RAIRO-Oper. Res. 52 (2018) 1295–1328
https://doi.org/10.1051/ro/2018024

RAIRO Operations Research
www.rairo-ro.org

A SELECTIVE ADAPTIVE LARGE NEIGHBORHOOD SEARCH HEURISTIC FOR THE PROFITABLE TOUR PROBLEM WITH SIMULTANEOUS PICKUP AND DELIVERY SERVICES

Hayet Chentli[1,*], Rachid Ouafi[1] and
Wahiba Ramdane Cherif-Khettaf[2]

Abstract. The *Vehicle Routing Problem with Simultaneous Pickups and Deliveries* (VRPSPD) is a variant of the *Vehicle Routing Problem*. In this variant, an unlimited fleet of capacitated vehicles is used to satisfy both pickup and delivery demands of each customer simultaneously. In many practical situations, such a fleet is costly. The present study extends the VRPSPD by assuming a fixed number of vehicles when the constraint of visiting all customers is relaxed. More specifically, profits are assigned to the customers with the goal of maximizing the difference between collected profits and routing costs. This variant is named *Profitable Tour Problem with Simultaneous Pickup and Delivery services* (PTPSPD). We present a mathematical model run with the CPLEX solver. We also present an extension of the *Adaptive Large Neighborhood Search heuristic* (ALNS) called *selective ALNS* (sALNS). sALNS uses a new operator selection that executes two phases alternately: the random and the score-dependent phases. An appropriate update of scores is employed. Furthermore, sALNS explores missed regions of the search space by evaluating solutions after the destruction step. Finally, we give tuned insertion and removal operators that handle the constraints of the PTPSPD, as well as a new update of temperature, that helps avoiding local optima, in the *Simulated Annealing* embedded in sALNS. sALNS is evaluated on 117 new instances with 50–199 customers. A comparison is made between the components of sALNS, the classical ALNS and a recent ALNS heuristic from the literature. sALNS is also evaluated on some VRPSPD instances from the literature. The computational results show that our heuristic provides good quality solutions in reasonable computing time.

Mathematics Subject Classification. 90B06, 90C11, 90C27, 90-08

Received April 30, 2017. Accepted March 11, 2018.

1. Introduction

Nowadays, there are more and more environmental problems due to the increase of industrial activities. In order to cope with those problems, several countries around the world commit themselves to provide different environmental protection strategies. Among those strategies, new laws, that impose to the companies a

Keywords and phrases: Vehicle routing problem with pickup and delivery, profitable tour problem, reverse logistic, adaptive large neighborhood search, metaheuristics.

[1] Department of Operations Research, USTHB, P.O. Box 32 El Alia, 16111 Bab Ezzouar, Algiers, Algeria.
[2] LORIA, UMR 7503, Lorraine University, Mines Nancy, Nancy, France.

* Corresponding author: chentli.hayet@gmail.com

better management of waste, have been established. Companies are then constrained to analyze, keep track of and maintain the life cycles of their products. Hence, for instance, instead of being thrown out, pallets of transportation (that are used for optimizing the distribution, the loading and the storage of commodities) are recovered and then reused. This results in a single scheme of transportation including both pickup and delivery.

As it is costly to deal with distribution cycles and waste management separately, companies are constrained to optimize their scheme of transportation including both pickups and deliveries. In some cases, transporters are required to visit each customer one and only one time. So, pickup and delivery are performed simultaneously. That can happen, for example, when the customers impose to be visited only once for avoiding excessive mobilization of agents when receiving, stocking and preparing the commodities. In the literature, the problem of simultaneously collecting and delivering products to customers is called *Vehicle Routing Problem with Simultaneous Pickup and Delivery services* (VRPSPD) (see [19]).

The VRPSPD has received a lot of attention from researchers during the last decade. However, in all studied variants, the authors consider the problem with an unlimited fleet of vehicles. This assumption does not describe properly the reality. Indeed, such a fleet is costly.

In the present paper, we propose a new realistic variant of the VRPSPD. This variant is named *Profitable Tour Problem with Simultaneous Pickup and Delivery services* (PTPSPD). From the technical point of view, (i) The PTPSPD is a variant of the *Vehicle Routing Problem* (VRP) that uses a homogeneous capacitated and limited fleet of vehicles, which are parked at a single depot, in order to simultaneously perform the pickup and the delivery operations of each customer. (ii) In a PTPSPD, profits are assigned to the customers in such a way that, the vehicle routes, which begin and end at the depot, ensure that each customer is visited at most once (some customers may not be visited at all) and by a unique vehicle. (iii) The deliveries are transported from the depot to the customers and the pickups are transferred from the customers to the depot. (iv) It is forbidden to exceed the capacity of the vehicles. (v) The objective of the problem is to maximize the difference between the sum of collected profits and the traveling costs.

The PTPSPD can be encountered in the soft drinks industries where each grocery store (or customer) receives filled bottles and gives back empty ones. Therefore, the transporters have to distribute (deliver) filled bottles and collect (pick up) empty ones in order to reuse them thereafter. Another application can be found in the distribution and the retrieving of certain machines at the end of their life cycles, in order to repair or recycle them.

The PTPSPD can also be viewed as an extension of the *Capacitated Profitable Tour Problem* (CPTP), in which only pickup operations are required (see [1]). We can easily remark that the CPTP is a special case of our problem. Actually, if deliveries are set to zero, we obtain a CPTP. Since Jepsen *et al.* [13] proved the NP-Hardness of the CPTP, we may say that the PTPSPD is NP-Hard too.

In order to get efficient solutions to our problem, we propose an extension of the *Adaptive Large Neighborhood Search heuristic* (ALNS) called *selective ALNS* (sALNS). sALNS deals with the selective aspect of PTPSPD as well as with the management of simultaneous pickup and delivery. To assess the performance of the proposed approach, we contrast our results with those obtained by three methods. First, we perform a run using CPLEX 12.2. Then, we implement the classical version of ALNS [22]. Finally, we run another variant of sALNS denoted *Li*. The latter variant includes the removal/insertion operators of a recent ALNS algorithm proposed by Li et al. [17] to solve another *Pickup and Delivery Problem* (PDP) with profits. The numerical solutions of CPLEX, ALNS, Li and sALNS are compared on new generated instances. Furthermore, we evaluate sALNS on some VRPSPD instances from the literature. Our experiments show that sALNS performs well with respect to solution quality and computing time.

The remainder of the present paper is organized as follows: In Section 2, we introduce the literature related to the PTPSPD. This is followed by a presentation of the mathematical model considered in our work in Section 3. Section 4 introduces the proposed algorithm emphasizing the main differences with ALNS heuristics commonly used in the literature. In Section 5, we give the computational results. Finally, Section 6 is devoted to the conclusion.

2. Literature review

The VRPSPD was first introduced by Min [19]. In that paper, the author described a real-life problem of simultaneously collecting and delivering books to 22 libraries, departing from and ending at a central library. Min has used a two-phase method to solve the problem. First, the customers were clustered and each cluster was assigned to a vehicle. After that, for each vehicle, a solution was obtained by solving a *Traveling Salesman Problem* (TSP). To ensure the feasibility of the solutions according to capacity constraints, arcs in which the capacity was exceeded were penalized.

After the introduction of the VRPSPD, many heuristic approaches have been proposed to solve this problem. One can cite for example, *Tabu Search* heuristics presented in [20, 32, 33]. In the first paper, the *Tabu Search* uses one intra- and four inter-route(s) operators, which are chosen according to two different strategies. The approach was applied to both the VRPSPD and the VRPSPD with time limit. In the second paper, a *Guided Local Search* is combined with the *Tabu Search*. In the third paper, an *Adaptive Memory* methodology is employed to save the best solution sequences. In the three papers the authors tested their approaches on instances with 50–400 customers. One can remark that the *Tabu Search* presented in [20] often chooses more vehicles than necessary. From the results provided in [33], one can see that the approach proposed in [32] is generally trapped in local optima.

Another heuristic that has been successfully applied to the VRPSPD is the *Adaptive Large Neighborhood Search* proposed in [25]. The latter heuristic produced many new best solutions with small numbers of vehicles at publication time for the the VRPSDP, the VRPSPD with time limit and several other variants of *Vehicle Routing Problems with Pickup and Delivery*. The heuristic is tested on 338 instances from the literature with up to 500 customers.

In [3, 14], two different versions of the *Ant Colony System* are developed. The two approaches are tested on instances of both VRPSPD and VRPSPD with time limit. In [3], the computational experiments are made on benchmark instances with 22–400 customer. While in [14], the number of customers varies between 50 and 199. The *Ant Colony System* implemented in [3] performs well on instances of VRPSPD with time limit but several VRPSPD benchmark solutions are not reached. On the other hand, the *Ant Colony System* proposed in [14] provides good quality solutions for both VRPSPD and VRPSPD with time limit. However, some best known solutions from the literature are not reached. No result is provided for instances with 100–400 customers.

