4OR-Q J Oper Res (2013) 11:57–73
DOI 10.1007/s10288-012-0212-1

RESEARCH PAPER

Variable neighborhood search for the travelling deliveryman problem

Nenad Mladenović · Dragan Urošević · Saïd Hanafi

Received: 4 April 2012 / Revised: 24 July 2012 / Published online: 18 September 2012

Abstract A travelling deliveryman needs to find a tour such that the total waiting time of all the customers he has to visit is minimum. The deliveryman starts his tour at a depot, travelling at constant velocity. In this paper we suggest a general variable neighborhood search based heuristic to solve this NP-hard combinatorial optimization problem. We combine several classical neighborhood structures and design data structure to store and update the incumbent solution efficiently. In this way, we are able to explore neighborhoods as efficiently as when solving the travelling salesman problem. Computational results obtained on usual test instances show that our approach outperforms recent heuristics from the literature.

Keywords Combinatorial optimization · Routing · Travelling deliveryman problem · Metaheuristics · Variable neighborhood search

Mathematics Subject Classification (2000) 90C059 · 90C27

N. Mladenović
School of Mathematics, Brunel University, Uxbridge, UK
e-mail: Nenad.Mladenovic@brunel.ac.uk

D. Urošević (✉)
Mathematical Institute, Belgrade, Serbia
e-mail: draganu@mi.sanu.ac.rs

S. Hanafi
LAMIH-Université de Valenciennes, Valenciennes, France
e-mail: said.hanafi@univ-valenciennes.fr

1 Introduction

The travelling deliveryman problem (TDP), also known as the minimum latency problem or travelling repairman problem, is a variant of the travelling salesman problem (TSP). Differently from the classical TSP, where the travelling salesman is trying to minimize the length of the tour connecting all customers, in TDP the designed tour should minimize the total time customers are waiting to be served by a deliveryman (or a repairman).

On metric space, TDP is NP-hard (Blum et al. 1994). However, there are special graphs where TDP can be solved in polynomial time. Such graph classes are paths and trees with the diameters equal to 3 (Blum et al. 1994). Regarding exact methods for general graphs, Wu et al. (2004) suggested two methods: (i) Dynamic Programming and (ii) Branch and Bound (B&B). Using B&B method for example, authors of this article were able to solve exactly 50-customer instances in less than 100 seconds. It appeared to be significantly less efficient than the B&B of solver CPLEX using the MIP formulation described below. Méndez-Díaz et al. (2008) propose a new integer linear programming model for Minimum Latency Problem and a new B&B method for solving it. Experimental results show that proposed method allow them to solve exactly instances with up to $n = 40$ customers. Their method is then compared with CPLEX solver. It is shown that the CPLEX execution time is significantly longer. Abeledo et al. (2010) propose branch–and–cut–and–price for solving TDP. The largest TSPLIB instance solved exactly had $n = 107$ customers.

Salehipour et al. (2011) have recently proposed a heuristic that combines greedy randomized adaptive search procedure (GRASP) and variable neighborhood descent (VND), the deterministic variant of variable neighborhood search (VNS). Their sequential VND uses the following 5 neighborhood structures: (i) swap adjacent move (1-opt); (ii) swap move; (iii) 2-opt move; (iv) Or-opt move and (v) drop/add move.

In this paper we propose a General VNS for solving TDP. We implement neighborhood structures widely used for similar combinatorial problems, but, for the first time, within General VNS framework. In order to perform search through the different neighborhoods in more efficient way, we use preprocessing and intelligent updating of the incumbent solution. Moreover, we show that the worst-case complexity of neighborhood moves that we use are the same as the complexity of those moves used in solving TSP. Computational results performed on usual test instances show that our heuristic outperforms recent heuristics, and thus, it can be considered as a new state-of-the-art heuristic for solving TDP.

The paper is organized as follows. In Sect. 2 we give combinatorial and mathematical programming formulations of the problem. Complexity results of different moves used within variable neighborhood descent are discussed in Sect. 3, while in Sect. 4 we give pseudo-codes of our implementation of GVNS. Sections 5 and 6 contain computational results and conclusions, respectively.

2 Problem formulation

Combinatorial formulation Let $G = (V, A)$ be a complete, directed and asymmetric graph, where $V = \{0, 1, \ldots, n\}$ is a set of nodes. Nodes labelled with numbers

1, 2, …, n are delivery nodes and node with label 0 is the depot. The distance $d(i, j)$ is associated with each arc $(i, j) \in A$. The TDP consists in determining the tour, i.e. Hamiltonian path on the graph G,

$$x = (x_0, x_1, x_2, x_3, \ldots, x_n),$$

starting at the depot node $x_0 = 0$, so as to minimize the total waiting time of all customers to be served. Assume that the deliveryman travels at constant velocity v. Then the time to reach a client i is equal to the ratio between the passed distance from the depot to the client i and his/her velocity v. The distance that the deliveryman passes to reach the client x_1 is equal to $\delta_1 = d(x_0, x_1)$. Similarly, to reach the client x_2 he/she needs to pass $\delta_2 = d(x_0, x_1) + d(x_1, x_2)$, generally to reach a client x_k, the travel distance is

$$\delta_k = d(x_0, x_1) + d(x_1, x_2) + \cdots + d(x_{k-1}, x_k).$$

Therefore, the corresponding time of reaching the customer x_k is as follows:

$$t_k = \frac{\delta_k}{v}.$$

Since the objective function represents the sum of all such obtained time (for all k), it can be formulated as

$$\sum_{k=1}^{n} t_k.$$

It is clear that the last expression may be formulated more simply as

$$\sum_{k=1}^{n} \frac{(n-k+1)d(x_{k-1}, x_k)}{v}.$$

Since v is constant, its value does not influence the optimal solution. Thus, the objective function to be minimized may be presented as

$$f(x) = \sum_{k=1}^{n} (n-k+1)d(x_{k-1}, x_k) \tag{1}$$

Note that the time needed to go back to the depot is not included in the objective function.

