap each pair of adjacent vertices in T. The complexity of exploring the neighborhood is $O(Tsol \times n)$.
- **swap** tries to swap the positions of each pair of vertices in T. The complexity of exploring the neighborhood is $O(Tsol \times n^2)$.
- **2-opt** removes each pair of edges from T and reconnects them. The complexity of exploring the neighborhood is $O(Tsol \times n^2)$.

- **Or-opt**: Three adjacent vertices are reallocated to another position of T. The complexity of exploring the neighborhood is $O(Tsol \times n^2)$.
- **Move-forward-k-vertices** moves k consecutive vertex positions to the right of T ($k = 1, 2, \ldots, l$). The complexity of exploring the neighborhood is $O(Tsol \times l \times n^2)$.
- **Move-backward-k-vertices** moves k consecutive vertex positions to the left of T ($k = 1, 2, \ldots, l$). The complexity of exploring the neighborhood is $O(Tsol \times l \times n^2)$.

In DMPTW, when the positions of vertices in T are changed, the other vertices' arrival times may be affected later. Therefore, $Tsol$ requires exactly $\Theta(n)$ time. Unfortunately, it leads to $\Theta(n)$ operations for each move evaluation resulting in $\Theta(n^3)$ operations for a full neighborhood search.

4.2 RVND+ILS with Tabu

A good metaheuristic must ensure the balance between diversification and intensification. Mladenovic et al. Mladenovic and Hansen (1997) show that RVND allows a better understanding of the characteristics and behavior of the problem. They further claim that RVND yields local optima that are nearer to the global optimum in a more straightforward manner than other metaheuristics. However, RVND only implements intensification, and it needs to be added to diversification. Our two-phase metaheuristic includes a constructive and optimization stage. Our metaheuristic combines RVND with ILS, in which ILS Talbi (2009) ensures diversification while RVND maintains intensification. This combination keeps the simplicity spirit of RVND while it effectively explores the search space. Moreover, TS Talbi (2009) is incorporated into our algorithm to mainly prohibit from getting trapped into cycles. Algorithm 1 depicts the whole process.

The construction phase: Algorithm 2 shows the constructive procedure. The objective function used here is the sum of all positive differences between the time to reach each vertex and its due time, that is, min $\sum_{i=1}^{n} \max(0, A(v_i) - l_i)$. The algorithm works until it finds a feasible solution. From lines 2 to 6, Restricted Candidate List (*RCL*) is determined by ordering all non-selected vertices in terms of a greedy manner that measures the benefit of including them on tour. After that, one element will be chosen randomly from *RCL*. The size of *RCL* is a parameter that controls the balance between greediness and randomness. Since all vertices are visited, we obtain a solution. In lines 7 and 8, if the solution is feasible, it is considered an initial solution, and the construction stops. In contrast, a local search procedure based on the VNS Mladenovic and Hansen (1997) is started, and the algorithm iterates until finding a feasible solution or $level\_max$ is reached. The solution is perturbed, by making level random shift moves, to prevent it from becoming stuck at local optima in line 12. Next, the VNS is applied to the perturbed solution to create a new solution. If it is better than the best-found solution, it is set to the new current solution through lines 13 to 24. At the end of each iteration from lines 25 to 30, *level* is increased by one if the current solution is not improved, or reset to 1, otherwise.

Algorithm 1 RVND+ILS with Tabu

Input: $v_1, V, N_i(T)(i = 1, ..., 9), level$ are a starting vertex, the set of vertices in K_n, the set of neighborhoods, the parameter to control the strength of the perturbation procedure, respectively.
Output: the best solution T^*.

for $i \leftarrow 1, ..., I_{max}$ **do**
  {Step 1}
  {Construction phase}
  $T \leftarrow$ **Construction**(v_1, V);
  $T' \leftarrow T$;
  $iterILS \leftarrow 0$;
  {Improvement phase}
  while $iterILS < m$ **do**
    {Step 2: RVND}
    Initialize the Neighborhood List NL;
    while $NL \neq 0$ **do**
      Choose a neighborhood N in NL at random;
      $T'' \leftarrow \arg\min N(T)$;{$T''$ must be feasible} {Neighborhood search}
      if $(L(T'') < L(T)$ and T'' is not tabu) or $(L(T'') < L(T^*))$ **then**
        $T \leftarrow T''$
        Update NL;
        Update tabu lists;
        if $(L(T'') < L(T^*))$ **then**
          $T^* \leftarrow T''$; {Update the best solution}
        end if
      else
        Remove N_i from the NL; {Do not implement neighborhood N_i}
      end if
    end while
    $T \leftarrow T''$;
    if $L(T) < L(T')$ **then**
      {Step 3: Intensification to exploit the region}
      $\overline{T}$ = Perform RVND without using tabu list restrictions on the solution T; {$\overline{T}$ must be feasible}
      if $(L(T) < L(\overline{T}))$ or $(L(T^*) < L(\overline{T}))$ **then**
        $T \leftarrow \overline{T}$;
      end if
      if $(L(T^*) < L(\overline{T}))$ **then**
        $T^* \leftarrow \overline{T}$; {Update the best solution}
      end if
      $T' \leftarrow T$;
      $iterILS \leftarrow 0$;
    end if
    {Step 4: Diversification to explore new region}
    Clear all tabu lists;
    $T \longleftarrow$ Perturbation$(T, level)$; {implement perturbation technique}
    $T' \leftarrow T$;
    $iterILS++$;
  end while
end for
return T^*;

The improvement phase: After the construction of the feasible initial solution, this step tries to improve it. In this phase, the objective function is to minimize $\sum_{i=2}^{n}(D(v_k) - D(v_1))$. Our algorithm begins with the feasible solution provided

