RAIRO-Oper. Res. 53 (2019) 917–935
https://doi.org/10.1051/ro/2019006

RAIRO Operations Research
www.rairo-ro.org

AN EFFICIENT TWO-PHASE METAHEURISTIC ALGORITHM FOR THE TIME DEPENDENT TRAVELING SALESMAN PROBLEM

Ha Bang Ban*

Abstract. The Time Dependent Traveling Salesman Problem (TDTSP) is a class of NP-hard combinatorial optimization problems which has many practical applications. To the best of our knowledge, developing metaheuristic algorithm for the problem has not been studied much before, even though it is a natural and general extension of the Minimum Latency Problem (MLP) or Traveling Salesman Problem (TSP). In this paper, we propose an effective two-phase metaheuristic which combines the Insertion Heuristic (IH), Variable Neighborhood Search (VNS) and the tabu search (TS) to solve the problem. In a construction phase, the IH is used to create an initial solution that is good enough. In an improvement phase, the VNS is employed to generate diverse and various neighborhoods, while the main attribute of tabu search is to prohibit our algorithm from getting trapped into cycles, and to guide the search to escape local optima. Moreover, we introduce a novel neighborhoods' structure in VNS and present a $O(1)$ operation for calculating the cost of each neighboring solution in a special case of TDTSP where the TDTSP becomes the MLP. Extensive computational experiments on 355 benchmark instances show that our algorithm can find the optimal solutions for small instances with up to 100 vertices in a reasonable amount of time. For larger instances, our algorithm obtains the new best solutions in comparison with the state-of-the-art algorithm solutions.

Mathematics Subject Classification. 90B06.

Received February 28, 2017. Accepted December 31, 2018.

1. Introduction

The Time Dependent Traveling Salesman Problem (TDTSP) is a general variation of the classic Traveling Salesman Problem (TSP) and the Minimum Latency Problem (MLP) which has many practical applications. In the Time Dependent Traveling Salesman Problem (TDTSP), the amount of time for a salesman to travel from one vertex to another changes drastically depending on certain time of the day because of traffic congestion. By allowing the travel time between vertices to vary, the TDTSP is closer to several real practical situations such as heavy traffic, road repair, and automobile accidents than the traditional TSP and MLP can be [18]. Because it incorporates time dependent costs, the simplest generalization is the variant of the TDTSP considered in Picard *et al.* [10,27], which has applications within scheduling contexts and generalizes the well-known Traveling Salesman Problem. The improvement with respect to the traditional TSP is that the travel cost function between

Keywords. The Time Dependent Traveling Salesman Problem, tabu search, Variable Neighborhood Search.

Computer Science Department, School of Information and Communication Technology, Hanoi University of Science and Technology, No. 1 Dai Co Viet, Hanoi, Vietnam.
*Corresponding author: bangbh@soict.hust.edu.vn

Article published by EDP Sciences

two vertices depends not only on the distance, but also on the position of the edge in the tour. Another variant of the problem has been known as Time Dependent Vehicle Routing Problem (TDVRP) in [17, 18, 20], where a whole fleet must be routed instead of a single vehicle. Thus, the TDTSP in this paper is a particular case of TDVRP.

In the general case, the problem is described as NP-hard [31]. In this paper, we can formulate the TDTSP as follows:

Consider a complete graph $K_n = (V, E)$, with the set of vertices $V = 1, 2, \ldots, n$ and the set of edges E. Assume that there exists an $(n+1) \times (n+1)$ time-dependent cost matrix C and for a given edge $(i, j) \in E$, $c(i, j, t)$ is the travel time from i to j at time t. The TDTSP involves finding a minimum cost tour that visits each vertex exactly once, starting at a main depot. In the TSP, the cost is defined as $c(i, j, t) = 1 \times c_{ij}$ (the presence of t is assumed to be negligible) while in the MLP, the cost is given by $c(i, j, t) = (n - t + 1) \times c_{ij}$ (c_{ij} and t denote the distance from vertex i to j and the position order of the edge c_{ij} in the tour, respectively) [32]. Also, for solving the TDTSP, some vertices are typically designated as a depot, where the salesman begins his tour.

Many real-world variants of the TDTSP deal with scheduling time-dependent tasks. Picard *et al.* [10] showed the process of scheduling manufacturing jobs on a machine with time-dependent setup costs. Fox [13] and Fox *et al.* [14] considered some machine scheduling problems as variants of the TDTSP. Another variant of the TDTSP (such as the Deliveryman Problem (DMP)) in [27], was used to route vehicles through a manufacturing system. Moreover, other applications of the TDTSP were also introduced in [6, 22, 28]. Specifically, it routes data through a network [22], creates timetables for university exams [6], and schedules vehicles and crews [6]. Testa *et al.* [28] modeled the riding of amusement park attractions where the time spent waiting in line at each attraction varied with the time of day in TDTSP.

Because the TSP and MLP are NP-hard [7], the TDTSP is also NP-hard [20]. However, the TDTSP is a much more difficult problem than the TSP. Exact solutions for TSP with up to a thousand of vertices have been proposed [30], but exact algorithms can only solve TDTSP with a few dozen vertices [2]. Malandraki *et al.* [20] note that several well-known TSP metauheuristics do not easily adapt to the TDTSP. When the costs between vertices are changed in the TDTSP, the travel times of many other vertices later in the tour may be affected, and recomputing these travel times can be a time consuming task. They also show that certain properties of the optimal solutions to Euclidean TSP do not extend to the TDTSP. Specifically, the convex hull property presents in the optimal Euclidean TSP solutions does not hold for the TDTSP. Similarity, Blum *et al.* [7] also indicate that solving the MLP is much harder than TSP as follows: (1) In TSP, small changes in the structure of a metric space only affect local changes in the structure of the TSP. However, this can cause high non-local changes in the structure of the MLP problem; (2) the MLP tour may revisit points with an unbounded number of times even when the underlying graph has a bounded degree. In terms of complexity and difficulty, the MLP is closer to the TDTSP than TSP. Therefore, we chose the state-of-the-art metaheuristics for MLP in [25, 26] as a baseline in our research.

For NP-hard problems, such as the TDTSP, 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. Exact algorithms for the TDTSP with small sizes are presented in [8, 10, 31], for the special case of the MLP in [5, 35]. Secondly, the approximation algorithms produce a solution within some factor alpha of the optimal solution. To the best of our knowledge, no approximation algorithm has been proposed so far, however for the MLP several algorithms can be found in [1, 3, 7, 9, 15]. Thirdly, heuristic (or metaheuristic) algorithms perform well in practice and validate their empirical performance on an experimental benchmark of interesting instances. The metaheuristic algorithm depends on this approach.

Previously, several works based on heuristic approach have not been proposed for the TDTSP [20, 21, 32, 33]. Nevertheless, these heuristics often are problem-dependent techniques. They are designed to adapt to the specific problem and to take full advantage of the particularities of this problem. However, the disadvantage of pure heuristic is often too greedy, therefore, they usually get trapped in a local optimum and thus fail to obtain the

global optimum solution. Metaheuristics, on the other hand, are problem-independent techniques. In general, they are not greedy and may even accept a temporary deterioration of the solution, which allows them to explore more thoroughly the solution space and thus to get a better solution. Metaheuristic algorithms for the TDTSP have not been studied much and this work presents the first metaheuristic approach for this problem. However, several metaheuristics have been published for solving the case of the MLP problem in [4,25,26]. The experimental results in [4, 25, 26] also indicate that the metaheuristic approach is suitable to solve the MLP problem. The algorithms in [25,26] are mainly based on the principles of Variable Neighborhood Search (VNS). However, their algorithms might become trapped into cycles. That means they return to the points previously explored in the solution space. Consequently, the algorithms can get stuck in local optima. In this paper, we propose a metaheuristic algorithm that combines Tabu search (TS) and Variable Neighborhood Search (VNS) for the MLP problem. In our algorithm, TS is used to avoid getting trapped in cycles and to guide VNS to escape from local optima. In a cooperative way, the VNS is employed to generate diverse neighborhoods for the TS. Moreover, we introduce a novel neighborhoods' structure in VNS and present a $O(1)$ operation for calculating the cost of each neighboring solution in a special case of MLP. We have evaluated our algorithm on 355 benchmark instances. The experiment results show that our algorithm can find the optimal solutions for the instances with up to 100 vertices in a fraction of seconds. For larger instances, our algorithm obtains the new best solutions in comparison with the state-of-the-art algorithms.

The rest of this paper is organized as follows. Section 2 presents the proposed algorithm. Computational evaluations and Discussions are reported in Sections 3 and 4, respectively and Section 5 concludes the paper.

2. The proposed meta-heuristic algorithm

The idea of avoiding repeated moves in Tabu Search (TS) [16] is to make tabu lists of the recent types of moves in the space solution, and to prohibit reversing these moves. The move here is a transition from one solution to another. Variable Neighborhood Search (VNS) [23] based on a simple principle systematically switches between different neighborhoods. Therefore, VNS supports diverse neighborhoods for our algorithm. In this work, we give an algorithm which brings the advantages of TS and VNS together. A pseudocode of our algorithm is given in Algorithm 1. Our algorithm starts with an initial solution obtained from Insertion Heuristic (IH) in step 1 and then consists of four main steps (from step 2 to 5) repeated until a stop condition is met. In step 2, we investigate a novel neighborhoods' structure in VNS. Moreover, in order to avoid tabu move, tabu lists are used. In step 3, a list of promising solutions is built up and serves as an input for step 4. The step aims at exploiting the current solution space. In order to explore the entire solution space, a diversification phase is considered in step 5. In the remaining of this section, more details about the five steps of our algorithm are given.