Mixed integer programming (MIP) formulation As for the TSP, the TDP can also be formulated in different ways. Fischetti et al. (1993) used the so-called flow MIP

formulation. The two sets of variables are defined as

$$X_{ij} = \begin{cases} 1, & \text{if deliveryman travels arc (i,j)} \\ 0, & \text{otherwise} \end{cases}$$

$$Y_{ij} = \begin{cases} 0, & \text{if arc } (i,\ j) \text{ is not used} \\ n-k, & \text{if } (i,\ j) \text{ is } k\text{-th arc on route} \end{cases}$$

Then the MIP formulation for the TDP is as follows:

$$\min \sum_{i=0}^{n} \sum_{j=0}^{n} d(i,\ j) Y_{ij} \tag{2}$$

subject to

$$\sum_{j=0, j\neq i}^{n} X_{ij} = 1 \qquad i = 0, 1, 2, \ldots, n \tag{3}$$

$$\sum_{i=0, i\neq j}^{n} X_{ij} = 1 \qquad j = 0, 1, 2, \ldots, n \tag{4}$$

$$\sum_{i=1}^{n} Y_{0i} = n \tag{5}$$

$$\sum_{i=0, i\neq k}^{n} Y_{ik} - \sum_{i=0, i\neq k}^{n} Y_{ki} = 1 \qquad k = 1, 2, 3, \ldots, n \tag{6}$$

$$Y_{i0} = 0 \qquad i = 1, 2, \ldots, n \tag{7}$$

$$Y_{0i} \leqslant n X_{0i} \qquad i = 1, 2, \ldots, n \tag{8}$$

$$Y_{ij} \leqslant (n-1) X_{ij} \qquad i, j = 1, 2, 3, \ldots, n; i \neq j \tag{9}$$

$$X_{ij} \in \{0, 1\} \qquad i, j = 0, 1, 2, \ldots, n; i \neq j \tag{10}$$

$$Y_{ij} \in \{0, \ldots, n\} \qquad i, j = 0, 1, 2, \ldots, n; i \neq j \tag{11}$$

Hence, in this formulation, there are $2n(n+1)$ variables and n^2+4n+3 constraints. Constraints (3) and (4) make a guarantee that each customer will be visited and left only once (i.e., it has one incoming and one outcoming arc). Constraints (5)–(6) are flow constraints. They assure that the final tour X has no subtours (tours that do not include all n clients). Constraints (7)–(9) make a logical relation between variables X and Y. For example, if $X_{ij} = 0$, then Y_{ij} should be 0 as well.

Lower bound Obviously, a minimum spanning tree (MST) of the corresponding weighted graph, may be used as a lower bounding procedure. Let us rank edges $e \in A$ of the MST in a nondecreasing order of their weights

$$d(e_1) \leqslant d(e_2) \leqslant \cdots \leqslant d(e_n).$$

Then the lower bound LB is given by

$$LB = \sum_{k=1}^{n} (n + 1 - k) d(e_k). \tag{12}$$

We will use this formula later, as Salehipour et al. (2011) to measure quality of heuristics, as the average percentage deviation from the LB.

3 Neighborhood structures for solving TDP

VNS is a metaheuristic for solving combinatorial and global optimization problems. Its basic idea is to change neighborhood in search for a better solution (Mladenović and Hansen 1997). Its main loop consists of 3 steps: (i) Shaking or perturbation of the incumbent solution x, (ii) local search and (iii) neighborhood change. For various VNS variants and their successful applications see recent surveys by Hansen et al. (2008, 2010).

Usual local search procedures used for solving TSP are 2-opt, Or-opt (or 1-2-3-insertion) and 3-opt. Their advantage is in the fact that the updating of the objective function may be obtained in constant time. However, this is not the case when solving TDP, at least if usual data structure is used and no preprocessing step is performed. We first show that the complexity of k-opt moves in such cases is $O(n^{k+1})$ (instead of $O(n^k)$ for solving classical TSP). Then we explain the steps necessary to be done, so as the data structure to be used in order to get the same complexity for TDP as it is for TSP, i.e., $O(n^k)$.

k-opt complexity Let us consider the solution x of TDP

$$x = (x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_n)$$

The solution that belongs to 2-opt neighborhood of the solution x (in solving TSP) is obtained after deleting the links between x_i and x_{i+1}, and between x_j and x_{j+1}. The new tour x' and its objective function value $f(x')$ are:

$$x' = (x_1, x_2, \ldots, x_{i-1}, x_i, x_j, x_{j-1}, \ldots, x_{i+2}, x_{i+1}, x_{j+1}, \ldots, x_n), \tag{13}$$

$$\Delta f = f(x') - f(x) = d(x_i, x_j) + d(x_{i+1}, x_{j+1}) - d(x_i, x_{i+1}) - d(x_j, x_{j+1}). \tag{14}$$

Clearly, $f(x')$ is obtained after 4 additions, thus, in constant time. Unfortunately, this property does not hold when solving TDP by 2-opt. In a straightforward implementation of 2-opt for TDP, all coefficients between x_i and x_j in formula (1) should be changed in calculating $f(x')$. Therefore, the difference Δf may be obtained in $O(j - i)$ calculations. In the worst case $j = n$ and $i = 1$, and thus Δf is obtained in $O(n)$.