Algorithm 2 Construction

Input: $v_1, V, k, \alpha, level$ are a starting vertex, the set of vertices in K_n, the number of vehicles and the size of RCL, the parameter to control the strength of the perturbation procedure, respectively.
Output: An initial solution T.

```
1: T = φ; {T is a tour}
2: while |T| < n do
3:     Create RCL that includes α nearest vertices to v_e in V; {v_e is the last vertex in T}
4:     Select randomly vertex v = {v_i | v_i ∈ RCL and v_i ∉ T};
5:     T ← T ∪ v_i
6: end while
7: if T is a feasible solution then
8:     return T;
9: else
10:    level = 1;
11:    while ((T is infeasible solution) and (level ≤ level_max)) do
12:        T' = Perturbation(T, level);
13:        for i : 1 → 9 do
14:            T'' ← arg min N_i(T'); {local search}
15:            if (L(T'') < L(T')) then
16:                T' ← T''
17:                i ← 1
18:            else
19:                i++
20:            end if
21:        end for
22:        if L(T') < L(T) then
23:            T ← T';
24:        end if
25:        if L(T') == L(T) then
26:            level ← 1;
27:        else
28:            level++;
29:        end if
30:    end while
31: end if
32: return T;
```

by the first phase. In Step 1, in line 4, the algorithm starts with the initial solution obtained from the construction phase and consists of three main steps repeated until a stop condition is met. In Step 2, from lines 11 to 25, we introduce the structure of several neighborhoods in RVND. Moreover, to avoid tabu moves, two tabu lists are used. Step 3, from lines 26 to 39, aims at exploiting the current solution space. To explore unvisited solution space, a diversification phase is added in Step 4 from lines 41 to 42. In the remaining of this section, more details about the four steps of our algorithm are described. Some successful applications of ILS on other combinatorial optimization problems can be found in Ibaraki et al. (2008).

In the local search procedure, the Neighborhood List (*NL*) is initialized with all neighborhoods. A given neighborhood is selected at random from the *NL*. Neighbor solutions are evaluated, and the best feasible neighboring solution is accepted if it is non-tabu, improving, or tabu but globally improving. In the case of improvement, *NL* is repopulated with all neighborhoods. Otherwise, we removed it from *NL*. The procedure

Algorithm 3 Perturbation(T, $level$)

Input: T, k are the tour, and the number of swap, respectively.
Output: a new tour T.
1: **while** ($level > 0$) **do**
2: $k_1 = 1 + \text{rand}(\frac{n}{4})$;
3: $k_2 = k_1 + 1 + \text{rand}(\frac{n}{4})$;
4: $k_3 = k_2 + 1 + \text{rand}(\frac{n}{4})$;
5: $\{T_1$ copies consecutive vertices from 1-st to $k_1 - th$ position in $T\}$
6: $T_1 = T[1 : k_1]$;
7: $\{T_2$ copies consecutive vertices from $k_3 - th$ to $k_4 - th$ position in $T\}$
8: $T_2 = T[k_3 : k_4]$;
9: $\{T_3$ copies consecutive vertices from $k_2 - th$ to $k_3 - th$ position in $T\}$
10: $T_3 = T[k_2 : k_3]$;
11: $\{T_4$ copies consecutive vertices from $k_1 - th$ to $k_2 - th$ position in $T\}$
12: $T_4 = T[k_1 : k_2]$;
13: $T = T_1 \cup T_2 \cup T_3 \cup T_4$;
14: $level = level - 1$;
15: **end while**
16: **return** T;

terminates when *NL* becomes empty. Due to different neighborhood structures, two tabu lists are built. A move of the type remove-insert, swap-adjacent, move-up (down), and shift is stored in the first tabu list while the second is for 2-opt, and or moves. After each move, only the tabu list of the corresponding neighborhood is updated. The reverse move becomes tabu after some iterations that are determined dynamically. For tabu lists, a tabu tenure is a random number between 5 and 10. Specifically, the tabu lists in this work are implemented as follows:

- Move-up (down) moved up (down) v_i forward (backward) one position cannot be applied again while it is tabu.
- Shift applied to v_i to another position of the tour cannot be applied again while it is tabu.
- Swap, and swap-adjacent swapped a pair of (v_i, v_j) cannot swapped again while it is tabu.
- 2-opt implemented to a pair of (v_i, v_j) cannot implement again to the same vertices.
- Or-opt applied to (v_i, v_j) and v_k is prohibited from applying the same vertices.

When a better solution is found, the intensification step starts. The solution is performed with RVND without any tabu move. Since the new best-known solution is found, $iterILS$ is set to 0. After that, the algorithm goes to the perturbation step. The Perturbation mechanism design is very important to achieve success in our algorithm. If the mechanism produces too small perturbation moves, the search procedure may return to the previously visited local optimum points. On the other hand, excessive perturbation moves may drive the search procedure to undesirable regions in the search space. To overcome these issues, we implement an adaptive perturbation mechanism. The perturbation mechanism, called double-bridge, is originally developed in Martin et al. (1991). It consists of removing and re-inserting four arcs in the tour. This mechanism can also be seen as a permutation of two disjoint segments of the tour. The detail of the step is described in Algorithm 3.

The last aspect to discuss is the stop criterium of our algorithm. A balance must be made between computation time and efficiency. Here, the algorithm stops if no improvement is found after m loops.

5 Evaluations

Our algorithm is run on a Pentium 4 core i7 2.40 GHz processor with 8 GB of RAM. On all experiments, parameters α, $level\_max$, m, and I_{max} are respectively set to 10, 5, 50, and 10. These parameters are chosen through empirical tests, and with them, the algorithm seems to produce good solutions at a reasonable amount of time compared to the other parameter values.