Step 1. We use insertion heuristic which is given in Algorithm 2 for finding an initial solution. Consider a partial tour, and define the set $\overline{V}$ as the set of all non-visited nodes, $\overline{V} \subseteq V$. In order to improve the partial tour, a node from $\overline{V}$ should be added. This process requires two decisions: which vertex to insert and where to place it in the tour. In order to keep balance between pure greediness and overall layout of the tour, two insertion schemes are used. The major difference between insertion schemes is the order in which the vertices are inserted.

- Cheapest insertion: Among all vertices are not inserted so far, choose a vertex whose insertion causes the lowest increase in the cost of the tour. The idea behind this strategy is certainly pure greediness.
- Farthest insertion: Insert the vertex whose minimal distance to a tour vertex is maximal. The idea behind this strategy is to fix the overall layout of the tour early in the insertion process.

Step 2. In this step, eight neighborhoods such as remove-insert, swap-adjacent, move-down, move-up, swap-adjacent, 2-opt, move-forward-k-vertices and move-backward-k-vertices [34] are used according to the principle of VNS. For a given current solution T, local search explores the neighborhood $N_i(T)$ of T iteratively and tries to replace T by the best solution $T' \in N_i(T)$. The main operation in exploring the neighborhood is calculation of a neighboring solution's cost. In straightforward implementation, this operation requires $O(n)$ time in the

Algorithm 1. Our Algorithm.

Input: v_1, K_n are a starting vertex, the complete graph, respectively.
Output: the best solution T^*.

```
Step 1 (Generate an initial solution):
T ← IH-Procedure(v_1, V);
T* ← T; {initiate the best solution}
LT ← ∅; {LT is the list of promising solutions}
while stop criteria not met do
  Step 2 (VNS):
  for i : 1 → 6 do
    T' ← arg min N_i(T); {local search}
    if ((L(T') < L(T) and T' is not tabu) or (L(T') < L(T*))) then
      T ← T'
      i ← 1
      update tabu lists;
      if (L(T') < L(T*)) then
        T* ← T';
      end if
    else
      i++
    end if
  end for
  Step 3 (Built up promising solutions list LT):
  if L(T) < (1 + ST) × L(T*) then
    LT ← LT ∪ {T};
  end if
  if (|LT| < 5) then
    go to step 2;
  end if
  step 4 (Implement Intensification):
  for j : 1 → 5 do
    do VNS as in step 2 without tabu list with an element of LT as start solution
  end for
  step 5 (Implement Diversification):
  clear all tabu lists;
  select a random tour T in LT;
  swap randomly k vertices in T;
  go to step 2;
end while
return T*;
```

TDTSP and MLP. In TDTSP, when the positions of vertices in T are changed, the travel times of other vertices later in the tour may be affected. However, in the case of $c(i, j, t) = (n - t + 1)c_{ij}$ (the MLP problem), by using on the known cost of current solution, we show that this operation can be done in constant time for some considered neighborhoods. Thus, the running time of exploring these neighborhoods can be speed up. Now, let $T = (v_1, \ldots, v_k, \ldots, v_n)$ be a tour, we introduce a novel neighborhoods' structure and complexity of its exploration for the TDTSP and MLP.

Remove-insert neighborhood moves each vertex v_i in the solution at the end of it. This neighborhood of T is defined as a set $N_1(T) = \{T_i = (v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n, v_i) : i = 2, 3, \ldots, n\}$. Obviously, the size of $N_1(T)$ is $O(n)$.

Property 2.1. The time complexity of exploring $N_1(T)$ is $O(n^2)$ for the MLP and TDTSP.

Algorithm 2. IH-Procedure(v_1, K_n).

Input: v_1, K_n are a starting vertex, the complete graph, respectively.
Output: An initial solution T.

```
T = v_1;
while |T| < n do
    rd = random(2); {Choose an insertion scheme randomly}
    if rd == 1 then
        Arbitrary select a vertex v that is not yet in the partial tour and an inserted position j < |T| at time t_j so that the cost of T' (T' = Insert(T, j, v)) is minimal; {Cheapest Insertion}
    else
        Arbitrary select a vertex v that is not yet in the partial tour and an inserted position j < |T| at time t_j so that c(v_j, v, j) is minimal and the cost of T' (T' = Insert(T, j, v)) is maximal; {Farthest Insertion}
        T ← T'
    end if
end while
```

Proof. For TDTSP: Calculation of each neighborhood's cost requires $O(n)$ time, therefore, the time complexity of exploring $N_1(T)$ is $O(n^2)$.
For MLP: Let us consider an initial solution $T = v_1, v_2, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_n$. The neighborhood generates a neighboring solution $T_i = v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n, v_i$. The latencies of T and T_i are calculated as follows:

$$\begin{aligned} L(T) &= (n-1)c(v_1, v_2) + \ldots + (n-i+1)c(v_{i-1}, v_i) + (n-i)c(v_i, v_{i+1}) \\ &\quad + (n-i-1)c(v_{i+1}, v_{i+2}) + \ldots + (v_{n-1}, v_n). \\ L(T_i) &= (n-1)c(v_1, v_2) + \ldots + (n-i+1)c(v_{i-1}, v_{i+1}) + (n-i)c(v_{i+1}, v_{i+2}) \\ &\quad + \ldots + 2(v_{n-1}, v_n) + (v_n, v_i). \end{aligned} \tag{2.1}$$

We have

$$\begin{aligned} L(T_i) = L(T) &- \sum_{k=i-1}^{n-1} (n-k)c(v_k, v_{k+1}) + (n-i+1)v_{i-1}v_{i+1} \\ &+ \sum_{k=i+1}^{n-1} (n-k+1)c(v_k, v_{k+1}) + c(v_n, v_i). \end{aligned} \tag{2.2}$$

It takes $O(n)$ time to calculate the formulation in (2.2). Therefore, the time complexity of exploring $N_1(T)$ is $O(n^2)$. □

Swap adjacent neighborhood attempts to swap each pair of adjacent vertices in the solution. This neighborhood of T is defined as a set $N_2(T) = \{T_i = (v_1, v_2, \ldots, v_{i-2}, v_i, v_{i-1}, \ldots, v_n) : i = 3, 4, \ldots, n\}$. The size of the neighborhood is $O(n)$.

Property 2.2. The time complexity of exploring $N_2(T)$ is $O(n^2)$ and $O(n)$ for the TDTSP and MLP, respectively.

Proof. For TDTSP: It takes $O(n)$ time to calculate each neighborhood's cost, therefore, the time complexity of exploring $N_2(T)$ is $O(n^2)$.
For MLP: The initial tour T and $L(T)$ are the same as in (2.1). The neighborhood generates a neighboring tour $T_i = v_1, v_2, \ldots, v_{i-2}, v_i, v_{i-1}, v_{i+1}, \ldots, v_n$. The latency of T_i is calculated as follows:

$$\begin{aligned} L(T_i) &= (n-1)c(v_1, v_2) + \ldots + (n-i+2)c(v_{i-2}, v_i) + (n-i+1)c(v_i, v_{i-1}) \\ &\quad + (n-i)c(v_{i-1}, v_{i+1}) + (n-i-1)c(v_{i+1}, v_{i+2}) + \ldots + (v_{n-1}, v_n). \end{aligned}$$

We have

$$L(T_i) = L(T) - (n-i+2)c(v_{i-2}, v_{i-1}) - (n-i)c(v_i, v_{i+1}) + (n-i+2)c(v_{i-2}, v_i) + (n-i)c(v_{i-1}, v_{i+1}). \tag{2.3}$$

It is obvious that we can calculate $L(T_i)$ by the formulation (2.3) in $O(1)$ time. Therefore, the complexity of exploring $N_2(T)$ is $O(n)$ in the MLP. □

Swap neighborhood attempts to swap the positions of each pair of vertices in the solution. This neighborhood of T is defined as a set $N_3(T) = \{T_{ij} = (v_1, v_2, \ldots, v_{i-1}, v_j, v_{i+1}, \ldots, v_{j-1}, v_i, v_{j+1}, \ldots, v_n) : i = 2, 3, \ldots, n-2; j = i+2, \ldots, n\}$. The size of the neighborhood is $O(n^2)$.

Property 2.3. The complexity of exploring $N_3(T)$ is $O(n^3)$ and $O(n^2)$ for the TDTSP and MLP, respectively.

Proof. For TDTSP: Calculation of each neighborhood's cost requires $O(n)$ time, therefore, the time complexity of exploring $N_3(T)$ is $O(n^3)$.

