International Workshop
on Green Supply Chain

GSC' 2012

June 21 - 22, 2012

ARRAS- FRANCE

A hybrid ILS/VND heuristic for the One-Commodity Pickup-and-Delivery Traveling Salesman Problem *

Libo REN [a], Christophe DUHAMEL [b],
Alain QUILLIOT [a]

[a] *LIMOS UMR CNRS 6158, Université Clermont-Ferrand II,*
Campus des Cézeaux, 63173 Aubière Cedex France
ren@isima.fr, alain.quilliot@isima.fr

[b] *ICD-LOSI, STMR UMR CNRS 6279, Université de Technologie de Troyes,*
12, Rue Marie Curie, BP 2060, F-10010 Troyes Cedex France
christophe.duhamel@utt.fr

Abstract

The One-commodity Pickup-and-Delivery Traveling Salesman Problem (1-PDTSP) is a variant of the Traveling Salesman Problem (TSP) introduced by Hernández-Pérez et Salazar-González [1]. In the 1-PDTSP, a set of customers has to be addressed by one vehicle initially located at the depot. Each customer is associated with a demand of the same item type. According to its demand, a customer can be a supplier (with pickup operation) if its demand is positive or a consumer (with delivery operation) otherwise. Any item picked up at a customer can be used to a delivery. The objective is to find a minimum cost Hamiltonian route for a capacitated vehicle. It has to perform the pickups and the deliveries at each customer in order to satisfy all of requests. In the real world application, the 1-PDTSP can be employed for the management of the public bicycle sharing systems like *Vélib'* in Paris. In this paper, we propose a hybrid Iterated Local Search (ILS) / Variable Neighborhood Descent (VND) heuristic for the 1-PDTSP. In the ILS, an assignment model is used at first to generate a feasible initial solution. The same model is served then as major shaking procedure, which aims at finding a *k-position* neighbor of the incumbent solution, where *k* defines the maximum number of customer positions different between two solutions. All generated neighbor solutions using this model are stored in a vector in order to avoid the solution duplication. An explored neighbor solution is then improved using the improvement procedure, which combines an auxiliary shaking step and a VND local search with a strategy of relaxation of capacity. The proposed heuristic is tested on small and medium instances. The computational results show that it gives competitive results comparing with the existing heuristics.

Key words: Pickup-and-Delivery Problem (PDP), Traveling Salesman Problem (TSP), 1-PDTSP, ILS, VND, heuristic.

1 Introduction

Public bicycle sharing systems like *Vélo'V* in Lyon (2005) and *Vélib'* in Paris (2007) are now widespread in French cities. Their primary aim is to reduce the congestion in urban centers by providing a greener transportation means. Consequently, this should also reduce the chemical and noise pollution and improve the quality of life in city centers. Yet, those systems suffer form some operational issues. For instance, coherent population moves (going to work in the morning, going back

* This paper was not presented at any other revue.

home at the end of the day) lead to disparities on the stations: some stations are empty at some time of the day while others are saturated. Moreover, stations at the most elevated parts of cities (Montmartre in Paris for instance) typically face a bicycle imbalance. Adjusting the pricing or improving the stations location can help reducing the problem, but this does not remove the need for proactive relocation operations. Agents use vehicles to pick bicycles form saturated stations to put them on empty stations. This improves the global availability of the system. Due to the size of the system, optimizing the availability is an important issue.

We consider here a single agent and we model the problem as the One-Commodity Pickup-and-Delivery Traveling Salesman Problem (1-PDTSP). It is a variant of the Traveling Salesman Problem (TSP) introduced by Hernández-Pérez et Salazar-González [1]. In the 1-PDTSP, a set of sites is considered. It consists of a group of customers and one depot. Each customer is associated with a request which corresponds to an items demand, positive or negative. The customers can be divided into two subsets according to their demands: the pickup customers provide items while delivery customers consume the items. Thus, there is a single homogeneous commodity, without specific identifications or known destination. Then, items collected at pickup customers can be delivered to any delivery customers. Each customer is served only once by a capacitated vehicle. The vehicle starts and ends its route at the depot. Moreover, the depot can provide items and the vehicle may leave the depot with some items. Without loss of generality, the sum of the customers' requests is null, which means the vehicle must return to the depot with the same initial load. The objective is to find a minimum cost Hamiltonian route for the vehicle, and each customer must be visited once and its request must be satisfied, while respecting the vehicle capacity constraint.