We also tested the performance of the proposed algorithm against the state-of-the-art algorithms Ban and Nghia (2017b); Heilporna (2010); Silva and Urrutia (2010); Ohlmann and Thomas (2007).

5.1 Instances

We implement the algorithm in four sets of instances that include more than 160 instances as followings:

- The first set is proposed by Dumas et al. Dumas et al. (1995) 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. For example, the instances n20w60.001, n20w60.002, n20w60.003, n20w60.004, and n20w60.005 have 20 vertices and the time window for each vertex is uniformly random, between 0 and 60.
- The second set of instances is proposed by Gendreau et al. (1998) and contains 140 instances grouped in 28 test cases.
- The third set of instances is proposed by Ohlmann and Thomas (2007) and contains 25 instances grouped in 5 test cases. The second and third sets, in the majority, are the instances proposed by Dumas et al. (1995) with wider time windows.
- The fourth is proposed by Silva et al. [36] to overcome the size limitations of the others since the size of instances is more than 200 customers and has wider time windows.

5.2 Results

We define the improvement of the proposed algorithm with respect to *Best.Sol* (*Best.Sol* is the best solution found by our algorithm) in comparison with the-state-of-the-art metaheuristic algorithms for the problem as followings:

$$Gap[\%] = |\frac{Best.Sol - BKS}{BKS}| \times 100\% \quad (1)$$

$$Improv[\%] = |\frac{Best.Sol - Init.Sol}{Init.Sol}| \times 100\% \quad (2)$$

Table 1 Comparison between results obtained by Ban et al. Ban and Nghia (2017b) and by our algorithm on instances proposed by Dumas et al. Bruni (2020), and Silva et al. Heilporna (2010)

Instances	Ban et al.		Our algorithm					
	Best.Sol	T	Init.Sol	Best.Sol	Aver.Sol	Improv	Gap	T
n20w20.001	2834	0.52	**2536**	**2528**	2528	0.32	10.8	0.49
n20w20.002	2703	0.57	**2560**	**2560**	2560	0.00	5.29	0.54
n20w20.003	2671	0.58	**2679**	**2671**	2671	0.30	0.00	0.50
n20w20.004	2989	0.60	**2989**	**2975**	2975	0.47	0.47	0.52
n20w40.001	2430	0.63	**2406**	**2270**	2270	5.65	6.58	0.49
n20w40.002	2825	0.51	**2679**	**2679**	2679	0.00	5.17	0.53
n20w40.003	2889	0.55	**2889**	**2774**	2774	3.98	3.98	0.50
n20w40.004	2602	0.54	**2602**	**2568**	2568	1.31	1.31	0.53
n20w60.001	2666	0.55	**2666**	**2421**	2421	9.19	9.19	0.54
n20w60.002	2694	0.57	**2261**	**2176**	2176	3.76	19.23	0.54
n20w60.003	2694	0.57	**2934**	**2694**	2694	8.18	0.00	0.52
n20w60.004	2205	0.52	**2205**	**2020**	2020	8.39	8.39	0.49
n20w80.001	3244	0.54	**3244**	**2990**	2990	7.83	7.83	0.50
n20w80.002	2790	0.56	**2790**	**2669**	2669	4.34	4.34	0.55
n20w80.003	2894	0.53	**2894**	**2643**	2643	8.67	8.67	0.49
n20w80.004	2685	0.60	**2685**	**2627**	2627	2.16	2.16	0.55
n20w100.001	2618	0.52	**2618**	**2294**	2294	12.38	12.38	0.52
n20w100.002	2476	0.53	**2476**	**2082**	2082	15.91	15.91	0.56
n20w100.003	2786	0.52	**2786**	**2416**	2416	13.28	13.28	0.49
n20w100.004	3319	0.52	**3319**	**2914**	2914	12.20	12.2	0.52
n40w20.001	7994	2.53	**7956**	**7875**	7875	1.02	1.49	2.55
n40w20.002	7640	1.63	**7640**	**7527**	7527	1.48	1.48	2.76

Table 1 continued

Instances	Ban et al.		Our algorithm					
	Best.Sol	T	Init.Sol	Best.Sol	Aver.Sol	Improv	Gap	T
n40w20.003	7654	2.92	**7654**	**7535**	7535	1.55	1.55	2.77
n40w20.004	7032	3.01	**7062**	**7031**	7031	0.44	0.01	2.93
n40w40.001	7827	2.41	**7827**	**7663**	7663	2.10	2.10	2.74
n40w40.002	7349	2.65	**7349**	**7104**	7104	3.33	3.33	2.70
n40w40.003	7556	3.02	**7689**	**7483**	7483	2.68	0.97	2.84
n40w40.004	6980	2.89	**7146**	**6917**	6917	3.20	0.90	2.87
n40w60.001	7247	2.81	**7557**	**7066**	7066	6.50	2.50	2.76
n40w60.002	6758	3.42	7445	7247	7247	2.66	-7.24	2.67
n40w60.003	7040	2.93	**7040**	**6758**	6758	4.01	4.01	2.58
n40w60.004	6070	2.68	**6070**	**5548**	5548	8.60	8.60	2.79
n40w80.001	8600	3.40	**8600**	**8229**	8229	4.31	4.31	2.63
n40w80.002	7500	3.47	**7500**	**7176**	7176	4.32	4.32	2.52
n40w80.003	7721	2.93	**7721**	**7075**	7075	8.37	8.37	2.88
n40w80.004	7408	2.61	**7408**	**7166**	7166	3.27	3.27	2.62
n40w100.001	7747	2.75	**7747**	**6858**	6858	11.48	11.48	2.72
n40w100.002	8075	2.90	**8075**	**6778**	6778	16.06	16.06	2.84
n40w100.003	6924	3.09	**6924**	**6178**	6178	10.77	10.77	2.68
n40w100.004	7339	2.76	**7339**	**7019**	7019	4.36	4.36	2.87
n60w20.002	13879	10.70	14080	13996	14041.63	0.60	-0.84	14.00
n60w20.003	13839	11.07	**13839**	**13782**	13808.25	0.41	0.41	14.80
n60w20.004	13006	10.80	**13006**	**12965**	12980.40	0.32	0.32	14.15
n60w40.003	15151	12.54	**15151**	**15034**	15081.67	0.77	0.77	14.69
n60w40.004	12635	11.89	**12635**	**12584**	12605.00	0.40	0.40	14.48