Observe also that this straightforward implementation of k-opt for solving TDP increases by n the complexity of k-opt move when TSP is considered, i.e., it is in

$O(n^{k+1})$. In this section, we will show that if an appropriate data structure is used, the worst-case complexity of k-opt in solving both TDP and TSP is the same: $O(n^k)$.

Our list of neighborhoods consists of (i) 1-opt, (ii) move forward (insertion to the right); (iii) move backward (insertion to the left); (iv) 2-opt. We did not put 3-opt move in our list because of its large complexity $O(n^3)$.

1-opt move is a special case of 2-opt, where the index j is set to $i+2$ in (13). Let us consider four consecutive clients in the current tour x:

$$x_k, x_{k+1}, x_{k+2}, x_{k+3}. \tag{15}$$

Then the new tour x' is obtained by swapping x_{k+1} and x_{k+2}.

Property 1 Exploring complete 1-opt neighborhood in solving TDP is in $O(n)$.

Proof From (1), (13) and (15) we see that the difference Δf between the new objective function and the old one is

$$\begin{aligned}\Delta f = &(n-k)(d(x_k, x_{k+2}) - d(x_k, x_{k+1})) \\ &+(n-k-2)(d(x_{k+1}, x_{k+3}) - d(x_{k+2}, x_{k+3}))\end{aligned}$$

If Δf is negative, then a better tour is obtained. Clearly, finding Δf is in $O(1)$. Since the cardinality of 1-opt neighborhood is $n-3$, the result follows. □

Move forward (mf) The solution from this neighborhood is defined by moving k consecutive customer positions to the right of the tour ($k = 1, 2, \ldots, \ell_{\max}$). If that block of k customers starts from the position $i+1$ and is inserted after the position j, (i.e., after customer x_j), then the tour

$$x = (x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_j, x_{j+1}, \ldots, x_n)$$

becomes

$$x' = (x_1, \ldots, x_i, x_{i+k+1}, \ldots, x_j, x_{i+1}, \ldots, x_{i+k}, x_{j+1}, \ldots, x_n)$$

Property 2 Exploring complete move-forward neighborhood in solving TDP is in $O(n^2 \ell_{\max})$.

Proof The difference between the new and the old objective functions is

$$\begin{aligned}\Delta f = &k\delta(x_{i+k+1}, x_j) - (j-i-k)\delta(x_{i+1}, x_{i+k}) \\ &+(n-i)(d(x_i, x_{i+k+1}) - d(x_i, x_{i+1})) \\ &+(n-j)(d(x_{i+k}, x_{j+1}) - d(x_j, x_{j+1})) \\ &+(n-j+k)d(x_j, x_{i+1}) - (n-i-k)d(x_{i+k}, x_{i+k+1})\end{aligned}$$

where $\delta(x_{i+k+1}, x_j)$ denotes a subtour distance from the customer x_{i+k+1} to the customer x_j. Similarly, $\delta(x_{i+1}, x_{i+k})$ denotes the distance from the client's position $i+1$

to the position $i + k$, i.e.

$$\delta(x_i, x_j) = \sum_{h=i}^{j-1} d(x_h, x_{h+1})$$

If i and j are fixed then, increase of k by 1 makes calculation of the new Δf in $O(1)$. The same holds for all (i, j) pairs. Therefore, finding objective function values in complete move-forward neighborhood is $O\left(\binom{n}{2}\ell_{\max}\right) = O(n^2\ell_{\max})$. □

Move backward (mb) This neighborhood structure is analogous to the previous one. The only difference is that the block of k consecutive customers is moved to the left.

These two neighborhoods together may be considered as one, i.e., $k - insertion$ neighborhood. Therefore, together, they may be seen as an extension of Or–opt move (note that in Or-opt $k = 1, 2$ and 3). However, we decided to split k-insertion neighborhood into its two disjoint subsets. This idea is shown to be useful in solving some other optimization problems on graphs (Brimberg et al. 2008, 2009).

2–opt move. We have already explained 2-opt move (see Eqs. (13) and (14)). Now we show that the 2-opt move may be done in $O(1)$ if appropriate data structure for storing and updating current incumbent tour is used.

Property 3 Exploring complete 2-opt neighborhood in solving TDP is in $O(n^2)$.

Proof Let the incumbent tour be presented as

$$x = (x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_n).$$

After deleting links (x_i, x_{i+1}) and (x_j, x_{j+1}), the new tour x' becomes

$$x' = (x_1, x_2, \ldots, x_{i-1}, x_i, x_j, x_{j-1}, \ldots, x_{i+2}, x_{i+1}, x_{j+1}, \ldots, x_n).$$

The corresponding objective function values are

$$\begin{aligned} f(x) = {} & (n-1)d(x_1, x_2) + \cdots + (n-i+1)d(x_{i-1}, x_i) + (n-i)d(x_i, x_{i+1}) \\ & +(n-i-1)d(x_{i+1}, x_{i+2}) + \cdots + (n-j+1)d(x_{j-1}, x_j) \\ & +(n-j)d(x_j, x_{j+1}) + (n-j-1)d(x_{j+1}, x_{j+2}) + \cdots + d(x_{n-1}, x_n) \end{aligned}$$

and

$$\begin{aligned} f(x') = {} & (n-1)d(x_1, x_2) + \cdots + (n-i+1)d(x_{i-1}, x_i) + (n-i)d(x_i, x_j) \\ & +(n-i-1)d(x_j, x_{j-1}) + \cdots + (n-j+1)d(x_{i+2}, x_{i+1}) \\ & +(n-j)d(x_{i+1}, x_{j+1}) + (n-j-1)d(x_{j+1}, x_{j+2}) + \cdots + d(x_{n-1}, x_n) \end{aligned}$$