The 1-PDTSP is defined on a complete and directed graph $G = (V, A)$, where $V = \{0, .., n\}$ is the set of sites and $E = \{(i, j) | \forall i, j \in V,\ i \neq j\}$ is the set of arcs. The set of sites contain the depot (denoted by 0) and n customers (denoted by $1, .., n$). Each customer i is associated with a demand q_i, $q_i < 0$ ($q_i > 0$) if i corresponds to a delivery (pickup) customer. The depot can be considered as a special customer by setting $q_0 = -\sum_{i \in V \setminus \{0\}} q_i$, i.e., a customer absorbing or providing the necessary amount of item to ensure product conservation [1]. The capacity of vehicle is denoted by Q. Moreover, the depot can provide an additional initial amount of items C_0 (that is not consumed by the customers) to the vehicle when leaving the depot, with $0 \leq C_0 \leq (Q - |q_0|)$. Each arc (i, j) is associated with a non-negative cost D_{ij}. The objective of the 1-PDTSP is to find the minimum cost Hamiltonian tour considering the following constraints:

- The route starts and ends at the depot;
- Each customer is visited one time;
- The vehicle load does not exceed its capacity;
- All requests are satisfied.

The 1-PDTSP is NP-hard (see [1]). Moreover, even finding a feasible solution may be very difficult when the vehicle capacity is small (close to the largest demand). According to the classification of Berbeglia *et al.* [2], the 1-PDTSP can be considered as a Pickup-and-Delivery Problem (PDP) with a *many-to-many* structure and a single type of commodity. The commodity is characterized by its specific feature, since any vertex can serve as a source or as a destination for each request. Recently, Hosny and Mumford [3] proposed to refer this class of problems as the One-Commodity Pickup and Delivery Problem (1-PDP). Those many-to-many problems are relevant in shared transportation systems since we do not need to consider each shared-vehicle separately. We focus here on the management of public bicycle sharing systems like *Vélo'V* in Lyon or *Vélib'* in Paris. Thus, the 1-PDTSP is used to model the reorganization of the bicycle availability at each location site under a static context. In such an application, the items are the bicycles. They all are similar and should be collected from some sites and delivered to others sites in order to balance the availability of each site of the sharing system.

In this paper, we propose a hybrid ILS/VND heuristic to solve the 1-PDTSP. In this approach, an assignment model is used in the construction phase and in the major shaking step of ILS [4]. The integer linear programming model aims at building the feasible initial solutions in the construction

phase. It uses a *k-position* neighborhood structure to define the neighbor solutions of the incumbent solution whenever in shaking step. The model can be solved with any linear programming solver in a short computational time. All generated neighbor solutions are stored in a global vector in order to avoid any possible duplication during the execution. A neighbor solution is then improved using an improvement phase which combines an auxiliary shaking step and a VND local search [5] with several adaptations of classic edge-exchange operators from the TSP. Those adaptations are defined on top of a capacity relaxation scheme in order to be able to explore the solution space more efficiently as described in Hernández-Pérez *et al.* [6]. The proposed heuristic is tested on the small and medium instances introduced in Hernández-Pérez *et al.* [7].

The paper is organized as follows. Section 2 is a summary of works related to the 1-PDTSP. Section 3 describes the proposed approach. The computational results are presented in section 4, before conclusions in section 5.

2 Related Work

Since the 1-PDTSP is a recent variation of the TSP, few articles have been published in the literature. Hernández-Pérez and Salazar-González presented a branch-and-cut algorithm to solve instances with up to 50 customers [8]. The exact algorithms are not suitable for solving large instances, but they can be used for providing good lower bounds.

Then, Hernández-Pérez and Salazar-González proposed two heuristic algorithms in [9]. The first one is based on a greedy algorithm followed by an improvement phase using edge-exchange operators. Since it is difficult to generate a feasible solution when the vehicle capacity is close to the largest demand, they introduced the notion of *infeasibility* to evaluate a solution. It measures the difference between the vehicle capacity and the largest load during its trip. A solution s is said to be feasible if $infeas(s) \leq 0$. A multi-start scheme is used in this heuristic: each iteration starts with an initial solution built by a nearest neighbor (NN) algorithm, which encourages the selection of edges connecting customers of different types by slightly penalizing the use of edges connecting same customer-type. The initial solution may still not to be feasible. It is then improved by a local search based on *2-Opt* and *3-Opt* edge exchanges. The second heuristic relies on a partial optimization strategy. It uses the same construction algorithm as before. A new neighborhood structure, called *k-neighborhood* is defined, where *k* measures the difference in terms of edges. In order to explore the *k-neighborhood* of a solution, they adapt the ILP model in [1] using an additional inequality, and then solve it using a branch-and-cut algorithm. Thus, the approach corresponds to a limited descent (with at most 6 steps) in the local branching strategy [10].

The same authors also propose a hybrid GRASP/VND heuristic in [6]. It combines a GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristic with a VND (Variable Neighborhood Descent) local search. GRASP is based on the repetition of a greedy construction algorithm and a local search. VND was initially defined as the local search of VNS [5]. It relies on several pre-defined neighborhood operators, which are usually sorted according to their complexity. When one operator is not able to improve the current solution, the VND switches to the next one, otherwise it performs the move before resuming to the first operator. The VND stops whenever a local optimum for all the operators is reached. Hernández-Pérez and Salazar-González also define a second VND (with vertex-exchange operators) in a post-optimization step. Infeasible solutions are allowed in the GRASP/VND step, since its aim is to find a good feasible solution. Then the post-optimization tries to improve the solution computed in the first step while satisfying the vehicle capacity.