Table 1 continued

Instances	Ban et al.		Our algorithm					
	Best.Sol	T	Init.Sol	Best.Sol	Aver.Sol	Improv	Gap	T
n60w60.001	13654	12.31	**13654**	**13459**	13534.80	1.43	1.43	14.17
n60w60.002	15220	12.85	**15220**	**14506**	14846.08	4.69	4.69	13.49
n60w60.003	12940	12.46	**12940**	**12523**	12738.17	3.22	3.22	14.33
n60w60.004	14243	11.97	**14243**	**13923**	14060.56	2.25	2.25	13.17
n60w80.001	14223	12.15	**14223**	**13572**	13850.79	4.58	4.58	14.25
n60w80.002	14593	11.47	**14593**	**13498**	13990.34	7.50	7.50	14.32
n60w80.003	14511	11.91	**14511**	**13787**	14096.03	4.99	4.99	14.46
n60w80.004	13350	12.92	**13350**	**12966**	13141.90	2.88	2.88	14.78
n60w100.001	12153	12.09	**12153**	**11882**	11982.27	2.23	2.23	14.96
n60w100.002	14374	12.04	**14374**	**13699**	14066.78	4.70	4.70	14.54
n60w100.003	12437	11.46	**12437**	**12086**	12260.93	2.82	2.82	14.16
n60w100.004	13971	11.97	**13971**	**13289**	13613.33	4.88	4.88	14.86
n80w20.001	19666	23.62	**19666**	**19565**	19616.09	0.51	0.51	24.54
n80w20.002	19754	22.35	20712	20680	20696.50	0.15	-4.69	27.56
n80w20.003	20578	24.23	**20578**	**20531**	20556.67	0.23	0.23	26.78
n80w20.004	20364	22.56	**20364**	**20192**	20267.08	0.84	0.84	26.48
n80w40.001	21031	21.30	**21031**	**20713**	20861.43	1.51	1.51	26.81
n80w40.003	19926	22.12	**19926**	**19860**	19895.36	0.33	0.33	26.26
n80w40.004	21459	22.12	**21459**	**21179**	21288.17	1.30	1.30	27.88
n80w60.001	19877	23.40	**19877**	**19429**	19623.11	2.25	2.25	27.91
n80w60.002	19728	22.30	**19728**	**19332**	19542.45	2.01	2.01	27.15

Table 1 continued

Instances	Ban et al.		Our algorithm					
	Best.Sol	T	Init.Sol	Best.Sol	Aver.Sol	Improv	Gap	T
n80w60.003	19598	24.50	**19598**	**19315**	19425.41	1.44	1.44	26.12
n80w60.004	20543	22.70	**20543**	**20317**	20414.38	1.10	1.10	26.47
n100w20.001	28394	40.62	30748	30604	30676.53	0.47	-7.78	48.28
n100w20.003	30124	40.12	**30138**	**30031**	30083.54	0.36	0.31	45.17
n100w20.004	28397	40.12	**28397**	**28316**	28359.00	0.29	0.29	45.68
n100w40.001	26156	41.03	**26156**	**25914**	25992.51	0.93	0.93	47.60
n100w40.002	28846	46.70	**29330**	**28819**	29070.29	1.74	0.09	47.93
n100w40.003	28262	44.12	**28315**	**28008**	28121.64	1.08	0.90	47.59
n100w40.004	27968	45.21	**27980**	**27489**	27730.64	1.75	1.71	46.80
n100w60.001	27909	45.60	**27909**	**27308**	27567.99	2.15	2.15	47.19
n100w60.002	27763	46.70	**27763**	**27630**	27696.79	0.48	0.48	46.19
n100w60.003	27750	47.80	**27750**	**27418**	27566.71	1.20	1.20	47.98
n100w60.004	26580	48.90	**26580**	**26154**	26302.60	1.60	1.60	45.76
n150w20.001	-	-	**54408**	**54150**	5427.6	0.47	-	124.12
n150w20.002	-	-	**53658**	**53396**	5352.9	0.49	-	121.10
n150w20.003	-	-	**48087**	**47715**	4788.6	0.77	-	122.21
n150w20.004	-	-	**49745**	**49534**	4963.1	0.42	-	123.75
n150w40.001	-	-	**48072**	**47695**	4787	0.78	-	124.68
n150w40.002	-	-	**53007**	**52804**	5287.6	0.38	-	120.49
n150w40.003	-	-	**50583**	**49919**	5023.4	1.31	-	125.58
n150w40.004	-	-	**48643**	**48230**	4841.8	0.85	-	124.65

Table 1 continued

Instances	Ban et al.		Our algorithm					
	Best.Sol	T	Init.Sol	Best.Sol	Aver.Sol	Improv	Gap	T
n200w20.001	-	-	**78821**	**78574**	7870.2	0.31	-	171.31
n200w20.002	-	-	**85516**	**85262**	8537.7	0.30	-	181.90
n200w20.003	-	-	**80733**	**80473**	8059.6	0.32	-	180.77
n200w20.004	-	-	**76449**	**76178**	7629	0.35	-	174.90
n250w100.001	-	-	**1656954**	**1656716**	1656839.36	0.01	-	230.00
n250w100.002	-	-	**1727494**	**1727085**	1727248.00	0.02	-	234.33
n250w200.001	-	-	**1607834**	**1606062**	1606971.26	0.11	-	243.34
n250w200.002	-	-	**1666983**	**1664917**	1665899.22	0.12	-	243.65
n250w300.001	-	-	**1732981**	**1729016**	1730955.16	0.23	-	245.65
n250w300.002	-	-	**1594207**	**1587930**	1590723.30	0.39	-	231.34
Aver						3.08	3.62	45.72