For MLP: Initially, we have an initial solution $T = v_1, v_2, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_n (i+2 < j)$. Swap generates a neighboring solution $T' = v_1, v_2, \ldots, v_{i-1}, v_j, v_{i+1}, \ldots, v_{j-1}, v_i, v_{j+1}, \ldots, v_n$. The latencies of T and T_i are calculated as follows:

$$\begin{aligned} L(T) &= (n-1)c(v_1, v_2) + \ldots + (n-i+1)c(v_{i-1}, v_i) + (n-i)c(v_i, v_{i+1}) \\ &\quad + \ldots + (n-j+1)c(v_{j-1}, v_j) + (n-j)c(v_j, v_{j+1}) \\ &\quad + \ldots + (v_{n-1}, v_n). \\ L(T_{ij}) &= (n-1)c(v_1, v_2) + \ldots + (n-i+1)c(v_{i-1}, v_j) + (n-i)c(v_j, v_{i+1}) \\ &\quad + \ldots + (n-j+1)c(v_{j-1}, v_i) + (n-j)c(v_i, v_{j+1}) + \ldots + (v_{n-1}, v_n). \end{aligned} \tag{2.4}$$

We have

$$\begin{aligned} L(T_{ij}) &= L(T) - (n-i+1)c(v_{i-1}, v_i) - (n-i)c(v_i, v_{i+1}) \\ &\quad - (n-j+1)c(v_{j-1}, v_j) - (n-j)c(v_j, v_{j+1}) + (n-i+1)c(v_{i-1}, v_j) \\ &\quad + (n-i)c(v_j, v_{i+1}) + (n-j+1)c(v_{j-1}, v_i) + (n-j)c(v_i, v_{j+1}). \end{aligned} \tag{2.5}$$

It is obvious that we can calculate $L(T_{ij})$ by the formulation (2.5) in $O(1)$ time. Therefore, the complexity of exploring $L(T_{ij})$ is $O(n^2)$ in the MLP. □

2-opt neighborhood removes each pair of edges from the solution and reconnects the vertices. This neighborhood of T is defined as a set $N_4(T) = \{T_{ij} = (v_1, v_2, \ldots, v_i, v_j, v_{j-1}, \ldots, v_{i+2}, v_{i+1}, v_{j+1}, \ldots, v_n) : i = 1, \ldots, n-4; j = i+4, \ldots, n\}$. The size of the neighborhood is $O(n^2)$.

Property 2.4. The complexity of exploring $N_4(T)$ is $O(n^3)$ for the TDTSP and MLP.

Proof. For TDTSP: It takes $O(n)$ time to calculate each neighborhood's cost, therefore, the time complexity of exploring $N_4(T)$ is $O(n^3)$.

For MLP: The initial tour and $L(T)$ are the same as in (2.4). The neighborhood generates a neighboring tour $T_{ij} = (v_1, v_2, \ldots, v_i, v_j, v_{j-1}, \ldots, v_{i+2}, v_{i+1}, v_{j+1}, \ldots, v_n)$. The latencies of T and T_i are calculated as follows:

$$\begin{aligned} L(T_{ij}) &= (n-1)c(v_1, v_2) + \ldots + (n-i)c(v_i, v_j) + (n-i-1)c(v_j, v_{i+2}) \\ &\quad + \ldots + (n-j+1)c(v_{j-1}, v_{i+1}) + (n-j)c(v_{i+1}, v_{j+1}) + \ldots + (v_{n-1}, v_n). \end{aligned}$$

We have

$$L(T_{ij}) = L(T) - (n-i)c(v_i, v_{i+1}) - \sum_{h=1}^{j-i-1} (n-i-h)c(v_{i+h}, v_{i+h+1}) - (n-j)c(v_j, v_{j+1}) + (n-i)c(v_i, v_j) + \sum_{h=1}^{j-i-1} (n-i-h)c(v_{j-h+1}, v_{j-h}) + (n-j)c(v_{i+1}, v_{j+1}). \quad (2.6)$$

It is obvious that we can calculate $L(T_{ij})$ by the formulation (2.6) in $O(n)$ time. Therefore, the complexity of exploring $N_4(T)$ is $O(n^3)$ in the MLP. □

Move-forward-k-vertices neighborhood of T is defined as a set $N_5(T) = \{T_{ijk} = (v_1, v_2, \ldots, v_i, v_{i+k+1}, v_{i+k+2}, \ldots, v_j, v_{i+1}, v_{i+2} \ldots, v_{i+k}, v_{j+1}, \ldots, v_n) : i = 1, 2, \ldots, n-k-1;\ i+k < j \leq n\}$ with $k = 2, \ldots, l$. The size of the neighborhood is $O(n^2)$.

Move-backward-k-vertices neighborhood of T is defined as a set $N_6(T) = \{T_{ijk} = (v_1, v_2, \ldots, v_i, v_{i+k+1}, v_{i+k+2}, \ldots, v_{i+1}, v_{i+2} \ldots, v_{i+k}, v_j, v_{j+1}, \ldots, v_n) : i = 1, 2, \ldots, n-k-1;\ i+k < j \leq n\}$ with $k = 2, \ldots, l$. The size of the neighborhood is $O(n^2)$.

Property 2.5. The complexity of exploring $N_5(T)$ and $N_6(T)$ is $O(m^3)$ for the MLP, respectively.

Proof. We prove Property 2.5 for move-forward-k-vertices and the same argument holds for move-backward-k-vertices. For a tour $T_{ijk} \in N_5(T)$, it can be shown that

$$L(T_{ijk}) = L(T) - (n-i)c(v_i, v_{i+1}) - \sum_{h=i+1}^{j-1} (n-h)c(v_h, v_{h+1}) - (n-i)c(v_i, v_{i+1}) + (n-i)c(v_i, v_{i+k+1}) + \sum_{h=1}^{j-i-k-2} (n-i-h)c(v_{i+k+h}, v_{i+k+h+1}) + (n-j+k+1)c(v_{j-1}, v_j) + (n-j+k)c(v_j, v_{i+1}) + \sum_{h=1}^{k-1} (n-j+k-h)c(v_{i+h}, v_{i+h+1}) + (n-j+1)c(v_{i+k}, v_{j+1}). \quad (2.7)$$

It is obvious that we can calculate $L(T_{ijk})$ by the formulation (2.7) in $O(n)$ time. Therefore, the complexity of exploring $N_5(T)$ is $O(n^3)$ in the MLP. □

It is realized that calculation of a neighboring solution's cost by using on the known cost of current solution in (2.6) and (2.7) cannot be done in constant time. As a result, our algorithm spends $O(n^3)$ operations for a full neighborhood search. However, Silva *et al.* [26] suggest a move evaluation procedure, which only requires $O(1)$ amortized operations since the number of edge exchanges is bounded. In this work, we use their evaluation procedure for 2-opt and move forward(backward)-k-vertices. Therefore, the time complexity of exploring all neighborhoods in the worst case is performed in $O(n^2)$.

In each iteration, the best neighboring solution is accepted if it is non-tabu and improving, or tabu but globally improving. The reason of allowing a tabu move is that TS prevents to get trapped into a cycle, however it may lead to an overall stagnation of the searching process. The simplest and most commonly used aspiration criterion consists in allowing a move, even if it is in the tabu list but the cost of the new solution is better than that of the current best solution and this solution has obviously not been previously visited. Due to the

use of different neighborhood structures, several tabu lists are built. A move of the type remove-insert, swap-adjacent, or swap is stored in the first tabu list while the second is for 2-opt moves. We do not use tabu list for move-forward-k-vertices, move-backward-k-vertices.

Step 3. After finding a local optimum in step 2, step 3 builds up a set of promising solutions LT. Since the objective value of any local optimum lies within 5–10% of the best found solution, then it is added into LT. If the size of LT is five, then the algorithm moves to step 4. The size of LT is chosen to be five because a small value for the size of LT enhances more implementations to the intensification and diversification steps. The search can be moved to another area of the solution space without the previous area explored. Otherwise, since the value of $|LT|$ is large, less intensification and diversification step are performed.

Step 4. If the promising solution list LT is full, the intensification step implements. Each solution of the list is returned to step 2 without any restriction of tabu move. Since a new local optimum is found, the algorithm goes to step 5 in which a diverse solution to re-initialize the search is created.

Step 5. Firstly, the tabu lists are cleared. Secondly, the diversification step is performed by swapping several vertices randomly in each solution in the list LT and then we return to step 2 with these solutions.

The last aspect to discuss is the stop criterium of tabu search algorithm. A balance must be made between computation time and efficiency. Here, the algorithm stops if no improvement is found after a given number of iterations (NL).

3. Computational evaluation

The experiments are conducted on a personal computer, which is equipped with an Intel Pentium core i7 2.93 GHz and 8 GB RAM memory.

3.1. Datasets

There have been a few published studies [20, 21, 32, 33, 37] with heuristics developed for the TDTSP. In [21,32,33], unfortunately, we could not get the instances as well as their code and therefore, we cannot directly compare the results of our metaheuristic with the results obtained by their heuristics. However, the experimental results in [32] indicate that their solutions are slightly better than those obtained by the r-opt heuristic (within 3.6% on the average). The running time of their algorithm is about 5 times more than the running time of the r-opt. In order to compare to their algorithm, we implement the r-opt in [32] for TDTSP and then run it on benchmark. Similary, we also implement and then run the algorithm in [21] on benchmark. Based on the experimental results, we can compare relatively the efficiency of these algorithms. Our datasets in this paper are benchmarks in [2,20,25,36,37] that are divided into two types. The first type includes instances in TDTSP benchmark [20, 37]. A second group regards instances for MLP instances (*i.e.*, $w(i,j,t) = (n-t+1) \times c_{ij}$) [2, 25, 26].

In type 1, a first benchmark [37] is built from 255 addresses randomly chosen from a list of delivery tours of transporters from Lyon, the number of time steps ($m = 130$) and the duration of a time step ($d = 360$) seconds. They randomly generated 100 instances for each problem size ($n = 10, 20, 30, 50$, and 100) by randomly selecting n locations among the 255 positions referenced in the travel time function. The duration of each visit is randomly selected in the interval [60, 300] seconds. Because the transition function tends to underestimate the travel time in congested areas, they generated two additional versions of the function with a dilatation of travel times of respectively 10% and 20% centered on the average travel time. Therefore, they end up with three travel time functions. They also provide the best known solutions for these instances. We gather these instances into a group named as real dataset 1. In a second benchmark [20], Li *et al.* have studied the TDTSP and given several instances. They considered the Bier127 instance from TSPLIB [36] and defined a region in the city center in which traffic jams occurred in the afternoon. Congestion occurred in the afternoon for beer gardens in the dashed rectangle (the traffic jam region), so that the time to drive between two locations in the rectangle was multiplied by a jam factor $f > 1$. Each value of jam factor generates an instance. We gather these instances into a group named as real dataset 2.