After summing these two equations we get

$$f(x) + f(x') = 2\sum_{k=1}^{i-1}(n-k)d(x_k, x_{k+1}) + 2\sum_{k=j+1}^{n-1}(n-k)d(x_k, x_{k+1}) + (2n-i-j)\delta(x_{i+1}, x_{j-1}) + (n-i)(d(x_i, x_{i+1}) + d(x_i, x_j)) + (n-j)(d(x_{i+1}, x_{j+1}) + d(x_j, x_{j+1}))$$

By choosing the appropriate data structures and by preprocessing (before start searching through the neighborhood), the last formula can be calculated in $O(1)$ time. Moreover, at the same time ,we determine if x' is a better solution; since $f(x)$ is known, we can also find $f(x')$.

Though, the problem is how to calculate the above formula. So, before starting the search through the 2-opt neighborhood, we have to calculate and store two sequences $S_b(i)$ and $S_e(i)$ representing the following sums:

$$S_b(i) = \sum_{k=1}^{i}(n-k)d(x_k, x_{k+1}) \quad \text{and} \quad S_e(i) = \sum_{k=i}^{n-1}(n-k)d(x_k, x_{k+1})$$

so as

$$\delta(i) = \sum_{k=1}^{i} d(x_k, x_{k+1})$$

All can be done in $O(n)$ time. If we have these values, then

$$f(x) + f(x') = 2S_b(i-1) + 2S_e(j+1) + (2n-i-j)(\delta(j-2) - \delta(i)) + (n-i)(d(x_i, x_{i+1}) + d(x_i, x_j)) + (n-j)(d(x_{i+1}, x_{j+1}) + d(x_j, x_{j+1}))$$

obviously, this value is calculated in constant $O(1)$ time and thus, the complete 2-opt neighborhood is explored in $O(n^2)$. □

Once we find the best solution x' in the 2-opt neighborhood, we need to answer the question about the complexity of the updating move. However, it requires $O(n)$ time as it is included in complexity of searching the 2-opt neighborhood. We now give pseudo-code of the updating step.

Algorithm 1: Updating within 2-opt move

Function `UpdateSbSe` (x, i, j);
for $k \leftarrow i-1$ **to** *n-1* **do**
  $S_b(k) \leftarrow S_b(k-1) + (n-k) \times d(x_k, x_{k+1})$
for $k \leftarrow j$ **downto** *1* **do**
  $S_e(k) \leftarrow S_e(k+1) + (n-k) \times d(x_k, x_{k+1})$

This algorithm will be later used in 2–opt local search $N_{2-opt}(R, x', r)$, within Sequential VND(`Seq-VND`) and Mix VND (`Mix-VND`).

Double bridge (db) is a special case of 4–opt. For any four indices i, j, k and ℓ ($i < j < k < \ell$), edges (x_i, x_{i+1}), (x_j, x_{j+1}), (x_k, x_{k+1}), $(x_\ell, x_{\ell+1})$ are removed. New 4 edges are then (x_i, x_{k+1}), (x_ℓ, x_{j+1}), (x_k, x_{i+1}) and $(x_j, x_{\ell+1})$. It is very important for some combinatorial problems since it keeps the orientation of the tour. The neighborhood size of double bridge is $O(n^4)$. However, we reduce it to $O(n^2)$ by fixing indices $j = i + 2$ and $\ell = k + 2$. The double-bridge move is used only within our `GVNS-2`, which will be explained later in Algorithm 6.

Interchange (Exchg) When the two non-successive customers x_i and x_j ($i < j$), interchange their places in the tour, than the objective function value is obviously changed. The interchange move is a special case of the 4-opt as well. The size of the whole neighborhood is obviously $O(n^2)$. We use it just to compare different multi-start local search procedures with the multi-start VND (Table 1).

4 General VNS for solving TDP

General VNS (GVNS) is a variant of VNS where VND is used as a local search routine within basic VNS scheme Hansen et al. (2008, 2010). It contains 3 parameters: (i) t_{max}–maximum time allowed in a search; (ii) k_{max}–the number of neighborhoods used in shaking; (iii) ℓ_{max}–the number of neighborhoods used in VND. Its pseudo code is given in Algorithm 2.

Algorithm 2: Steps of the general VNS

```
Function GVNS (x, ℓ_max, k_max, t_max)
x ← Initial(x)
repeat
    k ← 1
    repeat
        x' ← Shake(x, k)
        x'' ← VND(x', ℓ_max)
        if f(x'') < f(x) then
            x ← x''; k ← 1
        else
            k ← k + 1
    until k > k_max
until t > t_max
return x.
```

The set of neighborhoods used both in shaking and VND local search are problem specific.

4.1 Initial solution

We use two ways to get an initial solution. The first is random one, i.e., it is obtained as a random permutation of $n - 1$ customers. The second is the Greedy randomized procedure given in Algorithm 3. We call it `Greedy-R`.

