Optim Lett (2017) 11:1047–1056
DOI 10.1007/s11590-014-0788-9

ORIGINAL PAPER

A general variable neighborhood search variants for the travelling salesman problem with draft limits

Raca Todosijević · Anis Mjirda · Marko Mladenović · Saïd Hanafi · Bernard Gendron

Received: 13 February 2014 / Accepted: 23 August 2014 / Published online: 3 September 2014

Abstract In this paper, we present two general variable neighborhood search (GVNS) based variants for solving the traveling salesman problem with draft limits (TSPDL), a recent extension of the traveling salesman problem. TSPDL arises in the context of maritime transportation. It consists of finding optimal Hamiltonian tour for a given ship which has to visit and deliver products to a set of ports while respecting the draft limit constraints. The proposed methods combine ideas in sequential variable neighborhood descent within GVNS. They are tested on a set of benchmarks from the literature as well as on a new one generated by us. Computational experiments show remarkable efficiency and effectiveness of our new approach. Moreover, new set of benchmarks instances is generated.

Keywords Maritime transportation · Traveling salesman problem with draft limits · Sequential variable neighborhood descent · General variable neighborhood search

R. Todosijević · A. Mjirda (✉) · M. Mladenović · S. Hanafi
LAMIH-UVHC, Le Mont Houy Valenciennes, Cedex 9, France
e-mail: anis.mjirda@univ-valenciennes.fr

R. Todosijević
e-mail: raca.todosijevic@univ-valenciennes.fr

M. Mladenović
e-mail: marko.mladenovic@univ-valenciennes.fr

S. Hanafi
e-mail: said.hanafi@univ-valenciennes.fr

B. Gendron
CIRRELT, Université de Montréal, Québec, Canada
e-mail: bernard.gendron@cirrelt.ca

1 Introduction

Traveling salesman problem (TSP) is well-studied problem in the literature in terms of solution approaches and its variants [7–11,14]. Recently, new variant of TSP in the context of maritime transportation, called traveling salesman problem with draft limits (TSPDL) has been proposed [12]. TSPDL consists of visiting and delivering goods for a set of ports using a ship located initially at a depot. Since each port has a delivery demand known in advance the ship starts the tour with a load equal to the total demand, visits each ports exactly once and comes back to the depot performing the lowest cost tour. However, to each port it is assigned a draft limit, which represent the maximal allowed load on the ship upon entering some port.

Formally, the problem may be stated as follows [12]. Given an undirected graph $G = (V, E)$ where $V = \{0, 1, \ldots, n\}$ represents the set of ports including the starting port, i.e. the depot denoted by 0, while $E = \{(i, j)|i, j \in V, i \neq j\}$ represents the edge set where each edge (i, j) from the set E has the associated cost c_{ij}. For each port i, apart from the depot, a draft limit, l_i, and a demand, d_i, are given. Therefore the goal of TSPDL is to design minimal cost Hamiltonian tours, visiting each port exactly once and respecting draft limit constraints of all ports. Additionally, let us denote by L_i the load on the ship upon entering the port i, calculated relatively to a given tour T. The tour T will be called a feasible tour if $L_i \leq l_i$ for all $i \in V$, otherwise it will be called an infeasible tour.

To each arc $(i, j) \in E$, there are assigned two variables x_{ij} and y_{ij} which represent respectively, whether edge (i, j) is included in a tour or not, and the load onboard the ship on arc (i, j). TSPDL can be modeled as follows.

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{1}$$

$$\text{s.t.} \sum_{i\in V} x_{ij} = 1 \quad \forall j \in V \tag{2}$$

$$\sum_{j\in V} x_{ij} = 1 \quad \forall i \in V \tag{3}$$

$$\sum_{i\in V} y_{ij} - \sum_{i\in V} y_{ji} = d_j \quad \forall j \in V\backslash\{0\} \tag{4}$$

$$\sum_{i\in V} y_{0i} = \sum_{i\in V\backslash\{0\}} d_i \tag{5}$$

$$\sum_{i\in V} y_{i0} = 0 \tag{6}$$

$$0 \leq y_{ij} \leq l_j x_{ij} \quad \forall (i, j) \in E \tag{7}$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in E \tag{8}$$

Constraints (2) and (3) ensure that each vertex is visited exactly once. Constraints (4) guarantee that the demand of each port will be satisfied and also prevent the creation of subtours. Constraints (5) and (6) mean that the ship leaves the depot fully loaded

and returns to it empty. Finally, constraints (7) enforce the ship to respect draft limits of ports.