Zhao *et al.* [11] have proposed a Genetic Algorithm for the 1-PDTSP. The algorithm first builds a population of feasible solutions using a non-deterministic nearest-neighbor construction heuristic. The initial population is then improved by a local search based on the *2-Opt* edge-exchange operator. A pheromone-based crossover operator is proposed to generate new solutions. Such an idea comes from the ant colony algorithm, which stores pheromone trails related to arcs as global information. In order to generate a child solution, the pheromone-based operator first randomly selects a customer to be the first visited in the trip. It then iteratively appends a feasible customer taking into account its neighbors

in both two parents and the pheromone related to the last customer inserted. If all the neighbors have been already inserted or if none of them can be added to the end of the tours, a probabilistic rule is used to sort the non-visited customers. Then on of them is selected by considering its priority. This rule takes into account the pheromone on the arcs linked to the last customer of the trip. Otherwise, the nearest feasible neighbor is selected. The child solution has to be feasible. It is then improved by a 2-Opt before applying the 3 vertex-exchange mutation operator. A new population is generated by replacing half of its solutions by the children. The pheromones are updated after each population generation. This algorithm works on the space of feasible solutions, and thus does not need any relaxation of vehicle capacity.

More recently, Hosny *et al.* [3] proposed a hybrid VNS/SA (Variable Neighborhood Search/Simulated Annealing) for solving the 1-PDTSP. Their idea is to use a SA acceptation criterion when a local optimum explored during the VNS is worse than the current solution. This criterion allows local search to escape from a local optimum besides the original VNS shaking mechanism. The proposed algorithm uses an iterative scheme. Each iteration starts with the incumbent solution obtained in the previous iterations. The maximum neighborhood size is not fixed and it depends on the current iteration number. Actually it is gradually reduced during the iterations: early iterations use large neighborhood sizes, which allows wider jumps in the solution space (from the current solution); later iterations only allow limited explorations in the search space, in order to maintain the solution quality. Moreover, they use a variation of the construction algorithm proposed by [11]. The infeasible solutions are allowed both in the construction phase and in the local search.

3 The proposed approach for the 1-PDTSP

The approach we propose uses a hybrid ILS/VND scheme, which contains an assignment model and an improvement procedure. The assignment model is used firstly to generate a feasible initial solution in the construction step. It is served then as major shaking step, which aims at finding a *k-position* neighbor to the incumbent solution. Only feasible solutions are accepted by the assignment model. More detail of this model is described in the section 3.1. An accepted neighbor solution is then improved by a procedure combines an auxiliary shaking step and a VND local search. The improvement procedure relies on *k-Opt* edge-exchange operators and a vertex-exchange mutation operator (see section 3.2). It allows the relaxation of the vehicle capacity. The stopping criterion in the ILS is defined as a fixed number of iterations. Algorithm 1 outlines the proposed approach for the 1-PDTSP. Each component is described in next subsections.

Algorithm 1: Hybrid ILS/VND heuristic for the 1-PDTSP

```
1  s* ← Assignment_model();                    // construction step
2  while stopping criterion is not satisfied do
3  |     s ← Assignment_model() ;              // major shaking step
4  |     s ← Improvement_procedure(s) ;        // improvement step with VND local search
5  |     if s is feasible and improves the best solution s* then
6  |     |     s* ← s ;
7  |     end if
8  end while
```

3.1 Assignment model

Most of the recent approaches to the 1-PDTSP use the nearest neighbor heuristics to generate the initial solutions. According to our computational experience, the probability to get a feasible solution using those heuristics is usually poor. Moreover, they only produce infeasible solutions for several most hard instances. Therefore, we propose a construction method based on an integer linear programming model (referred as Assignment_model()), which guarantees feasible initial solutions. At the same time, it is used as major shaking procedure by introducing a vertex-position based neighborhood structure, called *k-position* neighborhood. Given two solutions s' and s'', let $dist(s', s'')$ be their distance measured as the sum of the customer positions differences between the two trips. Thus, s' is considered as a *k-position* neighbor of s'' if $dist(s', s'') \leq k$. This neighborhood is expressed as a constraint in the linear programming model, and k defines the maximum number of customer positions differences between the current trip and the incumbent trip.