Improved results are highlighted in boldface

Table 2 Comparison between results obtained by Ban et al. Ban and Nghia (2017b) and by our algorithm on instances proposed by Gendreau et al. Dumas et al. (1995), and Ohlmann et al. Mladenovic et al. (2012)

Instances	Ban et al.		Our algorithm					
	Best.Sol	T	Init.Sol	Best.Sol	Aver.Sol	Improv	Gap	T
n20w120.002	2252	0.59	**2243**	**2193**	2193	2.23	2.62	0.51
n20w120.003	2421	0.58	**2366**	**2337**	2337	1.23	3.47	0.48
n20w120.004	2853	0.54	**2826**	**2686**	2686	4.95	5.85	0.43
n20w140.002	2610	0.57	**2433**	**2330**	2330	4.23	10.73	0.50
n20w140.003	2283	0.60	**2238**	**2194**	2194	1.97	3.90	0.48
n20w140.004	2414	0.58	**2358**	**2279**	2279	3.35	5.59	0.45
n20w160.002	2020	0.56	**1987**	**1830**	1830	7.90	9.41	0.49
n20w160.003	2426	0.56	**2408**	**2286**	2286	5.07	5.77	0.45
n20w160.004	1832	0.59	**1725**	**1616**	1616	6.32	11.79	0.50
n20w180.002	2483	0.54	**2409**	**2315**	2315	3.90	6.77	0.52
n20w180.003	2580	0.54	**2513**	**2414**	2414	3.94	6.43	0.46
n20w180.004	2831	0.56	**2777**	**2624**	2624	5.51	7.31	0.51
n20w200.002	1969	0.59	**1931**	**1799**	1799	6.84	8.63	0.46
n20w200.003	2222	0.53	**2184**	**2144**	2144	1.83	3.51	0.44
n20w200.004	2831	0.56	**2777**	**2624**	2624	5.51	7.31	0.49
n40w120.002	6451	2.51	**6365**	**6265**	6265	1.57	2.88	2.84
n40w120.003	6525	2.99	**6520**	**6411**	6411	1.67	1.75	2.85
n40w120.004	5970	2.58	**5940**	**5855**	5855	1.43	1.93	2.72
n40w140.002	5863	2.69	**5833**	**5746**	5746	1.49	2.00	2.67
n40w140.003	6789	2.60	**6633**	**6572**	6572	0.92	3.2	2.71
n40w140.004	5964	2.74	**5891**	**5719**	5719	2.92	4.11	2.64
n40w160.002	6443	2.98	**6424**	**6368**	6368	0.87	1.16	2.91
n40w160.003	5975	2.96	**5943**	**5850**	5850	1.56	2.09	2.72

Table 2 continued

Instances	Ban et al.		Our algorithm					
	Best.Sol	T	Init.Sol	Best.Sol	Aver.Sol	Improv	Gap	T
n40w160.004	4611	2.53	**4571**	**4468**	4468	2.25	3.10	2.94
n40w180.002	6203	2.63	**6182**	**6104**	6104	1.26	1.60	2.88
n40w180.003	6073	2.71	**6057**	**6040**	6040	0.28	0.54	2.70
n40w180.004	6250	2.77	**6209**	**6103**	6103	1.71	2.35	2.90
n40w200.002	8158	2.71	**7347**	**6674**	6674	9.16	18.19	2.69
n40w200.003	5792	2.99	**5772**	**5542**	5542	3.98	4.32	2.61
n40w200.004	6250	2.65	**6209**	**6103**	6103	1.71	2.35	2.90
n60w120.002	12636	10.26	**12603**	**12517**	13097.23	0.68	0.94	13.97
n60w120.003	11758	12.00	**11741**	**11690**	12157.10	0.43	0.58	14.69
n60w120.004	11206	10.34	**11180**	**11132**	11721.03	0.43	0.66	13.42
n60w140.002	12064	11.12	**11895**	**11782**	12269.62	0.95	2.34	14.26
n60w140.003	13285	11.76	**13254**	**13128**	13897.28	0.95	1.18	13.06
n60w140.004	13688	11.34	**13392**	**13189**	13366.80	1.52	3.65	14.23
n60w160.002	12835	10.74	**12686**	**12471**	13790.48	1.69	2.84	13.10
n60w160.003	10741	10.92	**10708**	**10682**	11684.83	0.24	0.55	13.98
n60w160.004	11820	11.96	**11795**	**11645**	12544.80	1.27	1.48	13.39
n60w180.002	12144	11.71	**12135**	**12015**	12629.89	0.99	1.06	13.41
n60w180.003	12389	11.29	**12376**	**12214**	13018.66	1.31	1.41	13.29
n60w180.004	11206	10.75	**11184**	**11101**	11996.46	0.74	0.94	13.38
n60w200.002	11974	10.86	**11859**	**11748**	12672.98	0.94	1.89	14.27
n60w200.003	10982	10.96	**10853**	**10697**	12237.75	1.44	2.60	13.56
n60w200.004	11534	10.24	**11510**	**11441**	12243.66	0.60	0.81	14.08
n80w120.002	18268	22.85	**18200**	**18181**	18536.79	0.10	0.48	25.30