In type 2, our experiments are performed in MLP instances derived from benchmark instances [25,26]. The cost of $w(i,j,t)$ is defined as $(n-t+1)\times c(i,j)$, where $c(i,j)$ is taken from the original distance matrix and t is the position order of the edge $c(i,j)$ in the tour. This allows direct comparisons with the state-of-the-art metaheuristic algoritms in [25,26].

The experimental data in type 2 includes three random datasets and two real datasets. In all instances, every distance between vertices satisfies the triangle inequality. Each instance contains the coordinates of n vertices. Based on the value of n, we divide the instances into two types: the small instances ($n \leq 50$) and the large instances ($n > 50$).

The first two random datasets comprise non-Euclidean and Euclidean instances, and were named as random dataset 1 and 2, respectively. In the former one, the instances were generated artificially with an arc cost drawn from a uniform distribution. The values of the arc costs were integers between 1 and 100. In the latter one, the Euclidean distance between two vertices was calculated. The coordinates of the vertices were randomly generated according to a uniform distribution in a 200×200 square. We chose the number of vertices from 40 to 100 and generated different instances for each dataset. The last random dataset supported by Salehipour *et al.* [25] was named as random dataset 3.

The real dataset 3 consists of the well-known real instances from TSPLIB, which are selected by Abeledo *et al.* [2] and Salehipour *et al.* [25]. Their sizes are between 42 and 318 vertices. Besides that, we added more real instances by randomly choosing partial data from the larger instances in TSPLIB. The number of vertices of each partial instance is forty. We divided the partial instances into three groups based on the values of standard deviation (STDV). We have analyzed the data of TSPLIB and found that instances mostly belong to one of the following three groups. Group one with STDV ≤ 100 where vertices are concentrated; group two, STDV ≥ 1000 where vertices are scattered; or group three where vertices are spaced in a special way such as along a line or evenly distributed. Specifically, group one includes the instances extracted from Eil51, St70, Eil76, and Rat195. In group two, the instances are chosen from KroA100, KroB100, KroC100, and Berlin52. In the last group, the instances are from Tsp225, Tss225, Pr76, and Lin105. In group 3, two instances tss225 and lin105 have the value of STDV in a range of 100 and 1000. We gathered the partial instances into a group named as real dataset 4.

3.2. Metrics

In order to evaluate the efficiency of metaheuristic algorithm, its solution can be compared to (1) the optimal solution (OPT); (2) a good upper bound (UB); (3) the best solutions in the previous algorithms.

We define the improvement of our algorithm (*Best.Sol* is our best solution) with respect to optimal solution ($gap_1[\%]$), and upper bound ($gap_2[\%]$) in percent respectively as follows:

$$gap_1[\%] = \frac{Best.Sol - \mathrm{OPT}}{\mathrm{OPT}} \times 100\% \tag{3.1}$$

$$gap_2[\%] = \frac{\mathrm{UB} - Best.Sol}{\mathrm{UB}} \times 100\%. \tag{3.2}$$

The optimal solutions can be found from [2,5]. Nevertheless, the exact algorithms only solve the problems with small size. When the optimal solutions have been unknown for large instance sizes, our solutions can be compared to the good upper bounds (r-opt) in [32], the known best solutions in [20,21,37] or the state-of-the-art metaheuristic algorithms in [25,26,32].

We see that in the formula (3.1), the smaller the value of gap_1 is, the better and closer to the optimal solution our solution is. Conversely, in formula (3.2), the larger the value of gap_2 is, the better and larger our solution's improvement upon UB is.

3.3. Results

Two experiments are conducted on 355 instances. The datasets in the first experiment include the small instances of random dataset from 1 to 3 and real dataset 1, 3 while the large instances of real dataset 1,

TABLE 1. The average results for the small instances.

Instances	$\overline{gap_1}$[%]	$\overline{gap_2}$[%]	$\overline{T}$	$\overline{cTime}$
INST-20-Rx	0	0	0.00	0.00
INST-30-Rx	0	5.01	1.12	2.98
Real dataset 3	0	5.04	0.22	0.59
Random dataset 1	0	5.59	0.20	0.53
Random dataset 2	0	5.48	0.21	0.56
TRP-50-Rx	0	9.82	0.62	1.65
INST-50-Rx	–	7.09	5.70	15.16

2 and 3 in [20, 36, 37] and random dataset 3 in [25] are used for the second. For the small instances, their optimal solutions allow us to evaluate exactly the efficiency of the algorithms. For the large instances, since their optimal solutions have been not known, the efficiency of the algorithms is only evaluated relatively. In two experiments, we choose $k = 10$, ST = 5% or 10%, $l = 5$, and NL = 150. In addition, in a pilot study, the performance of the algorithm relatively depends on the order in which the neighborhoods are used. Generally speaking, the neighborhoods which have smaller size will be explored first. Since our algorithm gets stuck in local optimum, the larger neighborhoods are used, because larger sized neighborhoods may help escape from local optimum. In this paper, the order of the neighborhoods is as follows: swap adjacent, remove-insert, swap, 2-opt, move-k-vertices-forward (-backward).

Given an algorithm, OPT, *Aver.Sol* and *Best.Sol* are the optimal, average and best solution after ten runs, respectively. Let T be the running time in seconds and $cTime$ represent scaled run times, estimated on a Pentium IV 2.4 GHz by means of the factors of Dongarra in [19] by seconds (note that Salehipour *et al.*'s experiments were implemented on a Pentium IV 2.4 GHz). In terms of running time, comparisons with the results published in [20,21,32] would be meaningless, due to the disparity between machines after almost two decades. Therefore, the running time of our algorithm only compares to those of the state-of-the-art algorithms for MLP in [25,26]. The experimental results of Li *et al.*'s (RTR), Lucena *et al.*'s (AL), Salehipour *et al.*'s (AS), Silva *et al.*'s (MS) algorithm and the known best solutions of Melgarejo *et al.* (PAM) are extracted from [20, 21, 25, 26, 37], respectively.

3.3.1. Experiment for small dataset

The experimental results are illustrated in Table 1, which are the average values calculated from Tables A.1 to A.5 in the Appendix. In Table 1, we denote $\overline{gap_i}(i = 1, 2)$ and $\overline{T}$ by the average values of gap_i and T for each dataset.

The experimental results in Table 1 shows that our algorithm can find the optimal solutions in a reasonable amount of time for all instances (with up to 50 vertices) in random dataset 1 and 2, TRP-50-x in random dataset 4, and INST-20-x, INST-30-x in real dataset 1 (these optimal solutions are extracted from [2, 5, 37]). For INST-50-x, our algorithm finds the new best solutions for most of instances. In comparsion with AL [21] in Tables A.3 and A.5, our algorithm outperforms for all instances. In addition, in average, the improvement of our algorithm upon the well-known upper bound ontained by r-opt is significant, since the average values of gap_2 are from 5.01% to 9.82%. Obviously, our algorithm outperforms the approach in [32] because their average of gap_2 [32] is in within 3.6%.

3.3.2. Experiment for large dataset

The experimental results are illustrated in Table 2, which are the average values calculated from Tables A.6 to A.10 in Appendix. In Table 2, we denote $\overline{gap_i}(i = 1, 2)$ and $\overline{T}$ by the average values of gap_i and T for each dataset.

TABLE 2. The average results for the large instances.

Instances	$\overline{gap_2}$[%]	$\overline{T}$	$\overline{cTime}$
TRP-100-Rx	13.59	7.74	20.67
TRP-200-Rx	14.38	79.14	211.30
INST-100-x	6.97	37.10	99.06
TSPLIB	8.79	4.49	11.99
Bierl27	0.34	36.55	97.59

In Table 2, for all instances, it can be observed that our algorithm is capable of improving the solution in comparison with the upper bound from [32]. The average improvement of our algorithm with the average gap_2 between 6.97% and 14.38% while the average gap_2 in [32] is about 3.6%. Similary, our algorithm obtains the better solutions than those of AL [21] in Tables A.7 and A.8. Obviously, our algorithm can obtain a significant improvement for almost instances and required small running time. Moreover, for INST-100-x in Table A.6 and a set of Bier127 instances with different jam factors in Table A.10, the new best solutions are obtained compared to the state-of-the-art previous solutions in [20,37].

Tables A.7 and A.8 also indicate that, for MLP instances, our algorithm gives a much better solutions than those of Salehipour *et al.*'s algorithm for all instances in random dataset 3. In comparison with Silva *et al.*'s algorithm, in all cases in TPR-200-Rx, our algorithm gives the solutions as well as those of Silva *et al.*'s algorithm. In addition, our algorithm obtains the better solutions for several instances (TRP-100-R2, TRP-100-R3, TRP-100-R9, TRP-100-R15, TRP-100-R17, TRP-100-R18, TRP-100-R20) in TRP-100-Rx. In Table A.9, our algorithm can find the optimal solutions (the optimal solutions are extracted from [2]) for the problems which is up to 100 vertices such as KroA100, KroB100, KroC100, KroD100 in several seconds. Obviously, our algorithm can be applicable to multiple variants of the TDTSP although it not is designed for solving them.

In all Tables, the average scaled running time of our algorithm is much better than those of Salehipour *et al.*'s, and it grows quite moderately with the one of Silva *et al.*'s algorithm.

4. DISCUSSIONS

In combinatorial optimization problem such as TDTSP, there exist many "locally optimal" solutions-not the absolutely best, but good enough as well as global optimal solution. Often, algorithms get trapped in a local optimum because (1) the explored part of the solution space is not large enough; (2) algorithms might become trapped into cycles. That means they return to the points previously explored in the solution space.