In [28], P-ILS-RVND, a parallel algorithm using an *Iterative Local Search* combined with a *Variable Neighborhood Descent*, is implemented. That algorithm is tested on benchmark instances of the VRPSPD with 50–400 customers. P-ILS-RVND almost always provides the best solutions among all the literature approaches for the VRPSDP. However, the algorithm is run on a cluster with a multi-core architecture using up to 256 cores. Hence, P-ILS-RVND cannot be objectively compared with the other approaches.

A *Local Search* that uses a special movement encoding as well as a promise concept is given in [31]. The approach generally performs well on VRPSPD instances with 100–400 customers. However that *Local Search* is sometimes tapped in local optima. Tests on benchmark instances with 50–199 customers are not presented. Finally, a parallel *Simulated Annealing* algorithm is proposed in [21]. Computational experiments are made using benchmark instances with 50–400 customers. The parallel *Simulated Annealing* algorithm provides solutions in a small amount of computing time. However, many benchmark solutions are not reached.

A summary of the above described paper can be found in Table 1. For each approach, n refers to the number of customers in the instances solved.

By contrast with the VRPSPD, the CPTP has not been studied very much. Archetti *et al.* [1] first introduced the CPTP and implemented three approaches to solve this problem namely, *Variable Neighborhood Search* (VNS), *Tabu Feasible* (TF) and *Tabu Admissible* (TA). TF accepts only feasible solutions while TA allows the visit of unfeasible solutions but favors feasible ones. The VNS algorithm uses, on the other hand, the TF method over a small number of iterations.

Some researchers have also tackled PDPs with profits. In [9], the authors studied the *Single Vehicle Routing Problem with Deliveries and Selective Pickups* (SVRPDSP), a problem that requires a single vehicle to visit all delivery customers and relax this constraint for pickups. The objective of this problem is to minimize the

TABLE 1. Summary of VRPSPD literature approaches.

Authors	n	Approach
Min [19]	22 customers	Clustering first routing second + *3-Opt*
Ropke and Pisinger [25]	50–199 customers	Adaptive Large Neighborhood Search
Montané and Galvao [20]	50–400 customers	Tabu Search
Zachariadis *et al.* [32]	50–400 customers	Tabu Search + Guided Local Search
Zachariadis *et al.* [33]	50-400 customers	Tabu Search + Adaptive Memory methodology
Çatay [3]	22–400 customers	Ant Colony System
Subramanian *et al.* [28]	50–400 customers	Parallel algorithm using an Iterative Local Search combined with a Variable Neighborhood Descent
Zachariadis and Kiranoudis [31]	100–400 customers	Local Search using a special movement encoding as well as a promise concept
Mu *et al.* [21]	50–400 customers	Parallel Simulated Annealing
Kalayci and Kaya [14]	50–199 customers	Ant Colony System

difference between routing costs and collected profits. Several construction and improvement heuristics together with a *Tabu Search* algorithm are proposed to solve the SVRPDSP. Experimental tests are performed on instances with 15–100 customers.

In [10], the SVRPDSP is extended by including time window constraints. This variant of the problem is called *Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows* (VRPDSPTW). A *branch-and-price* algorithm is proposed for the VRPDSPTW. Tests are performed on instances with up to 100 customers.

In [29], a *Memetic Algorithm* is implemented for the *Selective Pickup and Delivery Problem* (SPDP). The problem consists of supplying all delivery customers from pickup customers when minimizing routing costs. In this variant, constraint of visiting all pickup customers is relaxed. The proposed *Memetic Algorithm* uses a new representation of solutions including all pickups and deliveries. It also employs genetic operators as well as a modified *2-Opt* procedure, both designed for the new representation of the SPDP. Computational results are reported for instances with up to 454 customers.

In [12], a *Greedy Randomized Adaptive Search Procedure* with *Path Relinking* is developed for the SPDP. The proposed approach is tested on the same set of instances as the one used in [29]. The obtained solutions are generally better than those of the *Memetic Algorithm* given in [29].

In [23], a randomized search and a greedy heuristic are proposed for the *Profit-Maximizing Pickup and Delivery Selection Problem* (PPDSP). The goal of a PPDSP is to maximize the difference between total collected profits and traveling costs. In the PPDSP, all customers are optional and have both pickup and delivery demands. In addition, the available fleet of vehicles is heterogeneous. The PPDSP involves bounds on total trip time as well as time window and capacity constraints. The computational results provided in [23] are performed on instances with 10–500 customers.

The *Pickup and Delivery Problem with Time Windows, Profits, and Reserved Requests* (PDPTWPR) is introduced in [17]. In a PDPTWPR, each vehicle (or carrier) has some reserved and some optional customers. The visit of the reserved customers is mandatory while optional customers may be visited by other carriers or not visited at all. Each customer has both pickup and delivery demands associated with an origin, a destination, two time windows, a service time and a payment. The aim of the PDPTWPR is to build, for each carrier, a feasible route including all mandatory customers and possibly some optional customers, in order to maximize the difference between the payment of the visited customers and the transportation cost. The authors in [17] proposed a multi-start ALNS heuristic to solve the problem. Computational results are conducted on instances with 10–100 customers.

Finally, in [7], two variants of the *General Variable Neighborhood Search* heuristic are developed for the *Multi-Vehicle Profitable Pickup and Delivery Problem* (MVPPDP). In a MVPPDP, a fleet of vehicles is used to transport commodities from pickup customers to the corresponding delivery customers. In addition, some travel

time limits are imposed. The two variants of the *General Variable Neighborhood Search* are compared with a *Guided Local Search* algorithm on instances with up to 1000 customers.

Note that, none of the above mentioned problems considers the case where both pickup and delivery operations are optional in such a way that, if a customer is selected, the two operations have to be performed simultaneously. A summary of the above described PDPs with profits and their approaches is given in Table 2. The main differences between literature problems and the PTPSPD are presented in column *difference*. For each problem, n stands for the customer number in the instances solved using the proposed approach.

One can remark that the *Large Neighborhood Search heuristic* (LNS) (and, in particular, ALNS) has been implemented several times for PDPs. Indeed, in addition to the previously cited works, one can cite for example the hybrid two-stage approach combining LNS with a *Simulated Annealing* algorithm presented in [2]. This approach is used to solve the *Pickup and Delivery Vehicle Routing Problems with Time Windows and Multiple Vehicles*. Another example can be found in the combination of LNS with a *Tree Search* heuristic presented in [18] for the *Vehicle Routing Problem with Pickup and Delivery and Three-dimensional Loading Constraints*. In [24], an ALNS heuristic is developed for the *Vehicle Routing Problem with Pickup and Delivery and Time Windows*. In [25], an ALNS heuristic is proposed for solving a large class of *Vehicle Routing Problems with Backhauls*. Finally, in [8], an ALNS heuristic is proposed for the *Pickup and Delivery Problem with Time Windows and Scheduled Lines*.

As the PDPs with profits are relatively new problems that have not been studied enough to see which heuristic performs better on them, and as all the heuristics proposed for the PDPs with profits are different, we decide to implement an ALNS heuristic for our problem. Our choice is motivated by the fact that ALNS performs well on the VRPSPD, on some variant of the PDPs with profits and on several variants of PDPs.

3. Mathematical formulation of the PTPSPD

In the present Section, we propose a mathematical formulation for the PTPSPD. This formulation is inspired by the model proposed in [6] for the VRPSPD, which is a commodity flow model. In what follows, we give a description of our model.