TSPDL is a NP-hard problem [12]. It was introduced, as we already mentioned, in Rakke et al. [12]. In that paper they gave two formulations of TSPDL and proposed branch and cut algorithm for solving both of them. Additionally, they introduced some valid inequalities and strengthened bounds that significantly reduce both the number of branch-and-bound nodes and the solution times. However, with these approaches they did not succeed to solve all instances to optimality. Recently, Battarra et al. [1] proposed three new formulations of TSPDL which enabled them to solve instances to optimality using exact algorithms.

Up to now, just exact methods have been considered for solving this problem. However, it turns out that these methods consume a lot of CPU time to solve benchmark instances either to optimality or near optimality. Therefore, there is a need for metaheuristics or heuristic able to find near-optimal solutions quickly and also able to solve instances of larger size. For that purpose in this paper, we propose two general variable neighborhood search methods.

In Sect. 2, we present the proposed two general variable neighborhood search metaheuristics. Computational results and a comparative analysis are given in Sect. 3 followed by some concluding remarks in Sect. 4.

2 General variable neighborhood search for TSPDL

General variable neighborhood search (GVNS) is a metaheuristic based on the systematic exploration of different neighborhood structures through a variable neighborhood descent (VND) as a local search routine. GVNS is composed by a local search phase in order to find a local optimum and a shaking phase to escape from local optimum trap [6].

In order to solve the TSP with draft limits, we used a GVNS metaheuristic. In this section, we first explain how we generate an initial solution for this problem. Then, we present the set of neighborhood structures used in the VND algorithm. Finally, we define the shaking phase and we present the different steps of our GVNS.

2.1 Initial solution

From the problem formulation it is obvious that not every solution of TSP, i.e. permutation of ports, is feasible for TSPDL and therefore some permutations might be infeasible. However, the following proposition, proved in [12], provides a necessary and sufficient condition for feasibility.

Proposition 1 *Let us suppose, without loss of generality, that all ports are sorted in non ascending order regarding draft limits*

$$l_i \leq l_{i+1}, \forall i \in \{1, \ldots n-1\}.$$

If the solution $X = (0, \ldots, n)$ *is infeasible, than there is no feasible solution of the problem.*

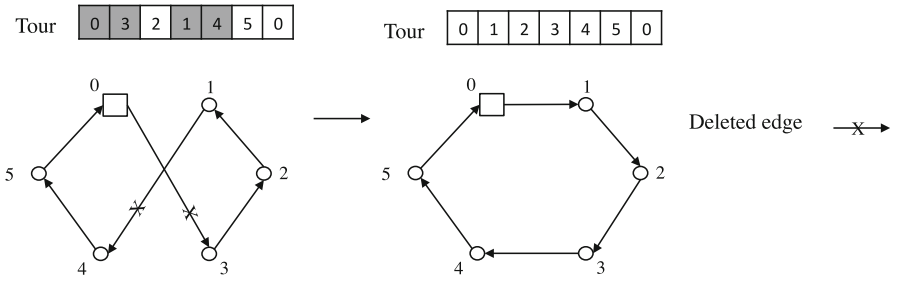

Fig. 1 Illustration of 2-opt move

This proposition provides a way how to generate a first feasible solution if any exist. More precisely, to do this we have to sort the ports in non increasing order of their draft limit values, and take this permutation as an initial solution.

2.2 Neighborhood structures and feasibility checking

Since we represent a solution of TSPDL as a permutation of ports, any neighborhood structure applicable for TSP can be also adapted for TSPDL problem taking into account the draft limit constraints. This adaption may be performed in a simple way, by excluding all permutations which are infeasible for TSPDL. In order to quickly recognize infeasible solutions for each used neighborhood structures, we implement the feasibility checking function. In what follows, we describe the used neighborhood structures and the feasibility checking procedures used within them.