Algorithm 3: Greedy randomized algorithm for initial solution

Function `Greedy-R` (x, q)
$S \leftarrow \{1, 2, 3, \ldots, n\}$
$x_0 \leftarrow 0$; $k \leftarrow 1$
while $S \neq \emptyset$ **do**
  $S' \leftarrow$ subset of S with the $\min(q, n-k)$ customers nearest to x_{k-1}
  $x_k \leftarrow$ Random selected vertex from S'
  $S \leftarrow S \setminus \{x_k\}$
  $k \leftarrow k+1$
return x.

In each constructive step of our `Greedy-R`(x, q) a random selection of the next customer x_k is restricted to the q (a parameter) closest to the last selected customer x_{k-1}. If the number of customers n, or the number of remaining customers $n-k$ is less than q, then one among all remaining customers is chosen. In our implementation we always use a value of 10 for q (q=10).

4.2 Shaking

In the VNS shaking step, the incumbent solution x is perturbed to get one random solution x' from the k^{th} neighborhood of x ($x \in \mathcal{N}_k(x), k = 1, \ldots, k_{max}$). Such a random solution is an initial one for the local search routine that follows. After some experiments, we decided to use the insertion neighborhood structure for the sake of shaking. In other words, our shaking consists of selecting a customer i at random and inserting it between j and $j+1$, where j is also randomly chosen. In fact, we repeat k times this random insertion move.

Algorithm 4: Shaking procedure

Function `Shake`(x, k)
while $k > 0$ **do**
  $i \leftarrow$ `Random`$(1, n)$; //select i from $[1, n]$ at random
  $j \leftarrow$ `Random`$(1, n)$; // select i from $[1, n]$ at random
  if $i \neq j$ **then**
    Insert x_i between x_j and x_{j+1}
    $k \leftarrow k-1$
return x

Note that insertion move consists of move forward or move backward moves. Therefore our shaking can be move forward or move backward if $j \geq i$ or $j \leq i$, respectively. Note also that one-insertion move is a special case of Or-opt move where one, two or three consecutive vertices are inserted.

4.3 VND algorithms

Finding the right order of neighborhoods used in deterministic VND heuristic may be very important for the quality of the final solution. After extensive testing of different

variants, we decided to use the following order of neighborhoods: (i) 1-opt, (ii) 2-opt, (iii) move-forward, (iv) move-backward. In `Mix-VND`, we additionally use (v) double-bridge neighborhood structure. The number of visited solutions from 2-opt neighborhood is also reduced: for fixed i (i.e., x_i), we take only r (a parameter) nearest customers as candidates for x_j and to perform the 2-opt move. If the total number of the customers is less than r, than we set $r = n/2$. Hence, we need to rank all distances (columns of matrix D) in pre-processing step to get ranked distances R and thus be able to easily find r closest customers to each customer x_i.

We develop two VND variants for solving TDP: `Seq-VND` (see Algorithm 5) and `Mix-VND` (see Algorithm 6). For details regarding Nested and Mixed VND methods see Ilić et al. (2010) and Mladenović et al. (2012).

Algorithm 5: Sequential VND for solving TDP

Function `Seq-VND` (R, x', r)
$\ell \leftarrow 1$
while $\ell \leqslant 4$ **do**
  if $\ell = 1$ **then**
    $x'' \leftarrow \text{argmin}\{f(x) | x \in N_{1-opt}(x')\}$
  if $\ell = 2$ **then**
    $x'' \leftarrow \text{argmin}\{f(x) | x \in N_{2-opt}(R, x', r)\}$
  if $\ell = 3$ **then**
    $x'' \leftarrow \text{argmin}\{f(x) | x \in N_{mf}(x')\}$
  if $\ell = 4$ **then**
    $x'' \leftarrow \text{argmin}\{f(x) | x \in N_{mb}(x')\}$
  if $f(x'') < f(x')$ **then**
    $x' \leftarrow x''; \ell \leftarrow 1$
  else
    $\ell \leftarrow \ell + 1$
return x'

In `Mix-VND` (see Mladenović et al. 2012) we perform `Seq-VND` starting from each solution that belong to N_{db} neighborhood of a current solution x. The N_{db} neighborhood is fully explored but in the first improvement strategy. If the solution x'' obtained with `Seq-VND` heuristic is better than the current solution x', we move there ($x' \leftarrow x''$).

Algorithm 6: Mixed Variable Neighborhood Descent for TDP

Function `Mix-VND` (R, x', r)
$improve \leftarrow$ `true`
while $improve$ **do**
  $improve \leftarrow$ `false`
  while there are unexplored solution in $N_{db}(x')$ and $.not.improve$ **do**
    $x'' \leftarrow$ next solution in $N_{db}(x')$
    $x'' \leftarrow$ `Seq-VND`(R, x'', r),
    if $f(x'') < f(x')$ **then**
      $x' \leftarrow x''$
      $improve \leftarrow$ `true`
return x'

The rationale of our Mix-VND procedure is to give double-bridge move a special status since it is known to be very important in solving the TSP problem (Johnson and McGeoch 1996).

4.4 GVNS algorithms

We designed here 2 GVNS variants for solving TDP. One uses sequential VND as a local search (GVNS-1 for short), and the other uses Mixed nested VND (GVNS-2). Correctness of these heuristics follows results from Markoski et al. (2011). We also implemented two variants to get initial solution: Greedy-R(x, 10) and Rand(x). Note that Rand represents a procedure for random permutation of n numbers and will not be presented here. The pseudo code of all our variants are given in Algorithm 7.