Let n, m and Q be the number of customers, the number of vehicles and the capacity bound respectively. Let us define $N = \{1, \ldots, n\}$ as the set of customers. Let 0 refers to the depot. The PTPSPD can be defined on a complete undirected and weighted graph $G = (V, E, c)$, where $V = N \cup \{0\}$ is the set of nodes, E is the set of edges (route sections between elements of V), and $c : E \to \mathbb{R}^+$ is a weight function, that assigns to each edge $(i, j) \in E$ a traveling cost $c_{ij} \in \mathbb{R}^+$ between nodes i and j. Note that, the traveling cost matrix between all pair of nodes $i \in V$ and $j \in V$ is symmetric. Hence, for all $(i, j) \in E$, $c_{ij} = c_{ji}$. For each customer $i \in N$, let $p_i \geq 0$, $d_i \geq 0$ and $pr_i \geq 0$ be the pickup demand, the delivery demand and the profit of i respectively. Note that, a customer with no pickup ($p_i = 0$) and no delivery ($d_i = 0$) demands (with or without profits) cannot exist as we are studying a pickup and delivery problem. Thus, the decision variables can be defined as follows: x_{ij}, $(i, j) \in E$, is a binary variable that takes the value 1 if a vehicle visits node j directly after node i and 0 otherwise, t_i, $i \in N$, is another binary variable that takes the value 1 if the node i is included in the solution and 0 otherwise. Finally, y_{ij} and z_{ij}, with $(i, j) \in E$, are positive real variables that stand for collected demands through the predecessors of node i (i included) which are transported on (i, j) and amount of load to be delivered to the successors of i that is transported on (i, j) respectively. As a result, we obtain the following mixed integer linear programming model:

$$\max f(x) = \sum_{i \in N} pr_i \cdot t_i - \sum_{i,j \in V} c_{ij} \cdot x_{ij} \tag{3.1}$$

$$\sum_{i \in V} y_{ji} - \sum_{i \in V} y_{ij} = p_j \cdot t_j \quad \forall\, j \in N \tag{3.2}$$

$$\sum_{i \in V} z_{ij} - \sum_{i \in V} z_{ji} = d_j \cdot t_j \quad \forall\, j \in N \tag{3.3}$$

TABLE 2. Summary of PDPs with profits literature approaches.

Problem	Authors	Difference	Approach	n
Single Vehicle Routing Problem with Deliveries and Selective Pickups (SVRPDSP)	Gribkovskaia *et al.* [9]	Pickup and delivery operations may be performed separately A single vehicle is required to fulfill all delivery demands and some pickup ones	Several construction and improvement heuristics + Tabu Search	Up to 100
Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows (VRPDSPTW)	Gutierrez-Jarpa *et al.* [10]	Same as SVRPDSP + time window constraints	Branch-and-Price	Up to 100
Selective Pickup and Delivery Problem (SPDP)	Ting and Liao [29] Ho and Szeto [12]	All delivery customers are supplied from pickup customers	Memetic Algorithm GRASP with Path Relinking	Up to 454 Up to 454
Profit-Maximizing Pickup and Selection (PPDSP)	Qiu *et al.* [23]	The fulfillment of pickup and delivery operations simultaneously is not required Bounds on total trip time + time window and capacity constraints Heterogeneous fleet of vehicles	Randomized search + greedy heuristic	Up to 500
Pickup and Delivery Problem with Time Windows, Profits, and Reserved Requests (PDPTWPR)	Li *et al.* [17]	Time window constraints Each vehicle has some reserved (and mandatory) and some optional customers Each customer has both pickup and delivery demands associated with an origin, a destination, two time windows, a service time and a payment	ALNS heuristic	Up to 100
Multi-Vehicle Profitable Pickup and Delivery Problem (MVPPDP)	Gansterer *et al.* [7]	Commodities are transported from a selection of pickup customers to the corresponding delivery customers Time limit	General Variable Neighborhood Search	Up to 1000

$$y_{ij} + z_{ij} \leq Q \cdot x_{ij} \quad \forall\, i \in V;\ \forall\, j \in V; \tag{3.4}$$

$$t_i = \sum_{j \in V} x_{ij} \quad \forall\, i \in N; \tag{3.5}$$

$$\sum_{j \in V} x_{ij} = \sum_{j \in V} x_{ji} \quad \forall\, i \in V; \tag{3.6}$$

$$\sum_{i \in N} x_{0i} \leq m \tag{3.7}$$

$$x_{ij} \in \{0, 1\} \quad \forall\, i \in V; \forall\, j \in V; \tag{3.8}$$

$$t_i \in \{0, 1\} \quad \forall\, i \in N; \tag{3.9}$$

$$y_{ij} \in \Re^{+} \quad \forall\, i \in V;\ \forall\, j \in V; \tag{3.10}$$

$$z_{ij} \in \Re^{+} \quad \forall\, i \in V;\ \forall\, j \in V. \tag{3.11}$$

The objective function is given by formula (3.1). Constraints (3.2) and (3.3) stand for the pickup and delivery demand satisfaction. Constraints (3.4) ensure that the capacity of the vehicles is not exceeded. Constraints (3.5) ensure that if there is an edge leaving a customer i then i is necessarily included in the solution. Constraints (3.6) are the flow conservation constraints. Constraints (3.7) enforce the number of vehicles leaving the depot to be less than or equal to the number of available vehicles. Constraints (3.8)–(3.11) define the decision variables. Moreover, constraints (3.5), (3.8) and (3.9) guarantee that each customer is visited at most once. Finally, constraints (3.2), (3.3), (3.5), (3.6), (3.8) and (3.9) guarantee sub-tour elimination by assuming that, for each customer i, p_i and d_i cannot be both equal to 0. A sub-tour elimination proof is provided in Appendix A.

One can remark that there are many similarities between our model and the one proposed in [6]. Indeed, if all the profits are fixed to a very large number in comparison with the traveling costs (traveling costs can be equivalent to distances), the optimal solution will contain all the customers. When comparing different solutions containing all the customers, the sum of customer profits ($\sum_{i \in N} pr_i \cdot t_i$) can be considered as a constant value. Hence, the objective will be the maximization of ($-\sum_{i,j \in V} c_{ij} \cdot x_{ij}$), which is equivalent to the minimization of the traveling costs.

Note that, if all the profits are fixed to 0, our model cannot be equivalent to the one proposed in [6]. Indeed, when the profits are set to 0, the optimal solution of the problem will be a solution with no customer, as there are no constraints enforcing the visit of each customer at least once.

4. The proposed methodology

To attempt the resolution of the PTPSPD, we have implemented an extension of the *Adaptive Large Neighborhood Search* heuristic called sALNS. The decision to adopt this methodology was guided by the fact that the ALNS is an efficient method for solving several variants of *Vehicle Routing Problems* in general and *Pickup and Delivery Problems* in particular (see [25] for instance).

The ALNS heuristic is based on the destruction/reconstruction principle. Actually, ALNS uses several operators called destroy (or removal) operators for deleting some of the customers from a current solution. All the unrouted customers are stored in a so-called *request bank*. Then, ALNS tries to reinsert the customers located in the request bank in better positions with construction (or repair, or also insertion) operators. At each iteration of ALNS, the heuristic chooses one destroy and one repair operators according to their scores in such a way that, efficient operators (with high scores or performance) have more chances to be selected. The resulting solution is after that accepted or not according to some criteria and the process continues until the stopping criterion is met. For more details on the ALNS methodology, we refer the reader to [22, 25].

Algorithm 4.1 describes the pseudo-code of sALNS. The details of the components are presented in the next subsections. To facilitate the comprehension, we provide, in Table 3, a description of each parameter and variable used in Algorithm 4.1.

Algorithm 4.1. sALNS

1: **Inputs:**
A PTPSPD instance
A list L_{rem} of removal operators
A list L_{ins} of insertion operators
$lev = 0$; $itr = 0$; $T = 1$;
2: **Outputs:**
The best solution found
3: **Initialize:**
Generate an initial solution with H_{const} (Sect. 4.1)
4: **while** $itr < IS$ **do**
5: **if** $lev = ml$ ▷ beginning of evaluation phase
6: Reset scores;
7: $lev = 0$;
8: **end if**
9: Randomly choose r from $\{a_1, \ldots, a_k\}$;
10: **if** $lev < 5 \cdot nb_{rh}$ **then** ▷ evaluation phase
11: **if** L_{rem} is empty **then**
12: Fill L_{rem} with the removal operators;
13: **end if**
14: Choose a removal operator R at random from L_{rem} and delete it from this list;
15: **else** ▷ application phase
16: Choose a removal operator R according to the actual scores; (see Sect. 4.3)
17: **end if**
18: Generate a new solution by deleting r customers from the current solution using R;
19: Accept or not the solution according to the acceptance criteria; (see Sect. 4.2)
20: **if** the new solution is better than the current one **then**
21: Update the score of R; (see Sect. 4.3.1)
22: **end if**
23: **if** $lev < 5 \cdot nb_{ih}$ **then** ▷ evaluation phase
24: **if** L_{ins} is empty **then**
25: Fill L_{ins} with the insertion operators;
26: **end if**
27: Choose an insertion operator I at random from L_{ins} and delete it from this list;
28: **else** ▷ application phase
29: Choose an insertion operator I according to the actual scores; (see Sect. 4.3)
30: **end if**
31: Generate a new solution by applying I to the current solution;
32: Accept or not the solution according to the acceptance criteria; (see Sect. 4.2)
33: **if** the new solution is better than the current one **then**
34: Update the score of I; (see Sect. 4.3.1)
35: **if** the score of R has not been updated **then**
36: Update the score of R; (see Sect. 4.3.1)
37: **end if**
38: **end if**
39: $T = T \cdot c$; $lev++$; $itr++$;
40: **if** $T < T_0$ **then**
41: $T = itr \cdot 10$
42: **end if**
43: **end while**

sALNS begins by calling the construction heuristic H_{const} (detailed in Sect. 4.1), which generates a feasible solution. The main loop is executed during IS iterations. Each ml iterations, the scores of the removal and insertion operators are reset (lines 5–8). The main algorithm chooses the number r of customers to delete. This number is randomly chosen from the user-defined set of k elements $\{a_1, a_2, \ldots, a_k\}$ where $a_1, a_2, \ldots, a_k \in \mathbb{N}$ and

TABLE 3. Description of parameters and variables used in Algorithm 4.1.