Table 2 continued

Instances	Ban et al.		Our algorithm					
	Best.Sol	T	Init.Sol	Best.Sol	Aver.Sol	Improv	Gap	T
n80w120.003	18163	21.80	**18073**	**17878**	18774.41	1.08	1.57	25.67
n80w120.004	17390	23.47	**17364**	**17318**	18499.71	0.26	0.41	25.92
n80w140.002	18006	23.19	**17932**	**17815**	19278.19	0.65	1.06	24.93
n80w140.003	17450	22.03	**17359**	**17315**	17981.11	0.25	0.77	23.57
n80w140.004	19046	22.75	**19029**	**18936**	19902.45	0.49	0.58	21.89
n80w160.002	17168	23.72	**17148**	**17091**	18281.90	0.33	0.45	21.67
n80w160.003	16936	23.64	**16809**	**16606**	18398.57	1.21	1.95	21.15
n80w160.004	17985	23.45	**17890**	**17804**	19203.29	0.48	1.01	25.70
n80w180.002	17511	22.78	**17495**	**17339**	18272.36	0.89	0.98	22.48
n80w180.003	17370	21.07	**17343**	**17271**	17597.40	0.42	0.57	22.66
n80w180.004	16804	22.28	**16784**	**16729**	17594.16	0.33	0.45	23.34
n100w120.002	29953	41.17	**29926**	**29882**	31112.66	0.15	0.24	46.68
n100w120.003	25659	41.34	**25490**	**25275**	26665.72	0.84	1.50	45.88
n100w120.004	30273	42.60	**30231**	**30102**	31010.51	0.43	0.56	46.04
n100w140.002	30363	44.27	**30271**	**30192**	31575.11	0.26	0.56	48.38
n100w140.003	28480	43.04	**28375**	**28309**	29872.99	0.23	0.60	47.34
n100w140.004	27592	45.10	**27571**	**27448**	28081.39	0.45	0.52	49.56
Aver						1.92	2.98	13.06

Improved results are highlighted in boldface

Table 3 Comparison between results obtained by Heilporn et al. Fischetti et al. (1993) and by our algorithm on instances proposed by Gendreau et al. Dumas et al. (1995), and Ohlmann et al. Mladenovic et al. (2012)

Instances	G. Heilporn et al.		Our algorithm		
	Best.Sol	T	Best.Sol	T	cTime
n20w120.001	2535	1	**2175**	0.46	1.24
n20w140.001	1908	1	**1846**	0.45	1.22
n20w160.001	2150	1	**2146**	0.46	1.24
n20w180.001	2037	1	**2477**	0.45	1.22
n20w200.001	2294	1	**1975**	0.45	1.22
n40w120.001	7496	5	**6800**	2.25	6.09
n40w140.001	7203	4	**6290**	2.7	7.31
n40w160.001	6657	4	**6143**	2.51	6.79
n40w180.001	6578	5	**6952**	2.7	7.31
n40w200.001	6408	5	**6169**	2.88	7.79
n60w120.001	9303	20	**11120**	14.73	39.86
n60w140.001	9131	21	**10814**	14.1	38.15
n60w160.001	11422	17	**11574**	13.72	37.12
n60w180.001	9689	18	**11363**	13.25	35.85
n60w200.001	10315	17	**10128**	13.09	35.42
n80w120.001	11156	52	**11122**	21.32	57.69
n80w140.001	14131	43	**14131**	25.29	68.43
n80w160.001	8614	43	**9108**	22.99	62.21
n80w180.001	11222	56	**11222**	22.51	60.91
n80w200.001	8272	47	**8302**	39.02	105.58
n100w120.001	19246	73	22269	47.61	128.83
n100w140.001	22078	93	**22078**	49.53	134.02
n150w120.002	27192	388	**27192**	120.32	325.57
n200w120.003	17886	356	18010	160.2	433.48

Improved or same results are highlighted in boldface

In all tables, *Init.Sol*, *Best.Sol*, *Aver.Sol*, *T* correspond to the initial solution, best solution, average solution, and average time in seconds of ten executions obtained by the proposed algorithm, respectively while *BKS* is the best-known solution in the literature. *cTime* represents scaled run times, estimated on a Pentium IV by means of the factors of Dongarra Dongarra (2011). Tables from 1, 2, 3, 4 compare the results of our algorithm with the best-known solutions of the other algorithms in Ban and Nghia (2017b); Heilporna (2010); Silva and Urrutia (2010); Ohlmann and Thomas (2007). The results of the state-of-the-art metaheuristic algorithms are directly extracted from Ban and Nghia (2017b); Heilporna (2010); Silva and Urrutia (2010); Ohlmann and Thomas (2007). Our algorithm runs on the same instances with the other algorithms. Note that: Most of the results published in Ban and Nghia (2017b) are used to compare. In addition, Ban et al. Ban and Nghia (2017b) kindly provide us with their code as well

Table 4 Comparison results obtained by Ohlmann et al. Mladenovic et al. (2012), and Silva et al. Heilporna (2010) on TSPTW-instances

Instances	OPT	BKS	Best.Sol	Gap	T
n20w20	378	378	**378**	0.00	0.14
n20w40	254	254	**254**	0.00	0.15
n20w60	335	335	**335**	0.00	0.13
n20w80	329	329	**329**	0.00	0.13
n40w20	500	500	**500**	0.00	1.64
n40w40	465	465	**465**	0.00	1.87
n40w60	494	494	**494**	0.00	1.87
n40w80	395	395	**395**	0.00	1.77
n60w20	551	551	**551**	0.00	16.31
n60w40	591	591	**591**	0.00	16.34
n60w60	609	609	**609**	0.00	16.52
n80w20	616	616	**616**	0.00	29.01
n80w40	606	606	**606**	0.00	27.38
n80w60	554	554	**554**	0.00	27.20
n80w80	624	624	**624**	0.00	27.62
n100w20	738	738	**740**	0.00	48.03
n100w40	770	770	**770**	0.00	45.45
Aver				0.00	15.39

The optimal solutions are highlighted in boldface

as more detailed empirical results on different instances at the link [35]. To compare with their results, we run the proposed algorithm on the same instances.