The proposed linear programming model corresponds to an assignment problem, which is defined as: given a set of vertices $V' = \{1, .., n\}$, in which vertex i is associated to the request $q_i \in [-Q, Q]$, and Q is the vehicle capacity; given a set of trips (generated since the beginning of execution) $S = \{s_1, s_2, .., s_t\}$ (each trip is a vector of size n) and given the incumbent solution s^*. Let s' be the vector corresponding to the current solution. The objective is to find an assignment between vertices and positions in s', while satisfying the following constraints:

- (c1) every vertex is assigned one position in s';
- (c2) every position in s' corresponds to one vertex;
- (c3) the vehicle capacity is ensured;
- (c4) s' has to be a *k-position* neighbor of s^* if $s^* \neq \emptyset$;
- (c5) s' has to be different to all the trips in S.

To provide a mathematical model of this assignment problem, the following variables are defined:

- $x_i^j \in \{0,1\}$: 1 if the vertex i is assigned to the position j, 0 otherwise;
- $f_j \in \mathbb{N}$: load of the vehicle after visiting the jth customer in the current trip.

Then the assignment problem can be formulated as:

$$
(\text{LP } 1)\begin{cases}
\text{Minimize } \displaystyle\sum_{i \in V'} \sum_{j=1 \dots n} x_i^j & & (1)\\
\text{s.c.} & & \\
\displaystyle\sum_{j=1 \dots n} x_i^j = 1 & \forall i \in V' & (2)\\
\displaystyle\sum_{i \in V'} x_i^j = 1 & \forall j = 1 \dots n & (3)\\
f_j - f_{j-1} - \displaystyle\sum_{i \in V'} q_i x_i^j = 0 & \forall j = 2 \dots n & (4)\\
\displaystyle\sum_{j=1 \dots n} x_{s^*[j]}^j \geq \lfloor \alpha \cdot n \rfloor & & (5)\\
\displaystyle\sum_{j=1 \dots n} x_{s_l[j]}^j \leq n - 3 & \forall\, l = 1 \dots |S| & (6)\\
x_i^j \in \{0, 1\} & \forall i \in V', j = 1 \dots n & (7)\\
f_j \in \mathbb{N} & \forall j = 1 \dots n & (8)
\end{cases}
$$

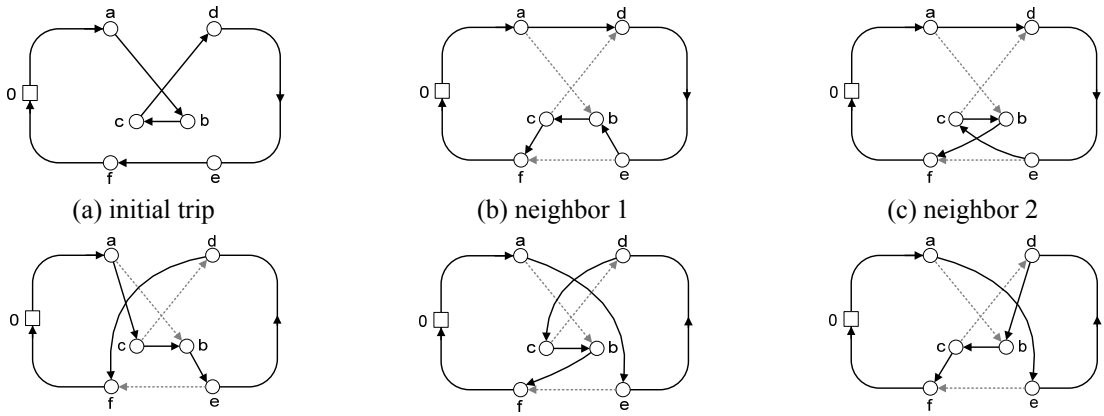
The objective function (1) is a constant since one only look for a feasible assignment. Constraints (2) to (6) correspond to (c1) to (c5); constraints (7) and (8) define respectively the variables x_i^j and f_j. Moreover, the constraint (5) uses a parameter $0.5 \leq \alpha < 1$, which sets the *k-position* neighborhood structure between s' and s^*. Its value depends on the current size of S and on the instances size: $\alpha = 0.5$ if $|S| \leq 10$; otherwise $\alpha = 0.8$ for small instances and $\alpha = 0.9$ for medium instances. Note that every initial trip generated by this model is then added into S. This assignment problem is quite easy to solve for any commercial solvers like CPLEX or Gurobi.

3.2 Improvement phase

The improvement phase is done by a procedure which combines here deterministic and stochastic neighborhoods. It relies on an iterative scheme: each iteration starts with the solution s' obtained in the previous iteration. Then an auxiliary shaking is applied to s' before performing the VND local search. The auxiliary shaking consists in randomly performing vertex-exchange operations (*swap*) and the VND local search uses several edge-exchange operators (*k-Opt*). As mentioned before, infeasible solutions are allowed during the improvement phase. More precisely, a Hamiltonian trip could violate the vehicle capacity constraint, in order to extend the solutions space and to avoid being trapped too easily in a local optimum. Consequently, all operators are adapted to handle infeasible solutions. Such a strategy has been already applied in [6].