Parameter/variable	description
L_{rem}	List of removal operators
L_{ins}	List of insertion operators
lev	A counter used to determine the current phase of sALNS (evaluation or application)
itr	The current number of iterations
T	The current temperature
IS	The total number of iterations in the main loop of sALNS
ml	The number of iterations performed before the scores are reset, defined by the user
r	The number of removed customers, $r \in \{a_1, a_2, \ldots, a_k\}$, where $a_1, a_2, \ldots, a_k \in \mathbb{N}$ and k are user-defined parameters
nb_{rh}	The number of removal operators
R	The chosen removal operator
nb_{ih}	The number of insertion operators
I	The chosen insertion operator
c	The cooling rate, defined by the user
T_0	The temperature threshold, defined by the user

k is a user-defined parameter (line 9). A removal operator is selected (lines 10–17). A list L_{rem} that initially contains all removal operators is used to make this selection. More specifically, during an evaluation phase (that lasts $5 \cdot nb_{rh}$ iterations for the removals, where nb_{rh} is the number of removals), sALNS chooses one operator that has not been chosen yet from L_{rem} and then deletes this operator from that list. In the case where L_{rem} is empty, all removals are included again in L_{rem} (line 12). On the other hand, in an application phase, the algorithm chooses a removal operator according to its actual score in such a way that, efficient operators (with high scores) have more chances to be selected (see Sect. 4.3). The selected removal operator is used to generate a new solution by deleting r customers from the current solution (line 18). On the next line, this new solution is accepted or not according to the acceptance criteria (see Sect. 4.2). If the new solution is better than the current one then, the score of the selected removal is updated (see Sect. 4.3.1) as shown in lines 20–22. The selection of an insertion operator is quite similar to the selection of a removal operator (lines 23–30). The only difference is that the evaluation phase lasts $5 \cdot nb_{ih}$ iterations, where nb_{ih} is the number of insertions. L_{ins} stands for the list of insertion operators. On line 31, a new solution is generated by adding some customers (possibly no customer) to the current solution. Next, the new solution is accepted or not according to the acceptance criteria (see Sect. 4.2). If the new solution is more profitable than the current one, the score of the insertion operator is updated (lines 33–38). In addition, if the score of the removal operator has not been updated in the current iteration, the procedure updates this score too (Sect. 4.3.1). The current temperature T, which is initially set to 1, is decreased according to the expression $T = T \cdot c$ using the cooling rate $c \in]0, 1[$ (line 39). If the temperature threshold $T_0 \leq 1$ is reached (lines 40–42), the value of the current temperature T is increased according to the Formula $T = itr \cdot 10$, where itr is the current number of iterations performed (which is incremented at each iteration).

As said in the literature review, in [17], a new variant of ALNS is developed for the PDPTWPR. In what follows, we highlight the main characteristics of our heuristic that do not exist in both of the classical ALNS proposed in [22] and the ALNS proposed in [17]. (i) sALNS evaluates the solutions and updates the scores of the operators after both removals and insertions (see Sect. 4.2); (ii) sALNS alternates two different phases for selecting removal/insertion operators (see Sect. 4.3); (iii) sALNS uses distinct combinations of removals and insertions handling profits and load fluctuations in their selection criteria (see Sect. 4.4); (iv) sALNS repeatedly restarts using a higher starting temperature to add more diversification to the search (see Sect. 4.5.1). In

addition, the diversification mechanism *meta-destroy* proposed in [17] can be considered as a special case of our operator evaluation (see Sect. 4.5.2).

In the following sections, we give more details about our components, namely the construction heuristic, the solution evaluation, the operator selection, the removal/insertion operators and the diversification strategies.

4.1. Construction heuristic

In order to get an initial solution for our problem, we use a multi-start sequential heuristic based on the I1 heuristic (see [27]), which was initially implemented for the *Vehicle Routing Problem with Time Window* (VRPTW). The pseudo-code of I1 is given in Algorithm 4.2.

Algorithm 4.2. I1

1: **Inputs:**
   A PTPSPD instance
   A list L_{unr} of unrouted customers
   The number $nbRoutes = 0$ of routes in the current solution
2: **Outputs:**
   A feasible solution
3: **while** $nbRoutes <$ number of vehicles
4: Create a new route;
5: $nbRoutes++$;
6: Fill the new route with a seed customer;
7: **while** $\exists u \in L_{unr}$ that can be inserted without leading to an infeasible solution
8: Evaluate the insertion of all the unrouted customers $u \in L_{unr}$ into the current route;
9: Select, for each u, the best insertion position according to criterion $cr_1(i,u,j)$;
10: Select the best position of the best customer u^* according to criterion $cr_2(i,u,j)$;
11: Insert customer u^* in its best position within the current route;
12: Update L_{unr};
13: **end while**
14: **end while**

The I1 heuristic starts with an empty route. This route is first filled with a seed customer. Then, the insertions of the remaining customers into the route are evaluated. I1 selects the best insertion position for each unrouted customer u between two consecutive customers i and j according a first criterion denoted $cr_1(i,u,j)$. Finally, I1 chooses the insertion that optimizes a second criterion $cr_2(i,u,j)$ among the best positions so far obtained. This process is repeated until no unrouted customer can be inserted into the route without leading to infeasibility. The other routes of the solution are constructed similarly. In the present paper, we adjust the selection criteria to the studied problem. Besides considering the traveling costs, our criteria also take into account the profits, when choosing the customers to insert. The criterion based on time windows is neglected.

Let $(i_0, i_1, \ldots, i_h)$ be the current route, where i_ρ refers to the customer in the ρth position of the route if $\rho \notin \{0, h\}$, and to the depot otherwise ($i_0 = i_h = 0$). I1 computes for each unrouted customer u its best feasible insertion position within the current route according to expressions (4.1)–(4.4), with (w.l.o.g) c_{ij} stands for the traveling cost between i and j, pr_u is the profit of customer u, and $\alpha_1, \alpha_2, \mu \geq 0$ are user-defined parameters, where $\alpha_1 + \alpha_2 = 1$ and $\mu \in [0, 3]$.

$$cr_1(i(u), u, j(u)) = \max\{cr_1(i_{\rho-1}, u, i_\rho), \rho = 1, \ldots, h\}; \tag{4.1}$$

$$cr_1(i,u,j) = \alpha_1 \cdot cr_{11}(i,u,j) - \alpha_2 \cdot cr_{12}(i,u,j); \tag{4.2}$$

$$cr_{11}(i,u,j) = pr_u; \tag{4.3}$$

$$cr_{12}(i,u,j) = c_{iu} + c_{uj} - \mu \cdot c_{ij}. \tag{4.4}$$

To avoid postponing the insertion of difficult customers to the last iterations, a second criterion $cr_2(i,u,j)$ is used. $cr_2(i,u,j)$ forces somehow the insertion of the farthest unrouted customer from the depot. More precisely, among all the unrouted customers, the optimization of the second criterion leads to the selection of the best customer u^* according to expressions (4.5) and (4.6). In (4.6), c_{0u} stands for the traveling cost between depot and customer u, $\lambda \in [0,1]$ is a user-defined parameter that sets the importance of c_{0u} when choosing the customer to insert.

$$cr_2(i(u^*),u^*,j(u^*)) = \max\{cr_2(i(u),u,j(u)), u \text{ unrouted}\}; \tag{4.5}$$

$$cr_2(i(u),u,j(u)) = \lambda \cdot c_{0u} + cr_1(i(u),u,j(u)). \tag{4.6}$$

In addition to the seed's selection based on the depot's farthest neighbor with the biggest profit $max_u\{pr_u + c_{0u}, u \text{ unrouted}\}$, we have added two types of seed's selection. In the first type, we choose the customer with the biggest value of $pr_u - (c_{0u} + c_{u0})$ while, in the second type, we choose the customer with the biggest profit pr_u. A user-defined parameter denoted *type_Select* is provided to determine which seed's selection will be performed. *type_Select* can take the values 1 for the depot's farthest neighbor with the biggest profit, 2 for the biggest value of $pr_u - (c_{0u} + c_{u0})$, and 3 for the biggest profit.