Our hybrid approach is between TS and VNS, as follows. First, many good elite solutions (local minima) created by VNS. Second, TS is perfectly attracted to global minima area. Even though the initial solution was set far from global minima, TS still can prevent from getting trapped into cycles in order to drive the search to global minima. In [12, 30], similar ideas have been proposed in order to hybridize different metaheuristic frameworks.

Our algorithm performs better than the other because of several reasons as followings:

(1) In the case of the general TDTSP, heuristic approaches in [20, 21, 32] are often too greedy, therefore, they usually get trapped in a local optimum and thus fail to obtain the global optimum solution. Our metaheuristic approach, on the other hand, is not greedy and may even accept a temporary deterioration of solution, which allows them to explore more thoroughly the solution space and thus to get ass better solution. The experiments show that our solutions are much better than those of the others in [20, 21, 32]. In addition, our algorithm finds the new best solutions in real dataset 1 and 2.
(2) In the case of the MLP, our algorithm uses six neighborhoods therefore, the explored part of the solution space is large enough. Hence, chances of finding even better solutions are high. The extension of explored

part is not time-consuming in our algorithm because of a constant time operation for calculating the cost of each neighboring solution. Moreover, in some cases, while the state-of-the-art algorithms in [25, 26] can get trapped into cycles, our algorithm overcomes their drawback and obtains the better solutions.

5. Conclusions

In this paper, we propose a new meta-heuristic algorithm which combines the Tabu search (TS) and Variable Neighborhood Search (VNS) for the TDTSP problem. The experimental results show that our algorithm is able to find the optimal solutions for the small instances, which is up to with 100 vertices, at a reasonable amount of time. For the larger instances, our solution's quality is comparable with the state-of-the-art metaheuristic algorithms, moreover, for sixty six instances it provides the new best solutions. However, the running time needs to be improved to meet real applications and enhancing it is still a challenge. This will be our aim in future research.

Appendix A.

Table A.1. The experimental results of our algorithm for real datatest 1 (INST-20-x).

Instances	Travel time function 1						Travel time function 2						Travel time function 3					
	UB	OPT	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T	UB	OPT	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T	UB	OPT	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T
INST-20-1	12 836	12 836	12 836	12 836	0	0	13 055	13 055	13 055	13 055	0	0	1305	13 233	13 233	13 233	0	0
INST-20-2	13 176	13 176	13 176	13 176	0	0	13 275	13 275	13 275	13 275	0	0	1327	13 220	13 220	13 220	0	0
INST-20-3	13 417	13 417	13 417	13 417	0	0	13 590	13 590	13 590	13 590	0	0	1359	13 714	13 714	13 714	0	0
INST-20-4	13 672	13 672	13 672	13 672	0	0	13 528	13 528	13 528	13 528	0	0	1352	13 646	13 646	13 646	0	0
INST-20-5	13 099	13 099	13 099	13 099	0	0	13 147	13 147	13 147	13 147	0	0	1314	13 242	13 242	13 242	0	0
INST-20-6	12 522	12 522	12 522	12 522	0	0	12 679	12 679	12 679	12 679	0	0	1267	12 749	12 749	12 749	0	0
INST-20-7	13 049	13 049	13 049	13 049	0	0	13 273	13 273	13 273	13 273	0	0	1327	13 323	13 323	13 323	0	0
INST-20-8	12 504	12 504	12 504	12 504	0	0	12 757	12 757	12 757	12 757	0	0	1275	12 892	12 892	12 892	0	0
INST-20-9	13 900	13 900	13 900	13 900	0	0	13 990	13 990	13 990	13 990	0	0	1399	13 852	13 852	13 852	0	0
INST-20-10	13 734	13 734	13 734	13 734	0	0	13 690	13 690	13 690	13 690	0	0	1369	13 542	13 542	13 542	0	0
INST-20-11	13 035	13 035	13 035	13 035	0	0	13 149	13 149	13 149	13 149	0	0	1314	13 155	13 155	13 155	0	0
INST-20-12	13 221	13 221	13 221	13 221	0	0	13 511	13 511	13 511	13 511	0	0	1351	13 446	13 446	13 446	0	0
INST-20-13	11 201	11 201	11 201	11 201	0	0	11 617	11 617	11 617	11 617	0	0	1161	11 814	11 814	11 814	0	0
INST-20-14	11 697	11 697	11 697	11 697	0	0	11 943	11 943	11 943	11 943	0	0	1194	12 146	12 146	12 146	0	0
INST-20-15	13 647	13 647	13 647	13 647	0	0	13 934	13 934	13 934	13 934	0	0	1393	13 899	13 899	13 899	0	0
INST-20-16	11 921	11 921	11 921	11 921	0	0	12 201	12 201	12 201	12 201	0	0	1220	12 245	12 245	12 245	0	0
INST-20-17	13 170	13 170	13 170	13 170	0	0	13 324	13 324	13 324	13 324	0	0	1332	13 500	13 500	13 500	0	0
INST-20-18	12 930	12 930	12 930	12 930	0	0	13 148	13 148	13 148	13 148	0	0	1314	13 026	13 026	13 026	0	0
INST-20-19	12 932	12 932	12 932	12 932	0	0	13 152	13 152	13 152	13 152	0	0	1315	13 220	13 220	13 220	0	0
INST-20-20	13 056	13 056	13 056	13 056	0	0	13 150	13 150	13 150	13 150	0	0	1315	13 234	13 234	13 234	0	0
Aver					0	0					0	0					0	0

TABLE A.2. The experimental results of our algorithm for real datatest 1 (INST-30-x).

Instances	Travel time function 1						Travel time function 2						Travel time function 3					
	UB	OPT	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T	UB	OPT	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T	UB	OPT	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T
INST-30-1	18 460	17 167	17 167	17 167	7.00	1.2	18 253	17 067	17 067	17 067	6.50	1.0	17 790	17 206	17 206	17 206	3.28	1.1
INST-30-2	16 437	15 853	15 853	15 853	3.55	1.1	16 410	15 672	15 672	15 672	4.50	1.1	17 319	15 946	15 946	15 946	7.93	1.1
INST-30-3	18 060	17 202	17 202	17 202	4.75	1.3	17 338	16 953	16 953	16 953	2.22	1.1	17 683	16 697	16 697	16 697	5.58	1.2
INST-30-4	17 324	15 688	15 688	15 688	9.44	1.0	17 396	15 636	15 636	15 636	10.12	1.0	16 680	15 801	15 801	15 801	5.27	1.0
INST-30-5	15 033	15 022	15 022	15 022	0.07	1.2	15 462	15 086	15 086	15 086	2.43	1.2	16 244	15 170	15 170	15 170	6.61	1.0
INST-30-6	18 342	17 192	17 192	17 192	6.27	1.2	18 010	16 949	16 949	16 949	5.89	1.1	17 853	16 925	16 925	16 925	5.20	1.2
INST-30-7	16 768	16 056	16 056	16 056	4.25	1.2	16 840	16 518	16 518	16 518	1.91	1.1	17 010	16 243	16 243	16 243	4.51	1.2
INST-30-8	18 649	17 431	17 431	17 431	6.53	1.1	17 856	17 051	17 051	17 051	4.51	1.0	18 357	16 790	16 790	16 790	8.54	1.1
INST-30-9	16 692	15 963	15 963	15 963	4.37	1.2	16 512	15 442	15 442	15 442	6.48	1.1	16 673	15 739	15 739	15 739	5.60	1.2
INST-30-10	17 328	16 525	16 525	16 525	4.63	1.1	17 281	17 052	17 052	17 052	1.33	1.0	17 774	16 423	16 423	16 423	7.60	1.2
INST-30-11	17 994	17 434	17 434	17 434	3.11	1.0	18 059	17 342	17 342	17 342	3.97	1.1	18 479	17 096	17 096	17 096	7.48	1.2
INST-30-12	17 410	17 404	17 404	17 404	0.03	1.1	17 566	16 508	16 508	16 508	6.02	1.2	17 725	16 356	16 356	16 356	7.72	1.1
INST-30-13	18 870	17 576	17 576	17 576	6.86	1.3	18 362	17 336	17 336	17 336	5.59	1.2	18 205	17 143	17 143	17 143	5.83	1.2
INST-30-14	19 878	18 595	18 595	18 595	6.45	1.0	19 336	18 250	18 250	18 250	5.62	1.0	18 984	17 907	17 907	17 907	5.67	1.1
INST-30-15	18 236	17 313	17 313	17 313	5.06	1.1	18 386	17 438	17 438	17 438	5.16	1.0	17 503	16 749	16 749	16 749	4.31	1.1
INST-30-16	15 185	14 927	14 927	14 927	1.70	1.0	14 893	14 893	14 893	14 893	0.00	1.2	15 501	14 947	14 947	14 947	3.57	1.0
INST-30-17	18 870	17 188	17 188	17 188	8.91	1.1	17 958	17 124	17 124	17 124	4.64	1.2	18 486	16 732	16 732	16 732	9.49	1.2
INST-30-18	17 879	17 605	17 605	17 605	1.53	1.0	17 809	17 296	17 296	17 296	2.88	1.1	17 425	16 616	16 616	16 616	4.64	1.1
INST-30-19	18 680	18 397	18 397	18 397	1.51	1.3	19 434	17 963	17 963	17 963	7.57	1.0	19 124	17 684	17 684	17 684	7.53	1.2
INST-30-20	17 849	16 960	16 960	16 960	4.98	1.1	17 831	17 007	17 007	17 007	4.62	1.2	17 601	17 380	17 380	17 380	1.26	1.2
Aver					4.55	1.13					4.6	1.10					5.88	1.12

TABLE A.3. The experimental results for the small instances in real dataset 4, random dataset 1, and random dataset 2.