Most common moves performed on a TSP solution are 2-opt moves and OR-opt moves [4]. The 2-opt move breaks down two edges of a current solution, and inserts two new edges by inverting the part of a solution in such a way that the resulting solution is still a tour. More formally, we denote by $\pi = (0, \pi_1, \ldots, \pi_i, \ldots, \pi_j, \ldots, \pi_n)$ the current solution. The solution obtained after applying 2-opt move can be $\pi' = (0, \pi_1, \ldots, \pi_i, \pi_j, \ldots, \pi_{i+1}, \pi_{j+1}, \ldots, \pi_n)$. Figure 1 illustrate the 2-opt move for a permutation of 6 ports where edges $(0, 3)$ and $(1, 4)$ are breaking down and inverted.

One variant of the 2-opt move is the so-called 1-opt move which is applicable on four consecutive ports, i.e. i_1, i_2, i_3 and i_4, in such a way that edges (i_1, i_2) and (i_3, i_4) are broken down, and the edge (i_2, i_3) is inverted.

OR-opt move is a special case of 3-opt move. It inserts a chain of up to three consecutive customers between any other two tour neighboors elsewhere in the tour without inverting any part of a solution. Therefore, there are OR-opt-1, OR-opt-2 and OR-opt-3 moves. If a chain contains k ports, we call such move OR-opt-k move. If a chain of k consecutive ports is moved backward, that move will be called backward OR-opt-k. Similarly, if a chain is moved forward, the move will be called forward OR-opt-k. If $k = 2$, this move can be formally defined as: $\pi = (0, \pi_1, \ldots, \pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j, \pi_{j+1}, \pi_{j+2}, \ldots, \pi_n)$, then $\pi' = (0, \pi_1, \ldots, \pi_i, \pi_j, \pi_{j+1}, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_{j+2}, \ldots, \pi_n)$. Figure 2 illustrates this move of a chain containing 2 ports (port 3 and 4) forward and backward.

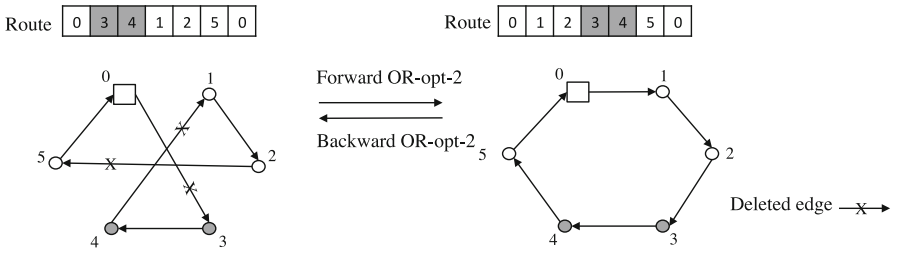

Fig. 2 Illustration of OR-opt-2 move

Each of these moves defines one neighborhood structure of the given TSP solution as a set of all solutions obtained by performing that move on the given TSP solution.