Our heuristic is executed 10 times, each time with a different set of parameter values for α_1, α_2, μ, λ and *type_Select*. The aim is to have different initial solutions by changing parameters in the construction heuristic. The best encountered solution is considered as starting point of the main algorithm. Note that, if the parameters α_1, α_2, μ, λ and *type_Select* are set to $\frac{1}{2}$, $\frac{1}{2}$, 1, 0 and 2 respectively, I1 inserts, among the unrouted customers, those that lead to the biggest value of the PTPSPD objective function. Note that, this cannot be true if the terms used in I1 were normalized.

4.2. Solution evaluation

Because of the selective aspect of the PTPSPD, a "destroyed" solution (obtained by a removal operator) remains feasible and can be more profitable than a "constructed" solution (obtained by an insertion operator). Indeed, if we remove some customers from a given feasible solution, no constraint of the problem will be violated. For this reason, and contrary to other ALNS heuristics from the literature, sALNS evaluates the solutions and updates the scores of the operators even after removal operators. Note that, in both the classical ALNS and the ALNS proposed in [17], solutions from this search space area would be missed. The evaluation of a solution is done using the *Simulated Annealing* (see [4, 15]) principle. Hence, a solution is accepted if it is better than the best or the current solutions or, if the *metropolis rule* is satisfied. In other words, bad solutions are accepted with the probability given by formula (4.7).

$$\exp^{\left(f(x')-f(x)\right)/T} \tag{4.7}$$

where $f()$ stands for the objective function, x' and x are the new and the current solutions respectively and $T > 0$ denotes the current temperature. Note that, the metropolis rule is only applied after insertion operators because this leads to better results. If a bad solution is not accepted, sALNS continues with the previously accepted solution (not necessarily the best). At each iteration, the current temperature T is decreased using the cooling rate $c \in]0,1[$ according to the expression $T = T \cdot c$.

4.3. Operator selection

Throughout the search process, sALNS uses alternately two phases to allow a good selection of the operators namely, the *evaluation* and the *application phases*. In each evaluation phase, the operators are selected randomly regardless of their performance (or score). At the beginning of this phase, the scores of all operators are reset and then recalculated. In each application phase, the selection of the operators relies on their previously computed scores. The scores are also computed during the application phases. Contrary to other ALNS heuristics from

the literature, sALNS does not divide the search into segments (given number of iterations in which the new scores are computed) and wait for a current segment to end before using the new scores to select operators. Instead, and more specifically in an application phase of sALNS, the scores obtained in previous iterations (including both the previous evaluation phase and the current application phase) are considered for the operator selection.

4.3.1. Update of scores

In both evaluation and application phases, the scores are computed similarly. This computation differs from other score updates commonly used in the literature. More specifically, a vector *vec* is used to save the scores. At the beginning of the search, this vector is empty and each time an operator performs well (provides a better solution than the actual one), the name of this operator is considered as a new element of *vec*. This is equivalent to update the score of the studied operator by incrementing it. Hence, when sALNS has to select an operator depending on its previous scores, a number *rdm* is randomly chosen from the set $\{0, \ldots, s_{vec} - 1\}$ where s_{vec} refers to the number of non-empty elements in *vec*. The name of the selected operator is the *rdm*th element of *vec*. As a result, operators with big scores have more chances to be selected. Remark that the scores of removal and insertion operators are updated separately. More specifically, each time a removal performs well, its score is updated. When examining a solution after an insertion operator, we distinguish two cases. In the first case, the solution has already been improved using the removal operator. Here, we only update the score of the insertion operator. However, if the removal operator does not improve the current solution but the insertion operator does, the scores of both operators are updated. This different score updating between removals and insertions is due to the fact that we do not know which operator performed better and was successful in the achievement of the improved solution.

4.3.2. Phase duration

The evaluation phases are executed for a given number of iterations denoted *eval*, such that $eval = nb_H \cdot l$, where l is a constant value fixed by the user, and nb_H is the number of used operators. In the calculations, nb_H is set to $nb_{rh} = 6$ for the removals (nb_{rh} is the number of removal operators used in sALNS) and to $nb_{ih} = 4$ for the insertions (nb_{ih} is the number of insertion operators used in sALNS). After each evaluation phase, sALNS moves to an application phase. Each *ml* iterations the heuristic goes back again to an evaluation phase. The parameter *ml* is set by the user.

4.4. Removal/insertion operators

In order to create an adapted approach for our problem, we tune some operators of the classical ALNS. In the following, we give a description of the operators used in our heuristic.

4.4.1. Removal operators

The removal operators remove a given number of customers (denoted r) from a current solution. sALNS uses six removal operators. The latter are inspired from the removals presented in [22]. They consist in the *random removal*, the *worst removal*, the *related removal*, the *historical node-pair removal*, the *historical request-pair removal* and the *cluster removal*. The selection criteria of the *related removal* and the *cluster removal* are modified in the present paper. For the other removal operators, the objective function of the PTPSPD is considered instead of the minimization of traveling cost. In [22], two randomization control parameters are introduced. The first (denoted p_w) controls the *worst removal*, while the second (denoted p_r) controls the *related removal*. For small values of a randomization control parameter, the involved removal deletes the best customer in term of removal criterion. While, for larger values of a randomization control parameter, the involved removal deletes low quality customers in term of removal criterion. Hence, big values of a randomization parameter involve

more diversification. In our work, the randomization control parameter is considered for all the removals. Details of our removal operators are given in what follows.

Random removal: This operator randomly removes r customers from the solution.

Worst removal: This operator removes the customers whose deletion produces the smallest decrease in the objective function. The idea behind that is either to reinsert the deleted customers in more advantageous positions or to exclude them completely from the solution. The randomization control parameter of the worst removal is denoted p_w.

Related removal: This operator aims at removing customers that are somehow similar in order to interchange them easily by an insertion operator, generating thereby a new solution. In the present paper, the similarity (that can be called relatedness) is computed according to formula (4.8)

$$r_{ij} = |pr'_i - pr'_j| + c'_{ij} \tag{4.8}$$

where r_{ij} denotes the relatedness between customers i and j. For a given customer i, $pr'_i = pr_i/pr_{\text{biggest}}$, where pr_i is the profit of i and pr_{biggest} stands for the biggest profit in the studied instance. $c'_{ij} = c_{ij}/c_{\max}$, where c_{ij} is the traveling cost between customers i and j and $c_{\max}$ stands for the longest arc in the studied instance. Note that pr'_i, pr'_j, and c'_{ij} belong to $[0, 1]$ for each customer i and j. This normalization allows the use of comparable values for profits and traveling costs. Therefore, customer j is related to customer i if r_{ij} is relatively small. This relatedness favors the deletions of customers that are located not too far from each other and that have almost the same profits. The randomization control parameter of the *related removal* is denoted p_r.

Historical node-pair removal: This operator makes use of historical information when removing a customer. Actually, a value is associated to each pair of customers. This value represents the objective function value of the best solution found so far including those two customers in consecutive positions. For a given customer i, the operator calculates the sum of the values associated to i and each one of the other customer. The r customers with the lowest sums are removed. The randomization control parameter of the *historical node-pair removal* is denoted p_{hn}.

Historical request-pair removal: This operator keeps track of the historical success of inserting a pair of customers in the same route. Actually, the top-100 best solutions (according to the objective function value) observed so far are saved. For each pair of customers i and j, the operator computes the number of times $nbTimes_{ij}$ in which the two customers have been visited by a same vehicle in the top-100 best solutions. $nbTimes_{ij}$ is considered as the relatedness between i and j, such that the bigger is $nbTimes_{ij}$ and the more related are i and j. The deletion is performed in the same manner as in the related removal but with the new relatedness. The randomization control parameter of the *historical request-pair removal* is denoted p_{hr}.

Cluster removal: This operator removes r customers from the solution. The first studied route is randomly selected. Then, Kruskal's algorithm for the minimum spanning tree (MST) problem [16] is used to define the customers to remove from that route. In the *cluster removal*, Kruskal's algorithm determines the MST on the sub-graph defined by the nodes and the arcs of the current route. The weight of any arc (weights are used to compute the MST) is set to the relatedness between the two involved customers (which are connected by this arc). Kruskal's algorithm is stopped when two disjoint components (clusters) are found while building the MST. After that, one of the two clusters is randomly removed. Then, a customer i is randomly selected from the removed cluster. j, the most related customer to i, is selected from a different route. The route of j is then considered as the next route to be studied. The operator continues this way until deleting at least r customers. Note that the relatedness between customer i and j is defined in the same manner as in the *related removal*. The randomization control parameter of the *cluster removal* is denoted p_c.