Dataset	Instances	UB	OPT	AL		Our algorithm				
				Best.Sol	*T*	*Best.Sol*	*Aver.Sol*	gap_1 [%]	gap_2 [%]	*T*
Real dataset 4	eil51	6233	5934	6294	0.2	5934	5934	0	4.80	0.19
	eil76	6551	6239	6872	0.2	6239	6239	0	4.76	0.20
	st70	8156	7754	9348	0.3	7754	7754	0	4.93	0.20
	ratl95	18 871	17 971	22 453	0.2	17 971	17 971	0	4.77	0.27
	kroA100	274 015	260 792	305 869	0.3	260 792	260 792	0	4.83	0.19
	kroB100	258 042	245 649	325 733	0.2	245 649	245 649	0	4.80	0.26
	kroC100	293 452	272 638	325 733	0.2	272 638	272 638	0	7.09	0.20
	berlin52	98 218	93 490	136 970	0.2	93 490	93 490	0	4.81	0.25
	pr76	1 263 695	1 203 069	1 522 361	0.3	1 203 069	1 203 069	0	4.80	0.18
	tsp225	30 544	28 951	141 632	0.2	28 951	28 951	0	5.22	0.26
	tss225	1 273 908	1 212 096	1 352 430	0.3	1 212 096	1 212 096	0	4.85	0.19
	lin105	146 588	139 600	141 632	0.3	139 600	139 600	0	4.77	0.19
	Aver				0.2			0	5.04	0.22
Random dataset 1	test 1	8514	7767	8781	0.2	7767	7767	0	8.77	0.24
	test 2	9416	8958	10 230	0.1	8958	8958	0	4.86	0.21
	test 3	8711	8279	8861	0.2	8279	8279	0	4.96	0.16
	test 4	9081	8648	10 084	0.2	8648	8648	0	4.77	0.18
	test 5	8263	7862	8462	0.3	7862	7862	0	4.85	0.26
	test 6	9545	9082	11 296	0.2	9082	9082	0	4.85	0.25
	test 7	8701	8216	8991	0.1	8216	8216	0	5.57	0.16
	test 8	8435	7995	8736	0.2	7995	7995	0	5.22	0.17
	test 9	9085	8644	10 751	0.2	8644	8644	0	4.85	0.16
	test 10	10 096	9369	10 754	0.3	9369	9369	0	7.20	0.16
	Aver				0.2			0	5.59	0.20
Random dataset 2	test 1	9156	8706	11 137	0.2	8706	8706	0	4.91	0.17
	test 2	8219	7763	8136	0.3	7763	7763	0	5.55	0.16
	test 3	8729	8292	10 433	0.2	8292	8292	0	5.01	0.26
	test 4	8871	8423	9792	0.2	8423	8423	0	5.05	0.20
	test 5	9781	9311	9964	0.3	9311	9311	0	4.81	0.27
	test 6	9015	8466	10 219	0.3	8466	8466	0	6.09	0.20
	test 7	9081	8642	10 611	0.2	8642	8642	0	4.83	0.18
	test 8	9088	8443	9893	0.2	8443	8443	0	7.10	0.17
	test 9	9834	9183	9957	0.3	9183	9183	0	6.62	0.23
	test 10	9337	8888	10 494	0.3	8888	8888	0	4.81	0.28
	Aver				0.3			0	5.48	0.21

TABLE A.4. The experimental results of our algorithm for real datatest 1 (INST-50-x).

Instances	Travel time function 1						Travel time function 2						Travel time function 3					
	UB	PAM	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T	UB	PAM	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T	UB	PAM	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T
INST-50-1	24 213	22 846	22 795	22795.0	5.86	5.6	23 659	22 108	22 108	22259.0	6.56	5.5	23 786	21 678	21 678	21678.0	8.86	6.1
INST-50-2	25 115	23 713	23 421	23421.0	6.74	5.4	24 703	23 233	22 661	22661.0	8.27	6.0	24 077	22 676	21 678	21678.0	9.96	5.7
INST-50-3	24 753	22 684	22 684	22684.0	8.36	6.1	23 451	21 777	21 777	22097.0	7.14	5.7	25 152	21 382	21 217	21362.2	15.64	6.2
INST-50-4	27 004	24 833	24 396	24421.5	9.66	6.4	25 427	23 610	23 579	23579.0	7.27	6.2	22 177	22 949	22 949	22949.0	3.48	6.1
INST-50-5	23 885	20 960	20 960	20960.0	12.25	6.0	22 654	20 877	20 877	20886.1	7.84	6.1	22 751	20 553	20 553	20553.0	9.66	6.2
INST-50-6	23 601	22 396	22 074	22074.0	6.47	5.7	23 078	21 795	21 380	21380.0	7.36	5.2	24 165	21 276	21 219	21219.0	12.19	6.7
INST-50-7	25 045	23 241	23 241	23242.8	7.20	5.8	25 490	22 645	22 645	22867.0	11.16	6.4	24 785	22 308	22 308	22308.0	9.99	5.3
INST-50-8	24 559	23 274	23 274	23274.0	5.23	5.1	23 271	22 558	22 558	22584.0	3.06	5.4	22 412	22 182	22 182	22188.5	1.03	5.7
INST-50-9	24 146	22 549	22 549	22549.0	6.61	5.5	23 207	22 015	22 015	22417.0	5.14	5.8	22 429	21 669	21 669	21669.0	3.39	5.4
INST-50-10	22 821	22 831	22 556	22557.3	1.16	5.5	22 648	22 249	21 778	21852.2	3.84	6.1	23 962	21 928	21 639	21639.0	9.69	5.0
INST-50-11	25 069	23 893	23 775	23775.0	5.16	6.0	24 347	23 063	23 015	23015.0	5.47	5.6	24 500	22 230	22 230	22230.0	9.27	5.8
INST-50-12	25 070	24 610	24 487	24487.0	2.33	5.4	25 006	23 969	23 792	23792.0	4.85	5.7	23 601	23 568	23 495	23495.0	0.45	5.5
INST-50-13	24 459	23 432	23 432	23432.0	4.20	6.2	23 918	22 585	22 585	22730.0	5.57	6.1	23 521	22 190	22 190	22191.2	5.66	5.5
INST-50-14	24 358	23 678	23 356	23357.3	4.11	5.8	23 661	22 737	22 677	22715.3	4.16	6.2	22 932	22 103	22 103	22103.0	3.62	5.4
INST-50-15	23 682	22 473	22 473	22473.0	5.11	5.6	22 972	21 838	21 838	22247.0	4.94	5.6	24 058	22 057	21 919	21919.0	8.89	5.1
INST-50-16	25 445	23 538	23 538	23538.0	7.49	6.4	25 019	23 035	23 032	23032.0	7.94	5.9	24 023	22 463	22 463	22463.0	6.49	5.4
INST-50-17	24 721	24 100	24 028	24028.0	2.80	6.2	24 397	23 311	23 273	23273.0	4.61	6.1	21 726	23 207	22 858	22861.2	5.21	5.7
INST-50-18	21 998	21 539	21 181	21181.0	3.71	6.3	21 658	20 785	20 785	20924.0	4.03	5.6	24 821	20 681	20 494	20494.0	17.43	5.8
INST-50-19	25 919	24 477	24 477	24477.0	5.56	6.3	25 392	23 899	23 899	23974.0	5.88	6.0	23 208	23 332	23 332	23332.0	0.53	5.3
INST-50-20	23 906	23 170	22 528	22528.0	5.76	6.2	23 657	22 499	22 180	22180.0	6.24	5.2	41 383	21 984	21 983	21983.0	46.88	5.8
Aver					5.79						6.07						9.42	5.7

TABLE A.5. The experimental results of our algorithm for random datatest 3 (TPR-50-Rx).

Instances	UB	OPT	AL		Our algorithm				
			Best.Sol	T	*Best.Sol*	*Aver.Sol*	gap_1 [%]	gap_2[%]	T
TRP-50-R1	13 479	12 198	14 775	0.8	12 198	12 198	0	9.50	0.56
TRP-50-R2	12 821	11 621	13 742	0.7	11 621	11 621	0	9.36	0.67
TRP-50-R3	13 723	12 139	13 765	0.7	12 139	12 139	0	11.54	0.67
TRP-50-R4	14 482	13 071	15 160	0.3	13 071	13 071	0	9.74	0.68
TRP-50-R5	13 381	12 126	13 414	0.7	12 126	12 126	0	9.38	0.65
TRP-50-R6	14 169	12 684	14 542	0.7	12 684	12 684	0	10.48	0.61
TRP-50-R7	12 791	11 176	12 690	0.6	11 176	11 176	0	12.63	0.61
TRP-50-R8	14 233	12 910	14 060	0.5	12 910	12 910	0	9.30	0.62
TRP-50-R9	13 903	13 149	14 535	0.6	13 149	13 149	0	5.42	0.63
TRP-50-R10	14 729	12 892	15 327	0.7	12 892	12 892	0	12.47	0.63
TRP-50-R11	13 567	12 103	13 931	0.7	12 103	12 103	0	10.79	0.64
TRP-50-R12	11 812	10 633	12 823	0.6	10 633	10 633	0	9.98	0.62
TRP-50-R13	13 339	12 115	14 874	0.7	12 115	12 115	0	9.18	0.50
TRP-50-R14	14 678	13 117	13 817	0.6	13 117	13 117	0	10.63	0.65
TRP-50-R15	12 948	11 986	13 719	0.3	11 986	11 986	0	7.43	0.67
TRP-50-R16	13 577	12 138	15 126	0.8	12 138	12 138	0	10.60	0.65
TRP-50-R17	13 576	12 176	13 907	0.6	12 176	12 176	0	10.31	0.51
TRP-50-R18	14 723	13 357	14 532	0.6	13 357	13 357	0	9.28	0.57
TRP-50-R19	12 505	11 430	13 298	0.7	11 430	11 430	0	8.60	0.63
TRP-50-R20	13 214	11 935	13 371	0.7	11 935	11 935	0	9.68	0.63
Aver				0.63			0	9.82	0.62

TABLE A.6. The experimental results of our algorithm for real datatest 1 (INST-100-x).