Previously described moves can be executed on each feasible TSPDL solution but we must be careful since some of them can lead into an infeasible solution. In order to recognize whether some move is feasible or infeasible, without performing it, we just have to keep array L updated and check whether the feasibility will be kept on some part of a tour if we execute that move. For example, let us suppose that we have the following tour $(0, i_1, i_2, \ldots, i_j, \ldots, i_k, \ldots, i_n, 0)$ and we want to perform a 2-opt move on it breaking down edges (i_j, i_{j+1}) and (i_k, i_{k+1}) which leads to the tour $(0, i_1, i_2, \ldots i_j, i_k, i_{k-1}, \ldots, i_{j+1}, i_{k+1}, \ldots i_n, 0)$. Comparing these two tours we may conclude that the order of visiting ports is unchanged in the first part of the tour starting at port 0 and ending at port i_j and therefore the feasibility will be kept on this part. Additionally, note that the load on board the ship for ports i_{k+1} to i_n is the same on both routes, thus the feasibility of that part will be preserved in the new tour. So, to check the feasibility of a new tour, one only has to check the feasibility on the inverted part of the tour. Let us suppose that we have the same tour as in the previous case but we want to perform forward OR-opt-k move which moves the chain of k ports, i.e. $i_j, \ldots i_{j+k-1}$, after port i_l resulting into a tour $(0, i_1, i_2, i_3, \ldots, i_{j-1}, i_{j+k}, \ldots, i_l, i_j, \ldots i_{j+k-1}, i_{l+1}, \ldots i_n)$. As in the previous case we conclude that feasibility will be kept on the first part of the tour, i.e. $(0, i_1, i_2, i_3, \ldots, i_{j-1})$ as well as on the final part, i.e. $(i_{l+1}, \ldots, i_n, 0)$. Further, comparing to the starting tour, in the new tour ports $(i_j, \ldots i_{j+k-1})$ will be visited later with smaller load onboard the ship and therefore the draft limits of these ports will not be violated. Hence, to check feasibility of the resulting tour it suffices to check whether draft limits of all ports $(i_{j+k}, \ldots i_l)$ will be respected if the move is executed. On the other hand, if we want to perform backward OR-opt-k move relocating the chain $(i_l, \ldots i_{l+k-1})$ backward after port i_j, to check feasibility of such move it is enough to check whether draft limits of ports $(i_l, \ldots i_{l+k-1})$ will be violated or not executing that move. Indeed, since the resulting tour will have a form $(0, i_1, i_2, i_3, \ldots i_j, i_l, \ldots i_{l+k-1}, i_{j+1}, \ldots i_{l-1}, i_{l+k}, \ldots i_n)$ all of the ports $(i_{j+1}, \ldots, i_{l-1})$ will be visited later with smaller load onboard the ship, in comparison to the starting tour, and therefore draft limits of these nodes will be respected. Additionally, feasibility of the first and the final part of the resulting tour, i.e. $(0, i_1, i_2, i_3, \ldots i_j)$ and $(i_{l+k}, i_{l+k+1} \ldots i_n)$, will be preserved for the same reasons as in the previous cases.

2.3 Sequential variable neighborhood descent

In this section, we will describe two variants of sequential variable neighborhood descent (SeqVND) which systematically explore different neighborhood structures using the first improvement strategy. The first SeqVND, i.e. SeqVND-1, examines respectively 1-opt ($\mathcal{N}_1$), 2-opt ($\mathcal{N}_2$), backward OR-opt-3 ($\mathcal{N}_3$), forward OR-opt-3 ($\mathcal{N}_4$), backward OR-opt-2 ($\mathcal{N}_5$), forward OR-opt-2 ($\mathcal{N}_6$), backward OR-opt-1 ($\mathcal{N}_7$), forward OR-opt-1 ($\mathcal{N}_8$) neighborhood structures. The second SeqVND, i.e. SeqVND-2 examines respectively 1-opt ($\mathcal{N}_1$), backward OR-opt-2 ($\mathcal{N}_5$), forward OR-opt-2 ($\mathcal{N}_6$), backward OR-opt-1 ($\mathcal{N}_7$), forward OR-opt-1 ($\mathcal{N}_8$), 2-opt ($\mathcal{N}_2$) neighborhood structures in that order. Apart of using different neighborhood structures, these two variants of SeqVND use different rules for choosing how to continue the search after finishing the search in some neighborhood structure. While SeqVND-1 continues search in the first neighborhood structure or in the next neighborhood structure or returns depending on whether some improvement has been found or not, SeqVND-2 always continues search in the next neighborhood structure. Therefore these two variants have different stopping conditions. SeqVND-1 stops when there is no improvement in the last neighborhood structure, i.e., forward OR-opt-1 neighborhood structure, while SeqVND-2 stops when there is no improvement between two consecutive iterations. Since SeqVND-2 explores a predefined set of neighborhood structures in the cyclic order, we name it Cyclic-VND. The pseudo-codes SeqVND-1 and SeqVND-2 are

Algorithm 1 SeqVND-1

```
Function SeqVND-1(X)
  k ← 1;
  repeat
    X' ← Local Search (X, N_k);
    if f(X') < f(X) then
      X ← X'; k ← 1;
    else
      k ← k + 1;
    end if
  until k > 8
  return X;
```

Algorithm 2 SeqVND-2

```
Function SeqVND-2(X)
  Let N' =< N_1, N_5, N_6, N_7, N_8, N_2 >;
  repeat
    k ← 1;
    while k < 7 do
      X' ← Local Search (X, N'_k);
      if f(X') < f(X) then
        X ← X';
      end if
      k ← k + 1;
    end while
  until there is no improvement
  return X;
```

provided at Algorithm 1 and Algorithm 2, respectively. Note that the repeat loop in Algorithms 1 and 2 may be also seen as a multi-level or a composite heuristic if each neighbourhood structure is treated as a heuristic at different level. In the algorithms below, we denote by $f(X)$ the cost of the tour X.