4.4.2. Insertion operators

We have implemented four insertion operators namely *basic greedy 1*, *basic greedy 2*, *regret* and *H_insert*. Pisinger and Ropke [22] used only one kind of *basic greedy* and a set of *regret* operators which differ from each other in the level of regret (number of best positions examined for each customer). In [17], *basic greedy* and *regret* operators are implemented too, using the objective function of the PDPTWPR as a selection criterion. In the present paper, we propose two different kinds of *basic greedy* together with a *regret* operator. These operators consider, at each iteration, several routes (the number of routes equals the number of vehicles) when inserting a customer. To handle the constraints of the PTPSPD, the two *basic greedy* and the *regret* operators include selection criteria based on profits, traveling costs and loads. To avoid excessive computational efforts, only *regret-2*, which considers in its criteria the second best insertion position, is used. On the other hand, *H_insert* was proposed in [11] within a hybrid metaheuristic to solve the *Vehicle Routing Problem with Time Window* (VRPTW). In the present work, *H_insert* is adapted to the PTPSPD. To the best of our knowledge, it is the first time that *H_insert* is used in an ALNS algorithm. In our insertion operators, the constraint of inserting all the customers is relaxed. In addition, if there are less routes in the current solution than the available number of vehicles, the insertion of customers into empty routes is also examined. A description of *basic greedy 1*, *basic greedy 2*, *regret* and *H_insert* is given below.

Basic greedy 1 operator: This operator inserts each customer in its best position, on its best route. As said above, the *basic greedy 1* operator includes in its selection criteria profits, traveling costs and loads. More specifically, the insertion criteria of the *basic greedy 1* are defined in a quite similar manner as presented in Section 4.1. The difference is that the first criterion $cr_1^{\text{global}}(i,u,j)$ is defined according to expressions (4.9)–(4.13), where ρ can belong to any route from the available set of routes, $\alpha_3 \geq 0$ is a user-defined parameter, $Max\_LB(i)$ and $Max\_LA(i)$ are the maximum load collected before (backward) and after (forward) customer i respectively, the remaining terms/parameters are the same as those defined in Section 4.1.

$$cr_1^{\text{global}}(i(u),u,j(u)) = \max\left\{cr_1^{\text{global}}(i_{\rho-1},u,i_\rho,\rho=1,\ldots,h\right\}; \tag{4.9}$$

$$cr_1^{\text{global}}(i,u,j) = \alpha_1 \cdot cr_{11}(i,u,j) - \alpha_2 \cdot cr_{12}(i,u,j) - \alpha_3 \cdot cr_{13}(i,u,j); \tag{4.10}$$

$$cr_{11}(i,u,j) = pr_u; \tag{4.11}$$

$$cr_{12}(i,u,j) = c_{iu} + c_{uj} - \mu \cdot c_{ij}; \tag{4.12}$$

$$cr_{13}(i,u,j) = \max\left\{Max\_LB(i) + d_u;\ Max\_LA(i) + p_u\right\}. \tag{4.13}$$

The second criterion among all routes $cr_2^{global}(i,u,j)$ is then defined according to expressions (4.14) and (4.15), where λ and c_{0u} are defined in same way as in Section 4.1.

$$cr_2^{\text{global}}(i(u^*),u^*,j(u^*)) = \max\left\{cr_2^{\text{global}}(i(u),u,j(u)), u \text{ unrouted}\right\}; \tag{4.14}$$

$$cr_2^{\text{global}}(i,u,j) = \lambda \cdot c_{0u} + cr_1^{\text{global}}(i(u),u,j(u)). \tag{4.15}$$

Remark that, when the value of α_3 is set to 1 and the other parameters are set to 0, we select the customer which produces the minimum maximum load in the vehicle.

After some experimental tests, we remark that the configuration $\alpha_1 = \alpha_2 = \alpha_3 = 1$ leads to better results in comparison with configurations where $\alpha_1 + \alpha_2 + \alpha_3 = 1$. Hence, the parameter values of *basic greedy 1* are set as follows $\alpha_1 = \alpha_2 = \alpha_3 = 1$, μ and λ are randomly selected from the intervals $[0,3]$ and $[0,1]$ respectively.

Basic greedy 2 operator: This is almost the same operator as *basic greedy 1*. The difference between the two operators lies in the tuning of parameters. Indeed, in *basic greedy 2*, $\alpha_1 = \alpha_2 = \alpha_3 = 1$, $\mu = 1$ and λ is randomly

selected from the interval $[0, 1]$. One can remark that the *basic greedy 1* generates more diversification than *basic greedy 2*.

Regret operator: The regret operator aims at selecting the customer which will be regretted the most if not inserted at the current iteration. More specifically, let $cr_1^{\text{global}}(i, u, j)$ be the first insertion criterion of a customer u between i and j among all routes, as defined in expressions (4.9)–(4.13). The second criterion among all routes $cr_2^{\text{global}}(i, u, j)$ is then defined according to expressions (4.14) and (4.15). Let ζ be the best route of the best insertion, obtained by evaluating expressions (4.14) and (4.15) among all routes. Let us define $cr_1^{\text{global}-\zeta}(i, u, j)$ as the first insertion criterion of customer u between i and j among all routes except ζ. Let $cr_2^{\text{global}-\zeta}(i, u, j)$ be the corresponding second criterion. The best customer to insert according to $cr_2^{\text{global}-\zeta}(i, u, j)$ is obtained among all routes except ζ according to expressions (4.16) and (4.17).

$$cr_2^{\text{global}-\zeta}(i(u^*), u^*, j(u^*)) = \max\left\{cr_2^{\text{global}-\zeta}(i, u, j), u \text{ unrouted}\right\}; \tag{4.16}$$

$$cr_2^{\text{global}-\zeta}(i, u, j) = \lambda \cdot c_{0u} + cr_1^{\text{global}-\zeta}(i(u), u, j(u)). \tag{4.17}$$

The regret operator inserts the customer that will be regretted the most i.e. the one with the biggest value of $cr_2^{\text{global}}(i(u^*), u^*, j(u^*)) - cr_2^{\text{global}-\zeta}(i(u^*), u^*, j(u^*))$.

The *regret* operator uses the same parameter values as the *basic greedy 2*.

H_insert: This operator examines all the unrouted customers. The customers are scanned according to their removal order. Each time a customer is considered, *H_insert* inserts it in the best position over all the routes if the insertion improves the current solution. In *H_insert*, a best position is a position that maximizes the insertion cost, which is defined by the objective function value. Note that each customer is examined only once.

4.5. Diversification strategies

Several diversification mechanisms are used in our approach. They are detailed in the following.

4.5.1. Diversification via temperature update

Each time the temperature threshold T_0 is reached, the *Simulated Annealing* is recalled with a bigger "initial" temperature $T = itr \cdot 10$, where itr is the current number of iterations. Hence, sALNS can be viewed as a kind of multi-start heuristic where the previous current solution stands for the new initial solution. When executing sALNS with a big starting temperature, the heuristic accepts more bad quality solutions due to the metropolis rule. Some of these bad solutions help the algorithm to escape local optima and lead to more profitable solutions. Note that, when the temperature is reset, the selection probabilities of the removal/insertion operators are not changed.

4.5.2. Diversification via removal operators

In cases where a solution is accepted after a removal but not accepted after an insertion, a new iteration begins and a removal operator is executed once again. The new solution is then accepted if it is better than the previous one and the insertion operator is called. This results in a combination of two removals and one insertion, which is equivalent to the *meta-destroy* diversification mechanism proposed in [17]. Moreover, in sALNS, the solutions of several removals can be accepted before moving to a solution generated by an insertion operator. This constitutes a significant advantage for our approach, as we are no longer required to test the deletions of large numbers of customers, which may involve more computational efforts. Hence, the approach remains fast as it starts by deleting small numbers of customers and if it seems more suitable to delete more customers, this will be done by accepting solutions from several successive removals. This strategy leads to a more diversified search.

4.5.3. Diversification via noise terms

As in the classical ALNS [22], a noise term is included in the selection criteria of insertion operators to add more diversification. In the present paper, the noise term is also added to the selection criteria of insertion operators and it is computed similarly. Indeed, each time an insertion cost of a customer u is calculated according to the first criterion $cr_1^{\text{global}}(i,u,j)$ (see expressions (4.9)–(4.13)), a noise term δ is added to generate a modified version of the first criterion $cr_1'\text{global(i, u, j)} = \text{cr}_1^{\text{global}}\text{(i, u, j)} + \delta$. δ is randomly chosen from the interval $[-N_{\max}, N_{\max}]$, $N_{\max} = \eta \cdot \max\{c_{ij}, (i,j) \in E\}$, where c_{ij} is the traveling cost between customers i and j, and η is a user-defined parameter that controls the amount of noise.