The selection criterion in our *k-Opt* operators considers both infeasibility and quality of the neighbor solutions. The infeasibility threshold in those operators is considered as an input parameter, and it is denoted by tol. It is updated after each move in order to guide the operator towards feasible solutions. A neighbor solution s is accepted if $infeas(s) \leq tol$ and if it has a better value. If the accepted solution is feasible, then tol is set to 0, i.e. no infeasible solutions will be accepted henceforth; Otherwise, tol is updated by setting $tol = max\{0, (tol - \Delta)\}$, where Δ is a pre-defined value depending on the instance size. The selection criterion in *swap* operator only takes into account the infeasibility of the neighbor solution. A new solution is accepted if its infeasibility is smaller than tol, and tol is not modified here.

The local search uses two edge-exchange operators: *2-opt* and *3-opt* for small instances (≤ 60 vertices); *2-opt* and *or-opt* for medium instances (> 60 vertices). The edge-exchange operators test all possible combinations and use *best-accept* as replacement strategy. The *3-opt* explores all neighbor structures shown in Figure 1 and the *or-opt* considers only the first one in order to reduce the processing time.



(a) initial trip (b) neighbor 1 (c) neighbor 2

(e) neighbor 3 (f) neighbor 4 (g) neighbor 5

Figure 1 : 3-arcs-exchange movements

The vertex-exchange mutation operator used in the auxiliary shaking step is *swap*. It randomly selects 2 vertices in the current trip before swapping them. The number of swaps done consecutively is randomly chosen between 5 and 15. It uses a *first-accept* criterion.

The stopping criterion used in the improvement procedure takes into account the feasibility of the solution obtained. Note that the input solution is always feasible, and infeasible solutions are allowed during the local search. The initial infeasibility threshold in this procedure is defined by tol_{init}. It is calculated using the same equation proposed in [6], i.e.: $tol_{init} = 3\,max\{\sum_{i \in V: q_i > 0} q_i\,, -\sum_{i \in V: q_i < 0} q_i\}/n$. Only feasible trips are accepted as incumbent solution. The procedure is stopped whenever the incumbent solution is improved in the last it_{max} iterations or the value of tol_{init} is null. Otherwise tol_{init} is decreased in order to restrict the infeasible solution space, before starting the next iterations. The iterations number is not fixed in the procedure, and it could grow up to $(tol_{init} * it_{max})$. The value of it_{max} is set to 5 in our configuration. Moreover, the infeasibility value tol is set to 0 whenever the current solution returned becomes feasible. The VND local search in this procedure using 2 edge-exchange operators. The improvement procedure is detailed in Algorithm 2.

Algorithm 2: Improvement procedure with VND local search | Improvement_procedure(s)

```
1  tol_init ← initial value ;
2  s' ← s ;
3  it ← 1 ;
4  stop ← false ;
5  while not stop do
6  |    tol ← tol_init;
7  |    s" ← swap(s', tol) ;
8  |    do
9  |    |    do
```

```
10  |   |   |   {s", tol} ← 2-opt(s", tol) ;
11  |   |   while s" is improved it this iteration
    |   |
12  |   |   if |s| ≤ 60
13  |   |   |   {s", tol} ← 3-opt(s", tol) ;
14  |   |   else
15  |   |   |   {s", tol} ← or-opt(s", tol) ;
16  |   |   end if
17  |   while s" is improved by last operator it the current iteration;
    |
18  |   if s" is feasible and it improves the incumbent solution s' then
19  |   |   s' ← s";
20  |   end if
    |
21  |   if it = it_max and (s' is improved or tol_init = 0) then
22  |   |   stop ← true ;
23  |   else if it = it_max then
24  |   |   tol_init ← tol_init − 1;
25  |   |   it ← 1;
26  |   else
27  |   |   it ← it + 1;
28  |   end if
29  end while
```

4 Computational results

The proposed approach is implemented in C++ with GNU gcc compiler (version 4.1.2) and Gurobi libraries (version 4.5.1). All tests are performed on a PC with AMD Opteron CPU (2.3 GHz) under Linux (CentOS 64-bit). The number of cores used in Gurobi libraries is set to 1. The computational experiments have been tested on the benchmark instances of 1-PDTSP from [7]. These instances are generated by Hernández-Pérez *et al.*, in which $n-1$ customer are randomly located in the [-500, 500] x [-500,500] square, and each customer i is associated with a random demand $q_i \in [-10, 10]$. The depot is located at (0, 0) with demand $q_0 = -\sum_{i=1}^{n} q_i$. Moreover, 2 groups of instances are used: small instances with number of vertices $n \in \{20, 30, 40, 50, 60\}$ and medium instances with $n \in \{70, 80, 90, 100\}$. For each value of n, 10 different instances are defined and identified from 'A' to 'J'. For each instance, each vehicle capacity $Q \in \{10, 15, 20, 25, 30, 35, 40, 45, 50, 1000\}$ is considered. Those instances can be downloaded from http://webpages.ull.es/users/hhperez/PDsite/index.html.