2.4 General variable neighborhood search for TSPDL

General variable neighborhood search (GVNS) [2,3] is a variant of variable neighborhood search [6] which uses SeqVND as the local search. It has been successfully applied for solving many variants of TSP [7–9] as well as for solving various combinatorial problems [5]. Since we have developed two SeqVND procedures, which are described in the previous section, we implement two GVNS methods as well. Both GVNS methods are developed using the scheme presented at Algorithm 4. The only difference between the GVNS methods is a different SeqVND used as a local search. Depending on how SeqVND is employed as a local search we denote these two GVNS methods as GVNS-1 (local search—SeqVND-1) and GVNS-2 (local search—SeqVND-2). In order to escape from the current local optima both GVNS methods use the same shaking function `Shake_TSPDL`(X, k) (Algorithm 3) which returns a solution obtained by preforming k times a random OR-opt-1 move on a given solution X. More precisely the shaking function at each iteration chooses at random one port from the tour X and moves it, either forward or backward, after another port, also chosen at random. Note that at each iteration of the shaking procedure, a feasible solution is generated. We allow generation of only feasible solutions since repairing feasibility of an infeasible solution might be time consuming especially for the largest instances. So, it is better to have more iterations within a GVNS procedure than to

Algorithm 3 Shaking procedure

```
Function Shake(X, k)
  for i = 1 to k do
    select X' ∈ OR_opt_1(X) at random; X ← X';
  end for
  return X;
```

Algorithm 4 GVNS for solving TSPDL

```
Function GVNS (p_max, t_max, X)
  repeat
    k ← 1;
    while k ≤ p_max do
      X' ← Shake(X, k) ;
      X'' ← SeqVND(X');
      if f(X'') < f(X) then
        X ← X''; k ← 1;
      else
        k ← k + 1;
      end if
    end while
  until CpuTime() > t_max
  return X;
```

spend allowed CPU time repairing feasibility of solutions generated by the shaking procedure. In that way, we have more chances to produce high quality solutions.

3 Computational results

For testing the proposed two approaches, we used the same benchmark instances as proposed in [12]. It consists of 240 instances derived from 8 classical TSP instances [13]: burma14, ulysses16, ulysses22, fri26, bayg29, gr17, gr22, gr48. Each instances is characterized by a matrix of edge costs, a number of ports, demands and draft limits. They can be found at: http://jgr.no/tspdl.

Furthermore, we generate 60 new large-size data sets for the TSPDL. The new ones are generated from three existing TSP instances: KroA100, KroA200, pcb442, [13] and contain respectively 100, 200 and 442 vertices. The generation of instances is performed following steps presented in [12]. From each TSP data instance i.e. KroA100, KroA200, pcb442, we generate randomly two new data set for TSPDL restricting the draft limits of 50 and 75 % of vertices to be less than the total demand. Each data set contains ten instances. They differ in vertices whose draft limits are restricted; they are chosen at random. The new instances are available at: http://www.mi.sanu.ac.rs/~nenad/TSPDL/.

For the proposed two GVNS based heuristics, two parameters have to be adjusted: p_{max} which represents the maximal level of perturbation and t_{max}, the maximum time allowed to be used by the GVNS heuristics. After some preliminary testing we set p_{max} and t_{max} to 30 and 100 s, respectively for both GVNS-1 and GVNS-2.