In Tables 1 and 2, it is shown that the difference in the average gap between the construction and improvement phase is 2.49%. The average gap is rather small. It indicates that the construction phase returns good quality solutions fast. Although the improvement of the post phase upon the construction one is not too large, it is significant since 136 new best-known solutions are found compared with the other algorithms Ban and Nghia (2017b); Heilporna (2010). For the larger instances with up to 250 instances, our search is quite time-consuming. Hence, the first way to reduce the large running time is to run the construction phase with a slight loss of 2.49% solution quality on average.

Tables 1 and 2 show the results on the Dumas et al.'s Dumas et al. (1995) instances and compare them with the results obtained by the algorithm in Ban and Nghia (2017b). The quality of our solutions is much better than Ban et al. in Ban and Nghia (2017b) in most instances. The average result of −3.3% is slightly better than the one obtained by Ban et al. in Ban and Nghia (2017b), while the run time is comparable on average. For the instances with more than 200 vertices, feasible solutions have been unknown in the previous algorithms Ban and Nghia (2017b); Heilporna (2010); meanwhile, the feasible solutions for instances with up to 250 vertices can be found by our algorithm. It is also a significant result because building a feasible solution to the problem is NP-hard problem Salvesbergh (1985).

In Table 3, our result is comparable with Heilporn et al.'s algorithm Heilporna (2010). Specifically, it reaches the better solutions for 10 out of 24 instances. Besides, it obtains the same results for 4 cases. The improvement is significant when Heilporn et al.'s algorithm is one of the state-of-the-art metaheuristics for the problem.

Table 4 shows that most algorithms are developed for a specific variant that does not apply to the other variants. Our algorithm still runs well to TSPTW, although it was not designed for solving it. In comparison with the best-known solution in Silva and Urrutia (2010); Ohlmann and Thomas (2007), our algorithm's solutions obtain the optimal solutions for the instances with up to 100 vertices. The average gap between the best-known solution and our result is about 0.0%. It shows that our result reaches the best-known solutions for all instances.

The algorithm in Ban and Nghia (2017b) was executed on the nearly similar configuration with the proposed algorithm. However, G. Heilporna et al's and the proposed algorithm were executed on computers with different configurations. To compare the running time of them, a scaled running time (*cTime*) estimated on an AMD Opteron 275/2.2 Ghz (this configuration is almost the same as one used in Heilporna (2010)) by means of the factors of Dongarra Dongarra (2011) is used. In this case, the running time comparison is evaluated in a relative manner. From the experimental results, the running time of the proposed algorithm grows quite moderate compared to the Heilporn et al.'s algorithm Heilporna (2010) while it is comparable with the one of Ban et al.'s algorithm Ban and Nghia (2017b).

6 Discussions

According to our observation, there are numerous local optima and only one global optimal solution in DMPTW. The search space overall seems to exhibit the big-valley structure. The big valley hypothesis suggests that, in combinatorial optimization, local optima of good quality are clustered and surround the global optimum. It is an important characteristic of many hard problems from combinatorial optimization Reeves (1999). The structure suggests the idea of the hybrid approach between RVND with ILS, and Tabu, as follows. Firstly, the combination between RVND and ILS, in which ILS ensures diversification while RVND maintains intensification, generates many good locally optimal solutions dispersed over the global optimal solution. Secondly, TS is entirely attracted to the global optimal. Even though the initial solution is set far from the global optimal solution, TS is capable to prevent from getting trapped into cycles to drive the search to the global optimal solution.

With respect to the instances, the algorithm finds better solutions than Ban et al.'s algorithm Ban and Nghia (2017b) for 136 out of 160 instances. Moreover, for 24 instances, it obtains better solutions for 10 cases and the same quality for 4 cases when comparing with Heilporn et al.'s algorithm Heilporna (2010). For the large instances with up to 250 customers, this is the first time our algorithm has provided feasible solutions. It is a significant improvement because finding a feasible solution is also NP-hard problem. In TSPTW, our algorithm obtains the optimal solutions for the instances with up to 100 vertices. Moreover, it also reaches the best-known solutions for all instances. Although our purpose is not to provide metaheuristic for TSPTW,

the obtained results for this problem show the efficiency and broad applicability of our algorithm.

Our algorithm performs better than the others thanks to two reasons as followings:

(1) Heuristic algorithms such as Heilporna (2010) are often too greedy, therefore, it can get stuck into a local optimum in many cases. On the other hand, the metaheuristic approach is not greedy and may even accept a temporary deterioration of solution, which allows it to explore the solution space more thoroughly, and thus the chance to get better solutions is higher.
(2) The algorithm in Ban and Nghia (2017b) can get trapped into cycles in some cases, by using Tabu lists our algorithm overcomes their drawback and obtains better solutions. Moreover, the proposed algorithm uses more neighborhoods than the algorithm in Ban and Nghia (2017b); therefore, the explored solution space is larger. As a result, the chances of finding better solutions are higher.

7 Conclusions

In this work, DMPTW is studied. As our main contribution, we propose a metaheuristic algorithm that combines ILS, RNVD, and Tabu for DMPTW. We tested the algorithm on the benchmark dataset, comparing it to several state-of-the-art metaheuristic algorithms. Our algorithm is comparable with the state-of-the-art metaheuristic algorithms and it is able to find optimal solutions for instances with up to 100 vertices in a short time. In addition, for 136 instances, it provides new best-known solutions. In the future, we intend to extend the algorithm by including more neighborhoods and carefully studying the effectiveness of each neighborhood. Increasing the efficiency as well as the running time of our algorithm, even more, to allow even larger problems to be solved, is another future research topic.