We report experiments with the smallest capacities: $Q = 20$ for instances with $n = 90$ (no existing results are given for smaller vehicle capacities), and $Q = 10$ for others. The proposed heuristic was run 5 times on each instance and the number of iterations is set to 300. The computational time is presented in two parts: shaking time for the assignment model in the major shaking step and improvement time for the improvement procedure.

Table 1 shows the computational results for the small instances. The optimum values come from in [8]. The combination of columns "Group" and "Id" give the instance name; column "Optimal" gives the optimal value; column "Average" shows the average value obtained in 5 runs; the best value obtained is in column "Best" with the number of times it has been found over the 5 runs; column "% gap" is the relative gap in percent between the best value found and the optimal value; column "Best it." reports the smallest number of iterations number for obtaining the best solution; columns "Times[1]" and "Time[2]" display respectively the average shaking time and the average improvement time in seconds (for 300 iterations).

Table 1: computational results for small instances

Group	Id	Optimal	Proposed heuristic					
			Average	Best	% gap	Best it.	Time[1]	Time[2]
N20q10	A	4963	4963.0	**4963** (5)	0.00	1	29.37	8.39
	B	4976	4976.0	**4976** (5)	0.00	1	33.45	8.74
	C	6333	6333.0	**6333** (5)	0.00	1	104.93	50.71
	D	6280	6280.0	**6280** (5)	0.00	2	77.52	30.99

	E	6415	6415.0	**6415** (5)	0.00	1	31.87	35.75
	F	4805	4805.0	**4805** (5)	0.00	1	42.52	11.78
	G	5119	5119.0	**5119** (5)	0.00	1	26.88	8.03
	H	5594	5594.0	**5594** (5)	0.00	1	32.73	10.81
	I	5130	5157.0	**5157** (5)	0.00	11	35.96	15.89
	J	4410	4410.0	**4410** (5)	0.00	1	38.62	11.25
N30q10	A	6403	6403.0	**6403** (5)	0.00	3	103.51	100.27
	B	6603	6603.0	**6603** (5)	0.00	1	98.12	76.25
	C	6486	6486.0	**6486** (5)	0.00	2	85.04	77.15
	D	6652	6652.0	**6652** (5)	0.00	7	89.09	137.83
	E	6070	6070.0	**6070** (5)	0.00	1	87.73	76.76
	F	5737	5737.0	**5737** (5)	0.00	1	79.87	82.05
	G	9371	9371.0	**9371** (5)	0.00	1	86.42	144.60
	H	6431	6431.0	**6431** (5)	0.00	1	80.67	71.64
	I	5821	5821.0	**5821** (5)	0.00	4	81.01	143.22
	J	6187	6187.0	**6187** (5)	0.00	4	94.80	83.74
N40q10	A	7173	7173.0	**7173** (5)	0.00	1	201.34	221.39
	B	6557	6560.0	**6557** (4)	0.00	4	249.09	213.05
	C	7528	7528.0	**7528** (5)	0.00	14	189.24	223.46
	D	8059	8059.0	**8059** (5)	0.00	7	213.88	255.52
	E	6928	6928.0	**6928** (5)	0.00	5	204.20	232.12
	F	7506	7514.8	**7506** (3)	0.00	17	232.48	255.83
	G	7624	7642.0	**7624** (3)	0.00	35	211.09	390.48
	H	6791	6791.0	**6791** (5)	0.00	5	224.20	290.93
	I	7215	7215.0	**7215** (5)	0.00	2	258.46	327.60
	J	6512	6512.0	**6512** (5)	0.00	2	205.43	156.94
N50q10	A	6987	6987.0	**6987** (5)	0.00	4	265.20	232.16
	B	9488	9488.0	**9488** (5)	0.00	15	677.49	521.52
	C	9110	9114.4	**9110** (3)	0.00	8	480.71	542.49
	D	10260	10299.5	**10260** (4)	0.00	40	435.20	591.66
	E	9492	9492.0	**9492** (5)	0.00	17	453.64	837.17
	F	8684	8684.0	**8684** (5)	0.00	21	486.02	984.81
	G	7126	7126.0	**7126** (5)	0.00	31	447.54	547.54
	H	8885	8912.4	**8885** (3)	0.00	61	423.08	436.45
	I	8329	8339.8	**8329** (4)	0.00	31	373.15	786.06
	J	8456	8256.0	**8456** (5)	0.00	27	639.50	611.44
N60q10	A	8602	8602.0	**8602** (5)	0.00	35	763.74	697.91
	B	8514	8514.0	**8514** (5)	0.00	48	1301.75	732.57
	C	9453	9476.6	**9453** (1)	0.00	257	1083.57	808.37
	D	11059	11131.8	11061 (1)	0.02	149	1208.36	781.50
	E	9487	9529.0	**9487** (3)	0.00	73	831.26	649.18
	F	9063	9105.4	**9063** (2)	0.00	191	1253.59	1232.53
	G	8912	8963.6	**8912** (2)	0.00	104	1246.94	895.69
	H	8424	8426.8	**8424** (4)	0.00	33	846.91	694.55
	I	9394	9451.6	**9394** (1)	0.00	287	894.97	886.43
	J	8750	8765.6	**8750** (4)	0.00	90	816.56	662.09