Instances	Travel time function 1						Travel time function 2						Travel time function 3					
	UB	PAM	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T	UB	PAM	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T	UB	PAM	*Best. Sol*	*Aver. Sol*	gap_2 [%1]	T
INST-100-1	41 831	39 249	39 532	39 532	5.50	38.3	39 139	39 336	38 245	38 245	2.28	41.8	41 363	36 884	36 884	36 884	10.83	38.6
INST-100-2	38 954	36 015	36 015	36 015	7.54	41.7	36 382	35 065	35 143	35 143	3.41	45.8	38 391	34 793	33 987	33 987	11.47	46.3
INST-100-3	38 010	36 807	36 807	36 807	3.16	38.3	40 597	37 240	35 729	35 729	11.99	43.0	41 085	37 000	34 477	34 477	16.08	39.2
INST-100-4	42 257	39 631	39 631	39 631	6.21	32.8	38 582	38 954	38 240	38 240	0.89	32.0	42 419	39 558	37 765	37 765	10.97	35.0
INST-100-5	39 215	37 377	38 396	38 396	2.09	33.9	39 712	35 852	35 852	35 852	9.72	36.8	41 824	34 580	34 580	35854.6	17.32	37.2
INST-100-6	39 769	38 312	37 957	37 957	4.56	40.6	40 963	36 508	36 508	36 508	10.88	45.1	40 090	34 950	34 950	34 950	12.82	43.5
INST-100-7	41 818	40 371	40 324	41462.7	3.57	26.0	41 273	37 971	39 081	39 081	5.31	43.5	41 107	36 395	36 395	36 395	11.46	34.5
INST-100-8	43 023	39 302	39 302	39 302	8.65	29.6	40 214	39 325	39 257	39 257	2.38	36.7	43 260	37 250	37 250	37 250	13.89	41.5
INST-100-9	40 434	38 189	38 189	38 189	5.55	40.5	41 349	36 813	36 813	36926.3	10.97	36.5	44 793	35 808	35 808	35 808	20.06	26.6
INST-100-10	41 875	40 021	40 021	40 021	4.43	31.7	45 449	37 958	37 958	37 958	16.48	32.7	40 021	37 185	37 185	37 185	7.09	38.9
INST-100-11	44 023	40 486	42 661	42 661	3.09	42.3	42 431	40 937	40 268	40504.6	5.10	42.4	40 935	39 210	39 195	40201.7	4.25	33.9
INST-100-12	41 672	41 122	41 026	41 026	1.55	34.8	40 093	38 727	38 727	38 727	3.41	33.9	41 730	37 730	37 730	37 730	9.59	41.5
INST-100-13	40 742	39 439	39 439	39441.3	3.20	32.3	39 463	38 188	38 188	38 188	3.23	38.3	38 441	37 023	37 023	37 023	3.69	28.9
INST-100-14	39 123	38 089	38 089	38 089	2.64	34.3	41 751	36 358	36 358	36358.6	12.92	33.8	42 503	35 553	35 553	35 553	16.35	33.0
INST-100-15	42 645	39 826	39 815	39 815	6.64	40.8	40 307	39 706	38 599	38 599	4.24	40.6	42 702	38 680	38 680	38 680	9.42	34.5
INST-100-16	41 135	39 128	39 128	39 128	4.88	39.0	41 119	38 964	37 923	37 923	7.77	33.4	40 639	37 010	37 239	37 239	8.37	39.7
INST-100-17	42 788	41 595	41 595	41 595	2.79	44.1	42 424	40 181	39 928	39 928	5.88	44.2	39 755	39 438	39 438	39931.7	0.80	35.5
INST-100-18	41 870	39 582	39 582	39612.0	5.46	42.0	40 527	39 486	39 302	39 302	3.02	34.6	41 363	37 977	37 977	37 977	8.19	36.4
INST-100-19	41 401	40 032	40 032	40 032	3.31	45.5	41 770	38 459	38 459	38 459	7.93	42.3	38 391	38 071	37 949	37 949	1.15	38.3
INST-100-20	41 587	40 163	40 723	40 723	2.08	37.7	37 371	38 212	38 212	38 212	2.25	33.5	41 085	36 619	36 619	36 619	10.87	39.1
Aver					4.35	37.3					6.50	38.6					10.23	37.1

TABLE A.7. The experimental results of our algorithm for random datatest 3 (TPR-100-Rx).

Instances	UB	AS		MS		AL		Our algorithm			
		Best.Sol	gap_2[%]	*Best.Sol*	gap_2[%]	*Best.Sol*	T	*Best.Sol*	*Aver.Sol*	gap_2[%]	T
TRP-100-R1	35 334	–	–	32 779	7.23	36 943	82	32 779	32779.0	7.23	8.43
TRP-100-R2	38 442	–	–	33 435	13.02	42 134	86	32 778	32778.0	14.73	8.71
TRP-100-R3	37 642	–	–	32 390	13.95	36 696	78	31 654	31654.0	15.91	8.56
TRP-100-R4	37 508	–	–	34 733	7.40	38 744	89	35 208	35212.2	6.13	8.46
TRP-100-R5	37 215	–	–	32 598	12.41	37 666	86	32 906	32906.0	11.58	8.28
TRP-100-R6	40 422	–	–	34 159	15.49	42 652	67	34 159	34159.0	15.49	8.67
TRP-100-R7	37 367	–	–	33 375	10.68	39 479	89	33 375	33375.0	10.68	8.28
TRP-100-R8	38 086	–	–	31 780	16.56	39 595	82	31 981	31981.0	16.03	8.18
TRP-100-R9	36 000	–	–	34 167	5.09	40 033	87	33 687	33687.0	6.43	8.73
TRP-100-R10	37 761	–	–	31 605	16.30	36 669	84	31 605	31605.0	16.38	8.61
TRP-100-R11	37 220	–	–	34 188	8.15	40 722	89	34 285	34285.0	7.89	8.01
TRP-100-R12	34 785	–	–	32 146	7.59	42 507	85	32 146	32146.0	7.59	7.98
TRP-100-R13	37 863	–	–	32 604	13.89	37 888	67	32 604	32604.0	13.89	8.75
TRP-100-R14	36 362	–	–	32 433	10.81	37 364	86	32 433	32435.2	10.81	7.71
TRP-100-R15	39 381	–	–	32 574	17.28	43 038	82	32 223	32223.0	18.18	7.90
TRP-100-R16	39 823	–	–	33 566	15.71	42 031	86	33 094	33094.0	16.9	8.93
TRP-100-R17	41 824	–	–	34 198	18.23	43 136	86	33 813	33813.0	19.15	7.08
TRP-100-R18	39 091	–	–	31 929	18.32	40 349	89	31 898	31898.0	18.4	8.95
TRP-100-R19	39 941	–	–	33 463	16.22	40 679	86	33 463	33463.0	16.22	8.38
TRP-100-R20	39 888	–	–	33 632	15.68	41 627	86	31 035	31035.0	22.19	8.33
Aver			11.56		13.00					13.59	8.30

TABLE A.8. The experimental results of our algorithm for random datatest 3 (TPR-200-Rx).

Instances	UB	AS		MS		AL		Our algorithm			
		Best.Sol	gap_2[%]	*Best.Sol*	gap_2[%]	*Best.Sol*	*T*	*Best.Sol*	*Aver.Sol*	gap_2[%]	*T*
TRP-200-R1	105 044	–	–	88 787	15.48	112 847	126	88 787	90132.0	15.48	78.46
TRP-200-R2	104 073	–	–	91 977	11.62	108 519	128	91 977	92334.0	11.62	80.81
TRP-200-R3	111 644	–	–	92 568	17.09	114 068	130	92 568	93053.0	17.09	77.68
TRP-200-R4	104 956	–	–	93 174	11.23	109 425	123	93 174	93174.0	11.23	89.97
TRP-200-R5	101 912	–	–	88 737	12.93	108 657	126	88 737	88737.0	12.93	75.79
TRP-200-R6	103 751	–	–	91 589	11.72	106 015	113	91 589	92183.0	11.72	80.82
TRP-200-R7	109 810	–	–	92 754	15.53	108 226	126	92 754	93212.3	15.53	76.75
TRP-200-R8	103 830	–	–	89 048	14.24	101 162	127	89 048	89048.0	14.24	76.16
TRP-200-R9	100 946	–	–	86 326	14.48	105 503	125	86 326	86326.0	14.48	78.30
TRP-200-R10	108 061	–	–	91 552	15.28	109 227	125	91 552	91976.0	15.28	81.41
TRP-200-R11	103 297	–	–	92 655	10.3	110 472	127	92 655	93102.0	10.3	83.14
TRP-200-R12	107 715	–	–	91 457	15.09	107 401	122	91 457	91457.0	15.09	73.51
TRP-200-R13	100 505	–	–	86 155	14.28	114 189	126	86 155	87002.0	14.28	77.83
TRP-200-R14	107 543	–	–	91 882	14.56	123 716	127	918 82	91882.0	14.56	85.33
TRP-200-R15	100 196	–	–	88 912	11.26	108 419	128	88 912	88914.5	11.26	72.93
TRP-200-R16	104 462	–	–	89 311	14.5	103 674	125	89 311	90145.0	14.5	77.49
TRP-200-R17	107 216	–	–	89 089	16.91	112 553	128	89 089	89089.0	16.91	77.70
TRP-200-R18	108 148	–	–	93 619	13.43	119 927	128	93 619	93619.5	13.43	79.65
TRP-200-R19	105 716	–	–	93 369	11.68	117 641	129	93 369	93369.0	11.68	78.96
TRP-200-R20	116 676	–	–	86 292	26.04	112 547	122	86 292	86292.0	26.04	80.02
Aver			11.33		14.38					14.38	79.14

TABLE A.9. The experimental results for the instances in real dataset 3 (TSPLIB).