Algorithm 7: General VNS for TDP

Function GVNS (D, x, k_{max}, t_{max})
$R \leftarrow$ Rank(D)
$x \leftarrow$ Greedy-R(x, 10) (or $x \leftarrow$Rand(x))
repeat
  $k \leftarrow 1$
  repeat
    $x' \leftarrow$ Shake(x, k)
    $x'' \leftarrow$ Seq-VND($R, x', 4$) (or $x'' \leftarrow$ Mix-VND($R, x', 4$))
    if $f(x'') < f(x)$ **then**
      $x \leftarrow x''; k \leftarrow 1$
    else
      $k \leftarrow k + 1$
  until $k > k_{max}$
until $t > t_{max}$
return x.

In preprocessing step we rank distances among customers D in non-decreasing order of their values. A new ordered matrix R is used in reducing the set of visited points of 2-opt neighborhood. As explained earlier, in our implementation the number of neighborhoods ℓ_{max} used within VND for solving TDP is equal to 4. So, we do not need to consider ℓ_{max} in Algorithm 7 as a parameter, as it is presented in Algorithm 2.

5 Computational results

Our GVNS-1 and GVNS-2 are coded in C++. Computational analysis is performed on Intel Pentium IV processor with 1.8GHz, under Linux.

Test instances. According to Salehipour et al. (2011), test instances specifically created for TDP do not exist in the literature. That is why the authors in Salehipour et al. (2011) generated their own instances but also used the instances from the TSP library to test their GRASP+VNS based heuristics.

In Salehipour et al. (2011) 140 instances were generated in total with n equal to 10, 20, 50, 100, 200, 500 and 1000. For any value of n, 20 different instances were

Table 1 Comparison of different multistart local search procedures applied on instance with 200 customers

Loc. search	Random initial				Greedy initial			
	% of best	% of worst	% of avg.	Time	% of best	% of worst	% of avg.	Time
1-opt	690.97	913.99	790.97	0.01	73.13	207.00	135.56	0.01
2-opt	13.72	49.54	31.75	0.40	10.83	63.42	25.63	0.16
1-ins	9.14	53.55	27.95	0.22	9.65	43.95	23.32	0.08
2-ins	21.48	71.53	38.80	0.12	21.07	70.21	37.31	0.05
3-ins	32.86	78.89	51.30	0.09	31.46	76.69	48.38	0.04
Or-opt	7.41	45.93	23.02	0.24	5.63	17.63	18.55	0.09
Exchg	70.59	150.69	108.24	0.13	39.70	93.12	64.04	0.06
Seq-vnd	2.18	13.69	6.85	0.38	2.22	14.83	7.70	0.20

generated in the following way: coordinates of all the customers, including the depot, were generated at random with a uniform distribution on the interval [0,100]. For instances with 500 and 1000 customers this interval was [0,500]. The matrix $D = (d_{ij})$ was obtained by using floor function of the Euclidean distance between any two nodes i and j.

Stopping condition. We stop our VNS based heuristic after n seconds ($t_{max} = n$ seconds). In the tables below, we give report on the time when the best solution of each instance is reached.

Local search procedures. We perform extensive computational analysis to figure out the power of each neighborhood structure for solving TDP, as well as the power of our VND local search procedures. For that purposes we repeat local search, with respect to each neighborhood, 1000 times. Tests are performed on many instances. In order to save the space, here we give the results only for $n = 200$ in Table 1. The % deviations from the best know solution (obtained by our GVNS methods) are reported for the best, the worst and the average solutions obtained by each local search. Or-opt presents the local search that uses insertion of 1, then 2 and then 3 consecutive vertices. 1-ins, 2-ins and 3-ins denote local search heuristics that inserts 1, 2 or 3 consecutive vertices respectively.

Different local search procedures' behavior is also presented in the so-called *distance-to-target* diagram given at Fig. 1. The same random instance with $n = 200$ is used, with two initialization methods: random (the left part of pictures) and Randomized greedy (on the right). The best known solution of that instance is given in origin; point (u, v) in the diagram represents a local optima obtained by Greedy-R. u and v coordinates represent the Hamming distance (between the best known solution and the obtained local minima) and the % deviation of local optima value from the best objective function value, respectively.

From Table 1, Fig. 1, we can draw the following conclusions: (i) The multi-start Sequential VND local search performs the best; (ii) It is interesting to note that the Greedy-R initialization does not improve the performance of the multi-start Seq-VND. In other words, GRASP+VND does not produce local minima of better qualities than