In the Table 1, we can see that the proposed heuristic is able to find the optimal values at least once in the 5 runs for 49 out of the 50 instances. Comparing with the existing heuristics in the literature, optimal values were found 46 in [6] (with 25 runs), 50 in [11] (with 10 runs) and 39 in [3] (with 5 runs). For the instance "N60q10D" where the optimum was not found, the gap is 0.02%. The average shaking time (respectively improvement time) is 45.39 seconds (19.23 seconds) for the group of 20 vertices and 1024.76 seconds (804.08 seconds) for the group of 60 vertices.

Table 2 shows the computational results for the medium instances. Its columns have the same definition as in Table 1, except for columns "BKR" and "% gap": the column "BKR" reports the value of the best known results in the literature, since the optimal values are not known; the column "% gap" becomes then the difference between the best value ILS/VND found and the value of the best known solution. The best known solutions come from [7] for $n \in \{70, 80, 90\}$ and from [3, 6 and 11] for $n = 100$.

Table 2: computational results for medium instances

Group	Id	BKR	Proposed heuristic					
			Average	Best	% gap	Best it.	Time[1]	Time[2]
N70q10	A	9312	9312.0	**9312** (5)	0.00	25	1295.13	101.15
	B	10106	10117.6	**10106** (4)	0.00	74	2798.73	133.13
	C	9959	9980.6	**9959** (2)	0.00	107	1324.58	154.55
	D	10386	10386.0	**10386** (5)	0.00	21	2315.64	176.23
	E	13055	13107.2	13069 (1)	0.11	247	2430.43	201.97

	F	10191	10224.2	10198 (3)	0.07	64	2571.73	158.81
	G	9916	9994.6	**9916** (1)	0.00	240	2106.90	152.64
	H	8868	8897.4	**8868** (2)	0.00	245	2127.29	140.63
	I	10051	10092.4	**10051** (3)	0.00	138	2820.31	188.81
	J	10414	10474.2	**10414** (2)	0.00	16	1437.17	126.47
N80q10	A	11597	11532.8	**11421** (1)	-1.52	298	3042.07	144.23
	B	12861	12883.2	**12861** (2)	0.00	203	5264.89	238.28
	C	12471	12414.0	**12358** (2)	-0.91	90	3372.82	273.82
	D	11050	11160.0	**11050** (2)	0.00	132	3804.69	201.55
	E	11185	11255.4	**11185** (2)	0.00	115	4835.44	332.34
	F	14012	13677.4	**13650** (1)	-2.58	275	3544.13	302.81
	G	11413	11154.0	**11108** (1)	-2.67	280	3292.22	228.64
	H	11307	11112.6	**11075** (1)	-2.05	124	3777.44	235.63
	I	11063	11185.0	**11063** (3)	0.00	181	4428.85	283.27
	J	9263	9278.2	**9263** (3)	0.00	113	2605.56	144.36
N90q20	A	8079	8083.2	**8079** (2)	0.00	24	639.65	73.04
	B	8673	8567.4	**8507** (1)	-1.91	248	954.33	95.10
	C	8448	8478.2	8468 (2)	0.24	38	658.10	89.80
	D	9369	9504.8	9469 (1)	1.07	272	1045.29	109.86
	E	10072	9749.0	**9674** (1)	-3.95	286	866.55	118.07
	F	10295	10053.4	**9961** (1)	-3.24	250	878.00	119.18
	G	8339	8360.6	8355 (1)	0.19	86	580.06	83.27
	H	9234	9243.6	**9234** (2)	0.00	165	580.08	116.39
	I	8601	8658.6	**8601** (1)	0.00	193	626.25	111.48
	J	8204	8204.0	**8204** (5)	0.00	8	710.18	84.24
N100q10	A	11741	11762.2	**11700** (1)	-0.35	258	6619.31	280.75
	B	13066	13128.4	**13003** (1)	-0.48	236	6242.00	383.59
	C	13893	13975.4	13896 (1)	0.02	221	10274.80	400.09
	D	14253	14341.6	14257 (1)	0.03	279	11312.30	481.68
	E	11411	11416.4	**11411** (2)	0.00	191	14003.20	576.32
	F	11644	11689.2	**11635** (1)	-0.08	297	5199.37	340.74
	G	12025	11983.0	**11866** (1)	-1.32	299	8573.25	326.39
	H	12818	12727.2	**12673** (2)	-1.13	144	7449.64	445.39
	I	14025	13929.0	**13834** (1)	-1.36	282	8952.98	479.24
	J	13293	13371.4	**13222** (1)	-0.53	244	5563.60	275.94