Instances	OPT	UB	Our algorithm				
			Best.Sol	*Aver.Sol*	gap_1[%1]	gap_2[%1]	*T*
dantzig42	12 528	12 650	12 528	12 528	0	0.96	0.57
att48	209 320	25 315	209 320	209 320	0	17.31	1.45
eil51	10 178	10 593	10 178	10 178	0	3.92	1.60
berlin52	143 721	15 209	143 721	143 721	0	5.51	1.56
st70	20 557	25 809	20 557	20 557	0	20.35	2.43
KroA100	983 128	10 912	983 128	983 128	0	9.91	8.23
KroB100	983 128	10 212	983 128	983 128	0	3.73	8.10
KroC100	961 324	11 013	961 324	961 324	0	12.71	8.28
KroD100	976 965	10 253	976 965	976 965	0	4.72	8.19
Aver					0	8.79	4.49

TABLE A.10. The experimental results of our algorithm for real dataset 2.

Jam factor	RTR		Our algorithm			
	Best.Sol	*T*	*Best.Sol*	*Aver.Sol*	gap_2[%]	*T*
1.00	118293.524	130	118293.524	118293.524	0.00	32.02
1.03	118796.154	259	118749.356	118749.356	0.06	37.59
1.04	119971.191	164	118901.300	118901.300	1.39	32.66
1.05	119503.279	197	119053.244	119053.244	0.70	36.79
1.10	119957.387	173	119957.387	119957.387	0.00	39.30
1.20	120637.093	217	119714.065	119714.065	0.77	43.13
1.30	120637.093	190	120637.093	120637.093	0.00	39.63
1.50	120617.178	253	120617.178	120617.178	0.00	41.61
1.60	121108.329	273	120679.186	120679.186	0.35	30.51
1.80	121195.816	182	120894.074	120894.074	0.25	32.77
1.90	121148.519	209	121001.518	121001.518	0.12	36.25
3.00	122222.204	260	121125.195	121125.195	0.90	32.65
10.00	121167.051	220	121125.195	121125.195	0.03	34.74
2000.00	121417.575	230	121125.195	121125.195	0.24	37.08
Aver		211			0.34	36.19

Acknowledgements. This work was supported by the project "An efficiency Metaheuristic Algorithm for the Time-Dependent Traveling Salesman Problem" funded by Hanoi University of Science and Technology (HUST) under grant number T2017-LN-14.

REFERENCES

[1] A. Archer, A. Levin and D. Williamson, A faster, better approximation algorithm for the minimum latency problem. *J. SIAM* **37** (2007) 1472–1498.

[2] H.G. Abeledo, G. Fukasawa, R. Pessoa and A. Uchoa, The time dependent traveling salesman problem: polyhedra and branch-cut-and price algorithm. *J. Math. Prog. Comput.* **5** (2013) 27–55.

[3] S. Arora and G. Karakostas, Approximation schemes for minimum latency problems. *Proc. STOC* (1999) 688–693.

[4] H.B. Ban and D.N. Nguyen, Improved genetic algorithm for minimum latency problem. *Proc. SOICT* (2010) 9–15.

[5] H.B. Ban, K. Nguyen, M.C. Ngo and D.N. Nguyen, An efficient exact algorithm for Minimum Latency Problem. *Progr. Inform.* **10** (2013) 167–174.

[6] N. Balakrishnan, A. Lucena and R.T. Wong, Scheduling examinations to reduce second-order conflicts. *Comput. Oper. Res.* **19** (1992) 353–361.

[7] A. Blum, P. Chalasani, D. Coppersmith, W. Pulleyblank, P. Raghavan and M. Sudan, The minimum latency problem. *Proc. STOC* (1994) 163–171.

[8] L. Bigras, M. Gamache and G. Savard, The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup time. *J. Dis. Optim.* **5** (2008) 685–699.

[9] K. Chaudhuri, B. Goldfrey, S. Rao and K. Talwar, Path, tree and minimum latency tour. *Proc. FOCS* (2003) 36–45.

[10] J.C. Picard and M. Queyranne, The Time-dependent traveling salesman problem and its application to the tardiness problem in one machin scheduling. *J. Oper. Res.* **26** (1978) 86–110.

[11] T.A. Feo and M.G.C. Resende, Greedy randomized adaptive search procedures. *J. Global Opt.* **6** (1995) 109–133.

[12] P. Festa and G.M.C. Resende, Hybridizations of GRASP with path-relinking, In Vol. 434 of Hybrid Metaheuristics - Studies in Computational Intelligence, edited by E.-G. Talbi. Springer, Berlin, Heidelberg (2013) 135–155.

[13] K. Fox, *Production scheduling on parallel lines with dependencies*. Ph.D. thesis, Johns Hopkins University, Baltimore, MD (1973).

[14] K. Fox, B. Gavish and S. Graves, The Time Dependent Traveling Salesman Problem and Extensions. Working paper No. 7904, Graduate School of Management, University of Rochester, NY (1979).

[15] M. Goemans and J. Kleinberg, An improved approximation ratio for the minimum latency problem. *Proc. SIAM SODA* (1996) 152–158.

[16] F. Glover, Tabu search. *J. INFORMS* **2** (1990) 4–32.

[17] M.A. Figliozzi, The time dependent vehicle routing problem with time windows: benchmark problems, an efficient solution algorithm, and solution characteristics. *J. Transp. Res. Part E Logist. Transp. Rev.* **48** (2012) 616–636.

[18] H. Hashimoto, M. Yagiura and T. Ibaraki, An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *J. Dis. Optim.* **5** (2008) 434–456.

[19] J.J. Dongarra, Performance of various computers using standard linear equations software. Linpack Benchmark Report, University of Tennessee Computer Science Technical Report, CS-89-85 (2013).

[20] F. Li, B. Golden and E. Wasil, Solving the time dependent traveling salesman problem. The Next Wave in Computing, Optimization, and Decision Technologies. Springer (2005) 163–182.

[21] A. Lucena, Time-dependent traveling salesman problem – the deliveryman case. *J. Networks* **20** (1990) 753–763.

[22] C. Malandraki and M. Daskin, Time dependent vehicle routing problems: formulations, properties and heuristic algorithms. *J. Transp. Sci.* **26** (1992) 185–200.

[23] N. Mladenovic and P. Hansen, Variable neighborhood search. *J. Oper. Res.* **24** (1997) 1097–1100.

[24] A. Orda and R. Rom, Shortest-path and minimumdelay algorithms in networks with time-dependent edgelength. *J. ACM* **37** (1990) 607–625.

[25] A. Salehipour, K. Sorensen, P. Goos and O. Braysy, Efficient GRASP+VND and GRASP+VNS meta-heuristics for the traveling repairman problem. *J. Oper. Res.* **9** (2011) 189–209.

[26] M. Silva, A. Subramanian, T. Vidal and L. Ochi, A simple and effective metaheuristic for the Minimum Latency Problem. *J. Oper. Res.* **221** (2012) 513–520.

[27] D. Simchi-Levi and O. Berman, Minimizing the total flow time of n jobs on a network. *IEEE. Trans.* **23** (1991) 236–244.

[28] L. Testa, A. Esterline and G. Dozier, Evolving efficient theme park tours. *J. Comput. Inform. Technol.* **7** (1999) 77–92.

[29] U. Ritzinger and J. Puchinger, Hybrid metaheuristics for dynamic and stochastic vehicle routing, In Vol. 434 of Hybrid Metaheuristics – Studies in Computational Intelligence. Edited by E.G. Talbi. Springer, Berlin, Heidelberg (2003) 77–95.

[30] K. Ruland, Polyhedral solution to the pickup and delivery problem. Ph.D. thesis, Washington University, Saint Louis, MO (1995).

[31] R. Vander Wiel and N. Sahinidis, An exact solution approach for the time-dependent traveling salesman problem. *Naval Res. Logistics (NRL)* **43** (1996) 797–820.

[32] R.J. Vander Wiel and N.V. Sahinidis, Heuristics bounds and test problem generation for the time-dependent traveling salesman problem. *J. Transp. Sci.* **29** (1995) 167–183.

[33] J. Bentner, G. Bauer, G.M. Obermai, I. Morgensten and J. Schneider, Optimization of the time-dependent traveling salesman problem with Monte Carlo methods. *Phys. Rev. E* **64** (2001) 36701-1–36701-8

[34] D.S. Johnson and L.A. McGeoch, The traveling salesman problem: a case study in local optimization. *Local Search Comb. Optim.* **1** (1997) 215–310.

[35] B.Y. Wu, Z.-N. Huang and F.-J. Zhan, Exact algorithms for the minimum latency problem. *Inf. Proc. Lett.* **92** (2004) 303–309.

[36] https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

[37] http://liris.cnrs.fr/christine.solnon/TDTSP.html.