Acknowledgements This research was supported by the Asahi Glass Foundation under grant number AGF.2020-02.

References

Ajama M, Akbari V, Salmana FS (2019) Minimizing latency in post-disaster road clearance operations. J Oper Res 277:1098–1112

Avci M, Avci MG (2017) A GRASP with iterated local search for the traveling repairman problem with profits. J Comput Ind Eng 113:323–332

Ban HB, Nguyen DN (2010) Improved genetic algorithm for Minimum Latency Problem. Proc. SOICT 9–15

Ban HB, Nguyen K, Ngo MC, Nguyen DN (2013) An efficient exact algorithm for Minimum Latency Problem. J PI 10:1–8

Ban HB, Nghia ND (2017) A Meta-Heuristic algorithm combining between Tabu and variable neighborhood search for the minimum latency problem. J FI 156(1):21–44

Ban HB, Nghia ND (2017) Metaheuristic for the Traveling Repairman Problem with Time Windows. Proc. RIVF 1–6

Beraldi P, Bruni ME, Lagana D, Musmanno R (2018) The Risk-Averse Traveling Repairman Problem with Profits. J. Soft Computing 1–15

Bertsimas D, Ryzin G (1989) "The Dynamic Traveling Repairman Problem, MIT Sloan School of Management Working Paper No. 3036-89-MS
Blum A, Chalasani P, Coppersmith D, Pulleyblank W, Raghavan P, Sudan M (1994) The Minimum Latency Problem. Proc. STOC 163–171
Bruni ME, Beraldi P, Khodaparasti S (2018) A heuristic Approach for the k-Traveling Repairman Problem with Profits under Uncertainty. J ENDM 69:221–228
Bruni ME, Beraldi P, Khodaparasti S (2019) A Hybrid Reactive GRASP Heuristic for the Risk-Averse k-Traveling Repairman Problem with Profits. J Comput Oper Res 115:1–16
Bruni ME (2020) Sara Khodaparasti and Samuel Nucamendi-Guillén, "The bi-objective Minimum Latency Problem with Profit Collection and Uncertain Travel Times". Proc. ICORES 2020:109–118
Chaudhuri K, Goldfrey B, Rao S, Talwar K (2003) Path, Tree and Minimum Latency Tour. Proc. FOCS 36–45
Dewilde T, Cattrysse D, Coene S, Frits CR, Spieksma FCR, Vansteenwegen P (2013) Heuristics for the Traveling Repairman Problem with Profits. J Comput Oper Res 40:1700–1707
Dumas Y, Desrosiers J, Gélinas E (1995) An optimal algorithm for the Traveling Salesman Problem with Time Windows. J Oper Res 43:367–371
Dongarra JJ (2011) Performance of various computers using standard linear equations software. Tech. Rep. CS-89-85, Computer Science Department, University of Tennessee, Knoxville, TN, USA
Fischetti M, Laporte G, Martello S (1993) The delivery man problem and cumulative matroids. J Oper Res. 41:1055–1064
Feo TA, Resende MGC (1995) "Greedy Randomized Adaptive Search Procedures", J. Global Opt., pp. 109-133
Gendreau M, Hertz A, Laporte G, Stan M (1998) A generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows. J. Oper Res 43:330–335
Goemans M, Kleinberg J (1996) An improved approximation ratio for the Minimum Latency Problem. Proc. SIAM SODA 152–158
Heilporna G, Cordeaua Jean-François, Laporte Gilbert (2010) The Delivery Man Problem with Time Windows. 7:269–282
Ibaraki T, Imahori S, Nonobe K, Sobue K, Uno T, Yagiura M (2008) An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. J Discrete Appl Math 11(156):2050–2069
Johnson DS, McGeoch LA. "The traveling salesman problem: A Case Study in Local Optimization in Local Search in Combinatorial Optimization", E. Aarts and J. K. Lenstra, eds., pp. 215-310
Martin O, Otto SW, Felten EW (1991) Large-step Markov Chains for the Traveling Salesman Problem. J Complex Syst 5(3):299–326
Mladenovic N, Hansen P (1997) Variable neighborhood search. J. Oper. Res. 24(11 24):1097–1100
Mladenovic N, Urosevi D, Hanafi S (2012) Variable neighborhood search for the Travelling Deliveryman Problem. J. 4OR 11:1–17
Salehipour A, Sorensen K, Goos P, Braysy O (2011) Efficient GRASP+VND and GRASP+VNS metaheuristics for the Traveling Repairman Problem. J Oper Res 9(2):189–209
Silva M, Subramanian A, Vidal T, Ochi L (2012) A simple and effective metaheuristic for the minimum latency problem. J EOR 221(3):513–520
Silva RF, Urrutia S (2010) A general VNS Heuristic for the Traveling Salesman Problem with Time Windows. J. Discrete Optim 7(4):203–211
Tsitsiklis JN (1992) Special cases of Traveling Salesman and Repairman Problems with time windows. J Netw 22:263–283
Talbi EG (2009) Metaheuristics: from Design to Implementation. Wiley, New Jersey
Ohlmann JW, Thomas BW (2007) A compressed-annealing heuristic for the traveling salesman problem with time windows. J Informs 19(1):80–90
Salvesbergh MW (1985) Local search in routing problems with time windows. J Ann Oper Res 4:285–305
Reeves CR (1999) Landscapes, operators and heuristic search. Ann Oper Res 86:473–490
https://sites.google.com/a/soict.hust.edu.vn/dmptw/trptw
https://homepages.dcc.ufmg.br/~rfsilva/tsptw/