In Table 1, a summary results that compare our heuristics with the exact one based on *branch and cut and price* (B&C&P) provided in [1] are presented. Note that the B&C&P algorithm succeeded to solve all 240 instances to optimality within a time limit set to 7,200 s (2 h). Table 1 reports in the first column the instance name while in the second column it reports the average of ten optimal solutions found by the B&C&P algorithm. The average times for all three methods are also given in Table 1. The last column denoted by f_{TSP} provide the optimal solution value for the classical TSP. It appears: (1) GVNS-2 heuristic reaches all 240 optimal solutions with 1.06 s on average. Note that the B&C&P used 4112.04 s. (2) GVNS-1 is slightly faster than GVNS-2 (it spends 0.82 s on average), but it did not get optimal solution in only 2 (out of 240) instances.

In Table 2 a comparison between the two proposed GVNS methods on new instances is presented. For each method, we calculate percentage deviation between average value of its solutions and the Lower Bound (LB), i.e., the optimal solution for TSP. Thus, we calculate the percentage deviation Δ as follows : $\Delta = ((f - LB)/LB) \times 100$, Where f is the value of the objective function.

From Table 2, we again conclude that GVNS-2 outperforms GVNS-1 regarding average values of provided solution, although it consumes slightly greater amount of CPU-time. Note that average percentage deviation increases as the number of vertices increases. It should be emphasized that on some instances with 100 vertices GVNS variants succeeded to reach solutions equals to lower bound value. In other words, some instances with 100 vertices have been solved to optimality.

Table 1 Summary of the comparison between the existing method and the proposed two GVNS approaches on small instances

Inst	B&C&P		GVNS-1		GVNS-2		f_{TSP}
	Optimal	Time	Value	Time	Value	Time	
Burma_14_10	3,386.70	0.52	3,386.70	0.00	3,386.70	0.00	3,323
Burma_14_25	3,596.80	0.37	3,596.80	0.00	3,596.80	0.00	3,323
Burma_14_50	3,862.30	0.35	3,862.30	0.00	3,862.30	0.00	3,323
Ulysse_16_10	6,868.20	50.34	6,868.20	0.00	6,868.20	0.00	6,859
Ulysse_16_25	7,165.40	18.58	7,165.40	0.00	7,165.40	0.00	6,859
Ulysse_16_50	7,590.30	4.60	7,590.30	0.00	7,590.30	0.00	6,859
Ulysse_22_10	7,087.60	32.02	7,087.60	0.04	7,087.60	0.00	7,013
Ulysse_22_25	7,508.70	24.61	7,508.70	0.01	7,508.70	0.00	7,013
Ulysse_22_50	8,425.60	26.55	8,425.60	0.04	8,425.60	0.03	7,013
fri26_25	963.80	12.37	963.80	0.02	963.80	0.00	937
fri26_50	1,104.70	16.43	1,104.70	0.06	1,104.70	0.05	937
fri26_10	1,178.70	16.26	1,178.70	0.01	1,178.70	0.01	937
bayg29_10	1,713.60	11.99	1,713.60	0.00	1,713.60	0.02	1,610
bayg29_25	1,792.60	11.05	1,792.60	0.14	1,792.60	0.02	1,610
bayg29_50	2,091.00	13.38	2,091.00	0.01	2,091.00	0.05	1,610
gr17_10	2,150.30	25.12	2,150.30	0.00	2,150.30	0.00	2,085
gr17_25	2,237.70	11.11	2,237.70	0.00	2,237.70	0.00	2,085
gr17_50	2,710.30	5.54	2,710.30	0.00	2,710.30	0.00	2,085
gr21_10	2,833.60	11.63	2,833.60	0.00	2,833.60	0.01	2,707
gr21_25	2,962.60	11.44	2,962.60	0.00	2,962.60	0.00	2,707
gr21_50	3,738.10	10.32	3,738.10	0.01	3,738.10	0.00	2,707
gr48_10	5,284.40	1,352.72	5,284.40	0.20	5,284.40	3.39	5,046
gr48_25	5,800.30	861.47	5,802.60	13.25	5,800.30	13.65	5,046
gr48_10	6,635.70	211.35	6,635.70	5.97	6,635.70	8.31	5,046
Average	4,112.04	114.17	4,112.14	0.82	4,112.04	1.06	3,697.5

Table 2 Comparison between the proposed two GVNS approaches on new large instances