When selecting an insertion operator, the algorithm decides whether to use the noise term or not depending on the performance of the insertion operators with and without the noise term. The decision is made according to the *roulette wheel selection principle*. More precisely, we consider ω^+ and ω^- as the weight of using the noise term and the weight of not using it respectively. A weight reflects the performance of a version (with or without noise) during the previous iterations. In a current iteration, the decisions of using the noise term or not are made according to the probabilities $\frac{\omega^+}{\omega^+ + \omega^-}$ and $\frac{\omega^-}{\omega^+ + \omega^-}$ respectively. Both ω^+ and ω^- are set to 1 at the beginning of the search. The values of ω^+ and ω^- are fixed until the couple of evaluation-application phases (denoted e-a) is performed. At the end of each e-a, the values of ω^+ and ω^- are adjusted according to the score of using the noise (denoted sc_{ω^+}) and the score of not using the noise (denoted sc_{ω^-}) obtained in that e-a respectively. At the beginning of each e-a, the scores are set to 0. The scores are then incremented at each iteration by adding the value σ_1, σ_2 or σ_3. σ_1 is added to sc_{ω^+} (or to sc_{ω^-}) if a better solution than the global best solution is obtained with (or without) the noise term. σ_2 is added to sc_{ω^+} (or to sc_{ω^-}) if the objective value of the solution that has been obtained with (or without) the noise term is better than the objective value of the current solution. Finally, σ_3 is added to sc_{ω^+} (or to sc_{ω^-}) if a solution obtained with (or without) the noise term, is accepted despite the fact that the objective value of this solution is worse than the objective value of the current solution.

As said before, the values of ω^+ and ω^- are updated at the end of each e-a. sc_{ω^+} and sc_{ω^-} are the scores of using the noise and not using it at the end of the current e-a respectively. Let $times_{\omega^+}$ be the number of times an operator uses the noise, and let $times_{\omega^-}$ be the number of times an operator does not use the noise in the e-a that just ends. Let $\rho \in [0,1]$ be a reaction factor that controls how quickly the algorithm reacts to the changes in scores. ω^+ and ω^- are then adjusted using expressions (4.18) and (4.19). Note that, if the value of ρ is set to 1, the selection only depends on the scores obtained during the e-a that just ends. If the value of ρ is set to 0, the selection only depends on the weight obtained in the previous e-a couples without considering the scores obtained during the e-a that just ends.

$$\omega^+ = (1-\rho)\cdot\omega^+ + \rho \cdot \frac{sc_{\omega^+}}{times_{\omega^+}}, \tag{4.18}$$

$$\omega^- = (1-\rho)\cdot\omega^- + \rho \cdot \frac{sc_{\omega^-}}{times_{\omega^-}}. \tag{4.19}$$

5. Computational results

In the present Section, we contrast the results of sALNS with those of CPLEX 12.2, the classical ALNS [22] and another ALNS version denoted *Li*. *Li* is a variant of sALNS that uses the removal and insertion operators of the ALNS proposed in [17]. We could not implement the ALNS proposed in [17] for our problem because the components of this heuristic handle some characteristics of the PDPTWPR that do not exist in the PTPSPD. For example, in a PDPTWPR, each vehicle has a set of selective customers that may be visited by other vehicles, and a set of reserved customers that have to be visited only by the vehicle. In addition, the ALNS proposed in [17] uses a mechanism that dynamically adjusts the behavior of the operators depending on their status (selective or reserved). This mechanism cannot be used in the PTPSPD. We also could not apply sALNS to the PDPTWPR because this heuristic may remove some reserved customers, which results in an infeasible

solution. Nevertheless, we remark that the removal/insertion operators of the ALNS proposed in [17] can be applied to the PTPSPD, if we enforce the pickup and the delivery operations of each customer to be performed simultaneously. This would result in a set of removal/insertion operators that differs from our set of operators. We want to test the removal/insertion operators of the ALNS proposed in [17] on our problem. Therefore, we create the *Li* version of sALNS, which uses these operators.

In the following, we begin by describing our instances. Then, we give details about the configuration of the construction heuristic and the classical ALNS [22]. A parameter tuning, a component evaluation and a comparison between sALNS, ALNS [22] and *Li* are considered. After that, we compare sALNS results with those of CPLEX 12.2 and the classical ALNS. Finally, as there are some similarities between our model and the one presented in [6], we evaluate the performance of sALNS on instances proposed in [6].

All heuristics are implemented in C and run on a laptop with an intel(R) Core (TM) i7-4600U CPU @ 2.10 GHz 2.70 GHz with 8.00 GB RAM and 64-bit operating system. CPLEX is executed on the same laptop. CPLEX default parameters are used and the solver is stopped when the computing time reaches two hours.

Due to the random aspect of the implemented heuristics, we run all of them 10 times for each instance. Each heuristic is run during 9 00 000 iterations.

As using 9 00 000 iterations when tuning the parameters of the implemented heuristics would take too much time, we use only 2000 iterations with 10 runs for the parameter tuning. For the component evaluation of sALNS, we want to see how good can be each configuration depending on the used components. Hence, we fix the iteration number to 200000 and the number of runs to 5.

5.1. Generation of instances for the PTPSPD

To generate instances for the PTPSPD, we modify the CPTP instances proposed in [1], which are obtained by modifying the *Capacitated Vehicle Routing Problem* instances presented in [5]. The number of customers in the instances proposed in [5] belongs to the set $\{50, 75, 100, 120, 150, 199\}$. The CPTP instances proposed in [1] are organized into three sets. The first set contains the 10 instances proposed in [5] with the same capacities and number of vehicles. The profit pr_i of customer i is generated for the instances proposed in [5] according to the expression $pr_i = (0.5 + h) \cdot d_i$, where d_i is the demand of customer i and h is randomly generated from the interval $[0, 1]$. In the second set, for each instance presented in [5], the cases $Q = 50$, $Q = 75$ and $Q = 100$ are considered, where Q refers to the capacity bound. For each case, three instances are generated with a different number of vehicles. The latter varies in the set $\{2, 3, 4\}$. The profits of the second set are generated in the same way as those of the first set. Hence, a total of 90 instances are considered in the second set. Finally, in the third set the capacity bounds used in [5] are unchanged. However, the number of vehicles varies in the set $\{2, 3, 4\}$. The profits of the third set are generated in the same way as those of the first set. Hence, the third set contains 30 instances. Therefore, a total of 130 CPTP instances are used in [1].

To adapt those instances to our problem, we generate pickup and delivery demands using the method given in [26]. This method was initially proposed for the VRPSPD. The mean pickup and delivery demands of the PTPSPD instances that are based on the first set presented in [1] (with the original vehicle numbers and capacity bounds) are given in Table 4. Table 4 also reports the number of customers n, the capacity bound Q, the number of vehicles m, the minimum delivery value $\min d_i$, the maximum delivery value $\max d_i$, the minimum pickup value $\min p_i$ and the maximum pickup value $\max p_i$ for each instance. The number of customers and the information about pickup and delivery demands of the remaining instances are the same as those of the first set presented in Table 4.

In [1] two problems were studied, namely the CPTP and the *Capacitated Team Orienteering Problem* (CTOP). In the CTOP, there are some constraints fixing the time limit for each vehicle route. The instances of the CPTP and the CTOP are quite similar. The difference between CPTP and CTOP instances lies in the introduction of a time limit for the CTOP instance. Instances of types "3" and "8" proposed for the CTOP in [1] differ from each other in the value of the time limit. For the CPTP, as time limit constraints do not exist, the instances of types "3" and "8" are exactly the same. Therefore, we only take type "8" into account. We obtain a total of 117 instances. Note that, we consider instances with unrounded demands.

TABLE 4. Description of PTPSPD instances derived from the first set presented in [1].

Instance	n	Q	v	Mean delivery	$\min d_i$	$\max d_i$	Mean pickup	$\min p_i$	$\max p_i$
6	50	160	10	9.210	0.86	30.00	6.330	0.00	22.84
7	75	140	20	10.892	0.25	31.31	7.295	0.00	22.60
8	100	200	15	8.379	0.04	29.82	6.201	0.00	27.35
9	150	200	10	8.656	0.04	30.00	6.244	0.00	27.35
10	199	200	20	9.402	0.04	31.31	6.608	0.00	27.35
13	120	200	15	5.092	0.08	18.09	6.367	0.00	24.00
14	100	200	10	9.420	0.00	40.00	8.680	0.00	32.00
15	150	200	15	8.656	0.04	30.00	6.244	0.00	27.35
16	199	200	20	9.402	0.04	31.31	6.608	0.00	27.35

5.2. Configuration of the construction heuristic

The parameters of the construction heuristic are chosen randomly from their respective intervals. See Section 4.1 for more details about the used intervals. This heuristic is run 10 times. Note that, the parameter values may change at each run. The best solution is considered as a starting point for the main algorithm.