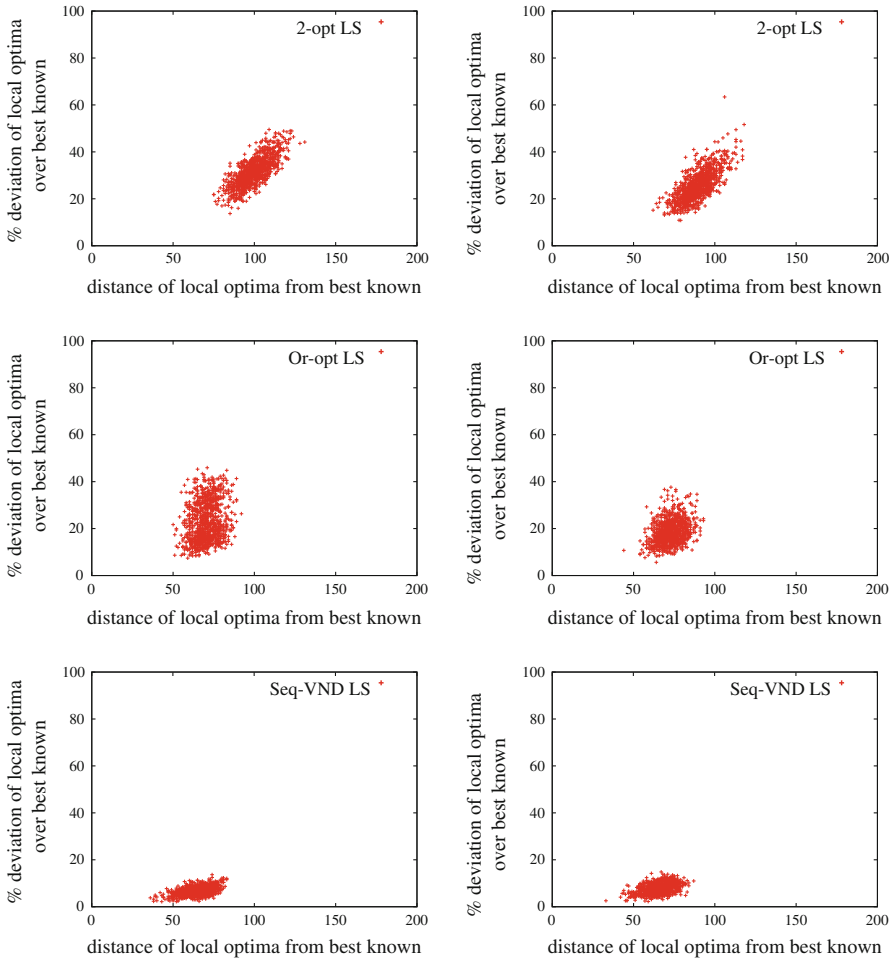

Fig. 1 Distribution of local optima in the distance-to-target diagrams; comparison of random multistart (*left*) and GRASP (*right*) local searches w.r.t. 2-opt, Or-opt and Seq-VND neighborhoods on instance with n=200 customers

Multistart Seq-VND; (iii) Local minima do not depend on the initialization method but on the neighborhood structure used. The exception is Exchange neighborhood structure (which appears not to be good for solving TDP anyway).

Comparison. To measure the quality of heuristics the average percentage deviation from the lower bound LB (calculated as in (12)) is used in this section as in Salehipour et al. (2011). Average deviation is calculated on 20 instances, generated for the same n.

Table 2 contains comparative results of the four methods: GRASP + VND, GRASP + VNS, GVNS-1 and GVNS-2. For all 4 methods the average deviation from the lower bound is reported so as their CPU time. The columns 8 and 11 give % improvement of our GVNS-1 and GVNS-2 suggested in this paper over GRASP-VNS that could be seen as the current state-of-the-art heuristic.

Table 2 Comparison on random instances

n	GRASP–VND		GRASP–VNS		GVNS-1			GVNS-2		
	% dev	Time	% dev	Time	% dev	Time	Impr.	% dev	Time	Impr.
10	33.04	0.00	33.04	0.00	33.04	0.07	0.00	33.04	0.23	0.00
20	39.63	0.00	40.34	0.01	39.21	0.41	2.88	39.29	0.87	2.67
50	45.30	0.00	47.20	0.08	43.90	2.41	7.52	43.97	4.36	7.35
100	43.26	1.11	44.28	2.95	40.79	25.86	8.56	40.82	38.41	8.48
200	43.16	3.75	38.77	68.03	37.95	124.32	1.11	38.14	93.80	0.60
500	46.11	604.89	49.50	1524.09	39.41	414.05	25.60	39.54	421.72	25.19
1000	45.99	3876.51	42.35	9311.21	41.51	942.47	2.02	41.28	921.52	2.59

Table 3 Comparison on TSP-Lib instances

Name	GRASP–VND		GVNS-1			GVNS-2		
	Obj fun	Time	Obj fun	Time	Impr.	Obj fun	Time	Impr.
st70	19533	22.24	19215	7.6	1.63	19215	26.15	1.63
att532	18448435	65121.06	17610285	488.95	4.54	17584979	532.02	4.68
kroD100	976830	54.69	949594	16.42	2.79	949594	46.55	2.79
lin105	585823	101.27	585823	86.76	0.00	585823	76.17	0.00
lin318	5876537	3951.02	5612165	179.13	4.50	5601351	294.85	4.68
pr107	1983475	225.32	1980767	97.51	0.14	1981141	49.94	0.12
pr226	7226554	2328.71	7106546	144.29	1.66	7100708	37.57	1.74
pr439	18567170	4563.44	17790562	327.15	4.18	17799975	396.84	4.13
rat99	56994	195.31	54984	42.41	3.53	55028	50.59	3.45
rat195	213371	884.78	210193	111.2	1.49	210319	150.94	1.43

Table 3 contains the same information as Table 2, but on the instances from the TSP library. In addition, instead of % deviation above lower bound, the values of objective functions are reported. They are obtained by a single run of the each method.

It appears that:

(i) `GVNS-1` and `GVNS-2` outperform both `GRASP+VND` and `GRASP+VNS`;
(ii) There is no significant difference between the `GVNS-1` and the `GVNS-2`. Nevertheless `GVNS-2` seems to behave better for larger instances, while for the small and middle size instances, it is opposite.