Inst	LB	GVNS-1		GVNS-2		Δ_{GVNS-1}	Δ_{GVNS-2}
		Value	Time	Value	Time		
KroA100_50	21,282	21,325.60	16.92	21,294.00	27.86	0.20	0.06
KroA100_75	21,282	21,359.40	29.41	21,303.40	32.67	0.36	0.10
KroA200_50	29,368	30,869.30	66.00	30,665.20	77.51	5.11	4.42
KroA200_75	29,368	32,041.50	79.37	30,896.10	77.05	9.10	5.20
pcb442_50	50,778	61,170.30	77.51	59,858.30	72.74	20.47	17.88
pcb442_75	50,778	63,889.70	72.32	61,010.10	83.36	25.82	20.15
Average	33,809.33	38,442.63	56.92	37,504.52	61.87	10.18	7.97

4 Conclusions

We presented two heuristics based on general variable neighborhood search for the travelling salesman problem with draft limits. They differ in a strategy of choosing the next neighborhood, from a given list, in the deterministic part of GVNS. These heuristics are tested on benchmark instances from the literature as well as on new generated instances by us. Computational results prove their efficiency and effectiveness in solving TSPDL. Additionally, computational results show that heuristic GVNS-2 is able to solve all instances from literature to optimality where optimal solutions were known. For our new generated instances available at the web, we analyze the difference between our two GVNS based variants.Future work may include an hybridization between a mathematical model and GVNS based heuristic proposed here in order to enhance the results.

Acknowledgements This work was supported by the *Centre National de la Recherche Scientifique* (CNRS), by the *Campus interdisciplinaire de recherche, d'innovation technologique et de formation Internationale sur la Sécurité et l'Intermodalité des Transports* (CISIT), and by the *Laboratoire d'Automatique, de Mécanique et d'Informatique industrielles et Humaines* (LAMIH) of the *Université de Valenciennes et du Hainaut-Cambrésis*.

References

1. Battara, M., Pessoa, A.A., Subramanian, A., Uchoa, E.: Exact algorithms for the travelling salesman problem with draft limits. Eur. J. Oper. Res. doi:10.1016/j.ejor.2013.10.042
2. Hansen, P., Mladenović, N., Moreno-Pérez, J.A.: Variable neighbourhood search: methods and applications. 4OR **6**, 319–360 (2008)
3. Hansen, P., Mladenović, N., Moreno-Pérez, J.A.: Variable neighbourhood search: methods and applications. Ann. Oper. Res. **175**, 367–407 (2010)
4. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: a case study in local optimization. In: Aarts, E.H.L., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, pp. 215–310. Wiley, New York (1995)
5. Mjirda, A., Jarboui, B., Macedo, R., Hanafi, S., Mladenović, N.: A two phase variable neighborhood search for the multi-product inventory routing problem. Comput. Oper. Res. doi:10.1016/j.cor.2013.06.006i
6. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**, 1097–1100 (1997)
7. Mladenović, N., Todosijević, R., Urosević, D.: An efficient General variable neighborhood search for large TSP problem with time windows. Yugosl. J. Oper. Res. **22**, 141–151 (2012)
8. Mladenović, N., Todosijević, R., Urosević, D.: Two level General variable neighborhood search for Attractive traveling salesman problem. Comput. Oper. Res. doi:10.1016/j.cor.2013.04.015
9. Mladenović, N., Urosević, D., Hanafi, A., Ilić, A.: A General variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. Eur. J. Oper. Res. **220**, 270–285 (2012)
10. Mladenović, N., Urosević, N., Hanafi, S.: Variable neighborhood search for the travelling deliveryman problem. 4OR **11**, 57–73 (2012)
11. Panchamgam, K., Xiong, Y., Golden, B., Dussault, B., Wasil, E.: The hierarchical traveling salesman problem. Optim. Lett. **7**, 1–8 (2013)
12. Rakke, J.G., Christiansen, M., Fagerholt, K., Laporte, G.: The travelling salesman problem with draft limits. Comput. Oper. Res. **39**, 2162–2167 (2012)
13. Reinelt, G.: TSPLIB—a traveling salesman problem library. ORSA J. Comput. **3**, 376–384 (1991)
14. Yadlapalli, S., Rathinam, S., Darbha, S.: 3-Approximation algorithm for a two depot, heterogeneous traveling salesman problem. Optim. Lett. **6**, 141–152 (2012)