The results in Table 2 show an average improvement of 0.97% on the gap for $n = 80$ and 0.76% for $n = 90$. Moreover, for the instances with 100 vertices, we improved 7 out of 10 best known solutions. This corresponds to a 0.52% gap improvement for the best known solutions, and 1.60% gap improvement when comparing with [6] (all of the 10 solutions in [6] were improved). The average shaking time (respectively improvement time) is 2122.79 seconds (153.44 seconds) for the group of 70 vertices and 8419.05 seconds (399.01 seconds) for the group of 100 vertices.

The computational results show that the proposed approach gives better results than the existing heuristics. However its processing time is still large and it relies on a mathematical model. More specifically, the shaking time increases from 45.39 seconds (0.15 seconds per iteration) for 20 vertices to 8419.05 seconds (28.06 seconds per iteration) for 100 vertices. Note that the shaking time could be reduced by increasing the parameter α in the model. The solutions of several instances could be improved by increasing the number of iterations as well.

5 Conclusions

In this paper, we have presented a hybrid ILS/VND heuristic for the 1-PDTSP, in which a new way of generating feasible initial solutions using an assignment model is developed, and it is used as major shaking procedure. We also implemented an improvement procedure which combines an auxiliary shaking step and a VND local search. The relaxation of the vehicle capacity constraint is allowed during the improvement procedure. A neighbor solution explored by the assignment model in the major shaking step is considered as a *k-position* neighbor to the incumbent solution, where *k* is the maximum number of customer positions different between two solutions. The improvement procedure uses a vertex-exchange operator (*swap*) in the auxiliary shaking step, and edge-exchange operators (*k-Opt*) in the VND local search.

The proposed approach is evaluated on small and medium benchmark instances with the smallest vehicles capacities for which the problem is known to be harder. The experimental results show that

our method gives almost all optimum solutions on the small instances (49 out of 50). We improved 7 out of 10 for the group of most difficult medium instances when comparing with the best known solutions. However, the running time remains high, especially when considering the shaking step.

As future work, we are interested in reducing the running time of this, especially to address the dynamic version of the problem where the stations' availability can quickly evolve over the time.

On the other hand, a variant of the 1-PDTSP with demand splitting could be also investigated, in which a customer request could be separated into several parts and each part is treated separately by the vehicle. This could model situations where the imbalance at some stations exceeds the vehicle capacity and several operations/visits should be done instead of a single one.

References

[1] H. Hernández-Pérez and J. J. Salazar-González. *The one-commodity pickup-and-delivery travelling salesman problem*. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization-Eureka, You Shrink!*, vol. 2570, pp. 89–104. *Lecture Notes in Computer Science*, Springer, 2003.

[2] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia and G. Laporte. *Static pickup and delivery problems : a classification scheme and survey*. TOP, vol. 15, pp. 1–31, 2007.

[3] M. I. Hosny and C. L. Mumford. *Solving the one-commodity pickup and delivery problem using an adaptive hybrid VNS/SA approach*. In Proceedings of the 11th International Conference on Parallel Problem Solving From Nature (PPSN2010), LNCS, Springer-Verlag, pp. 189-198, 2010.

[4] H.R. Lourenço, O. Martin and T. Stützle. *A beginner's introduction to Iterated Local Search*. In Proceeding of the 4th Metaheuristics International Conference, Porto, Portugal, pp. 1-11, 2001.

[5] N. Mladenović and P. Hansen. *Variable neighborhood search*. Computers & Operations Research; 24: 1097–1100, 1997.

[6] H. Hernández-Pérez, I. Rodríguez-Martín and J. J. Salazar-González. *A hybrid GRASP/VND heuristic for the One-Commodity Pickup-and-Delivery Traveling Salesman Problem*. Computers & Operations Research, 36(5), pp. 1639-1645, 2008.

[7] H. Hernández-Pérez and J.J. Salazar-González. *The one-commodity pickup and delivery traveling salesman problem: Inequalities and algorithms*. Networks, 50(4), 258–272, 2007.

[8] H. Hernández-Pérez and J. J. Salazar-González. *A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery*. Discrete Applied Mathematics, 145:126-139, 2004.

[9] H. Hernández-Pérez and J. J. Salazar-González. *Heuristics for the one-commodity pickup-and-delivery traveling salesman problem*. Transportation Science, 38(2):245-255, 2004.

[10] M. Fischetti and A. Lodi. *Local branching*. Math. Programming 98, 23–47, 2003.

[11] Zhao F., Li S., Sun J. and Mei D. *Genetic algorithm for the one-commodity pickupand-delivery traveling salesman problem.* Computers & Industrial Engineering 56(4), 1642 – 1648, 2009.