In Table 4 the results obtained by branch–and–cut–and–price method (B&C&P) proposed by Abeledo et al. (2010) and by our GVNS-1 are compared. Note that in this case objective function is slightly different: the first edge on tour is multiplied by $n + 1$, the second with n, etc. Finally, the edge between the last customer on tour and the depot is multiplied by 1. Values in boldface are proved to be optimal. It appears that our GVNS-1 reaches all best optimal and three best known values with much smaller CPU time.

Table 4 Comparison of the results obtained by branch–and–cut–and–price method proposed by Abeledo et al. (2010) and by our GVNS-1

Instance	B&C&P		GVNS-1	
	Obj.val.	Time	Obj.val.	Time
dantzig42	**12528**	28	**12528**	0.25
swiss42	**22327**	20	**22327**	0.09
att48	**209320**	103	**209320**	0.07
gr48	**102378**	76	**102378**	0.12
hk48	**247926**	124	**247926**	0.98
eil51	**10178**	33	**10178**	1.98
berlin52	1**43721**	104	**143721**	1.38
brazil58	**512361**	633	**512361**	6.16
st70	**20557**	2895	**20557**	2.68
eil76	**17976**	223	**17976**	1.36
pr76	3**455242**	58045	**3455242**	27.77
gr96	**2097170**	160250	**2097170**	2.08
rat99	57986	>48hs	57986	23.99
kroA100	**983128**	106336	**983128**	52.05
kroB100	**986008**	7684	**986008**	29.76
kroC100	**961324**	39559	**961324**	1.94
kroD100	976965	>48hs	976965	6.97
kroE100	**971266**	117965	**971266**	2.90
rd100	**340047**	18582	**340047**	63.82
eil101	27513	>48hs	27513	61.91
lin105	**603910**	6317	**603910**	12.50
pr107	**2026626**	9947	**2026626**	4.23

6 Conclusions

In this paper we propose General variable neighborhood search (GVNS) based heuristic for solving Travelling deliveryman problem (TDP). Neighborhoods that we use are the classical ones explored for solving Travelling salesman problem. We show how they can be efficiently adapted for solving TDP as well. Moreover, we calculated the complexity of such adapted moves. Extensive computational results show that our two GVNS versions outperforms recent heuristic suggested by Salehipour et al. (2011). Future research may contain extension of the methodology and data structures to another similar combinatorial optimization problems. Also, the clustering of customers on the network (as in Carrizosa et al. 2011) may be tried out before routing.

Acknowledgments The present research work has been supported by Research Grants 174010 and III 044006 of the Serbian Ministry of Education, Science and Technological Development. The present research work has been also supported by International Campus on Safety and Intermodality in Transportation the Nord-Pas-de-Calais Region, the European Community, the Regional Delegation for Research and Technol-

ogy, the Ministry of Higher Education and Research, and the National Center for Scientific Research. The authors gratefully acknowledge the support of these institutions.

References

Abeledo H, Fukasawa R, Pessoa A, Uchoa E (2010) The time dependent traveling salesman problem: polyhedra and algorithm. Optim Online (http://www.optimization-online.org/DB_HTML/2010/12/2872.html)

Blum A, Chalasani P, Coppersmish D, Pulleyblank B, Raguavan P, Sudan M (1994) The minumum latency problem. In: Proceedings of 26th annual ACM symposium on theory of computting, Montreal

Brimberg J, Mladenović N, Urošević D (2008) Local and variable neighborhood search for the k-cardinality subgraph problem. J Heuristics 14:501–517

Brimberg J, Mladenović N, Urošević D, Ngai E (2009) Variable neighborhood search for the heaviest k-subgraph. Comput Oper Res 36:2885–2891

Carrizosa E, Mladenović N, Todosijević R (2011) Sum-of-square clustering on networks. Yugosl J Oper Res 21(2):157–161

Fischetti M, Laporte G, Martello S (1993) The delivery man problem and cumulative matroids. Oper Res 41(6):1055–1064

Hansen P, Mldenović N, Moreno-Pérez JA (2008) Variable neighbourhood search: methods and applications. 4OR 6:319–360

Hansen P, Mladenović N, Moreno Pérez JA (2010) Variable neighbourhood search: algorithms and applications. Ann Oper Res 175:367–407

Ibm, ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer

Ilić A, Urošević D, Brimberg J, Mladenović N (2010) Variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. Eur J Oper Res 206:289–300

Johnson DS, McGeoch LA (1996) The travelling salesman problem: a case study in local optimization. In: Aarts EHL, Lenstra JK (eds) Local search in combinatorial optimization. Wiley, New York

Markoski B, Hotomski P, Malbaški D, Obradović D (2011) Dijkstra's interpretation of the approach to solving a problem of program correctness. Yugosl J Oper Res 20:229–236

Méndez-Díaz I, Zabala P, Lucena A (2008) A new formulation for the traveling deliveryman problem. Discret Appl Math 156:3223–3237

Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24:1097–1100

Mladenović N, Urošević D, Hanafi S, Ilić A (2012) A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. Eur J Oper Res 220:270–285

Mladenović N, Urošević D, Todosijević R. An efficient GVNS for solving traveling salesman problem with time windows. Yugosl J Oper Res. doi:10.2298/YJOR120530015M

Salehipour A, Sörensen K, Goos P, Brysy O (2011) Efficient GRASP+VND and GRASP+VNS metaheuristics for the traveling repairman problem. 4OR-Q J Oper Res 9(2):189–209

Wu BY, Huang Z, Zhan F (2004) Exact algorithms for the minimum latency problem. Inf Process Lett 92(6):303–309