5.3. Configuration of the original ALNS

We have implemented an adapted version of the classical ALNS heuristic proposed in [22]. As the original version of the ALNS uses the minimization of routing costs without considering profits, we have replaced its removals by those presented in Section 4.4.1. This adapted version of ALNS also uses five insertions. The first insertion is the *basic greedy* with the parameter values 1 for μ, α_1 and α_2; and 0 for α_3 and λ. The other insertions are the *regret-2*, *regret-3*, *regret-4* and *regret-m*, where m is the number of vehicles. The *regret* operators consider the same parameter values as the *basic greedy* operator. Note that with these parameter values, we obtain the same insertions as those of the classical ALNS using the PTPSPD objective function. The operator selection, the score update, the solution evaluation and the temperature update are identical to those used in the original ALNS. As there are no major modifications in this adapted version of ALNS, it will be denoted *classical ALNS* or *ALNS* in the remainder of the present paper.

5.3.1. Parameter tuning for ALNS

The parameters of the classical ALNS are described in Table 5. p_r and p_w are the randomization control parameters as described in Section 4.4.1. T_s is the starting temperature of ALNS and c is the cooling rate which decreases the current temperature value at each iteration. q is a random number that stands for the number of removed customers at each iteration. Definitions of parameters σ_1, ρ and η can be found in Section 4.5.3, by considering performance evaluation between several operators instead of performance evaluation between two versions (with and without noise). Definitions of σ_2 and σ_3 are quite different. Indeed, σ_2 and σ_3 are added to the score of an operator if a solution that has not been accepted before is obtained with this operator, in such a way that the objective value of the obtained solution is: (i) better than the objective value of the current solution for σ_2, and (ii) worse than the objective value of the current solution, but the solution is still accepted for σ_3. To detect a new solution, the objective function values of the previous solutions are stored. In order to generate an efficient version of the classical ALNS heuristic, we have studied several configurations of its parameters as shown in Table 6. In this table, C_x stands for the configuration name, and f is the value of the initial solution. $v1$ is a random number between a and b such that, a equals $0.1 \cdot n$ if $0.1 \cdot n$ is smaller than 30, a equals 30 otherwise, and b equals $0.4 \cdot n$ if $0.4 \cdot n$ is smaller than 60, b equals 60 otherwise. $v2$ is randomly chosen from the set $\{2, \ldots, 7\}$. $v3$ is randomly chosen from the set $\{1, 2\}$. Parameter T_s varies in the set $\{V_1, V_2, V_3\}$, where $V_1 = 0.0072 \cdot f$, $V_2 = 0.0010 \cdot f$ and $V_3 = 0.0005 \cdot f$, with f is the objective value of the initial solution. Parameter c varies in the set $\{V1, V2, V3, V4\}$, where $V1 = 0.99975$, $V2 = 0.9997$, $V3 = 0.9998$

TABLE 5. Description of ALNS parameters.

Parameter	Description
p_r	The randomization control parameter of the related removal operator
p_w	The randomization control parameter of the worst removal operator
T_s	The starting temperature
c	The cooling rate
σ_1	The scores of removal/insertion operators are incremented with σ_1 if they resulted in a new global best solution
σ_2	The scores of removal/insertion operators are incremented with σ_2 if they resulted in a solution that has not been accepted before and the objective value of this solution is better than the objective value of the current solution
σ_3	The scores of removal/insertion operators are incremented with σ_3 if they resulted in a solution that has not been accepted before, the objective value of this solution is worse than the objective value of the current solution, but the solution is accepted
ρ	The reaction factor: used to control how quickly the algorithm reacts to changes in the scores (see [22])
q	The number of removed customers
η	The noise rate: that controls the amount of noise

TABLE 6. Parameter configuration for the classical ALNS.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
p_r	6	6	6	6	6	6	6	6	6	6
p_w	3	3	3	3	3	3	3	3	6	10
T_s	V_1	V_1	V_1	V_2	V_3	V_3	V_3	V_3	V_3	V_3
c	$V1$	$V1$	$V1$	$V1$	$V1$	$V2$	$V3$	$V4$	$V3$	$V3$
σ_1	33	33	33	33	33	33	33	33	33	33
σ_2	9	9	9	9	9	9	9	9	9	9
σ_3	13	13	13	13	13	13	13	13	13	13
ρ	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
q	$v1$	$v2$	$v3$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$
η	v_1	v_1	v_1	v_1	v_1	v_1	v_1	v_1	v_1	v_1

	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20
p_r	3	10	6	6	6	6	6	6	6	6
p_w	6	6	6	6	6	6	6	6	6	6
T_s	V_3	V_3	V_3	V_3	V_3	V_3	V_3	V_3	V_3	V_3
c	$V3$	$V3$	$V3$	$V3$	$V3$	$V3$	$V3$	$V3$	$V3$	$V3$
σ_1	33	33	33	33	50	20	33	33	33	33
σ_2	9	9	9	0	9	9	0	9	9	9
σ_3	13	13	0	13	13	13	0	13	13	13
ρ	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.1	0.1
q	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$
η	v_1	v_1	v_1	v_1	v_1	v_1	v_1	v_1	v_2	v_3

TABLE 7. Comparison between parameter configurations for the classical ALNS.

		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Avg gap	Best	−8.30	−14.51	−8.85	−13.85	−14.36	−13.99	−14.16	−14.55	−14.74	−14.19
	Avg	1.88	−4.78	4.31	−4.86	−5.16	−4.83	−5.34	−4.97	−5.48	−4.85
	Worst	12.08	6.71	21.07	5.61	6.04	6.44	5.67	6.34	6.25	7.64
Avg CPU (s)	Best	8.46	1.56	0.78	1.57	1.63	1.58	1.61	1.57	1.56	1.56
	Avg	9.44	1.75	0.89	1.76	1.88	1.79	1.82	1.76	1.76	1.74
	Worst	10.84	2.01	1.04	2.05	2.23	2.06	2.15	2.05	2.05	1.98
		C11	C12	C13	C14	C15	C16	C17	C18	C19	C20
Avg gap	Best	−14.53	−14.56	−14.27	−14.49	−14.10	−14.35	−14.27	−14.04	−14.63	−14.48
	Avg	−4.85	−5.30	−5.24	−5.26	−5.05	−5.14	−4.81	−4.93	−4.89	−5.03
	Worst	6.45	6.68	6.47	6.60	7.56	7.69	7.05	6.27	7.93	6.04
Avg CPU (s)	Best	1.56	1.57	1.62	1.60	1.60	1.60	1.60	1.60	1.59	1.68
	Avg	1.74	1.75	1.85	1.81	1.80	1.78	1.80	1.80	1.79	1.91
	Worst	1.98	1.98	2.18	2.11	2.07	2.04	2.07	2.11	2.06	2.24

and $V4 = 0.99985$. Parameter η varies in the set $\{v_1, v_2, v_3\}$, where $v_1 = 0.025$, $v_2 = 0.03$ and $v_3 = 0.02$. $C1$ is the configuration given in [22]. To generate the remaining configurations, we repeatedly modify one parameter (or a set of parameters that act on a same component) from the best configuration uncounted so far.

The results of the studied ALNS configurations are presented in Table 7. ALNS is run 10 times with each configuration on all the studied instances and during 2000 iterations. The results are compared with those of CPLEX 12.2 (only 8 of CPLEX solutions are proven to be optimal).

More precisely, the average percentage deviation (among all the instances and the 10 runs) with respect to the last solutions obtained by CPLEX are presented. The percentage deviation (gap) of a solution sol_1 from a solution sol_2 is computed following the expression $gap = 100 \cdot (z_{sol_2} - z_{sol_1}) / |z_{sol_2}|$, where z_{sol_1} and z_{sol_2} are the objective values of sol_1 and sol_2 respectively. In the remainder of the present paper, the gap of a given solution with respect to another solution will be computed similarly.

In the first three rows of Table 7, the best, the average and the worst gaps are reported. The best, the average and the worst computing time (in seconds) are presented in the last three rows. Note that, the average computing time of CPLEX is 1.91 hours. In some cases, CPLEX terminates before 2 hours of computation with an "out of memory" status. To calculate the average computing time of CPLEX among all instances, the computing time in cases with an "out of memory" status is considered to be equal to 2 hours.

We can see that $C9$ outperforms the other configurations in terms of solution quality in the best and the average cases. $C9$ reaches the 4th position for the worst case. In addition, $C9$ obtains the second best computing time after $C3$, and the third average computing time after $C3$ and $C10$. For these reasons, $C9$ will be retained for the subsequent comparison and evaluation of ALNS with other methods. We can also see that $C1$ and $C3$ are the worst configurations in terms of solution quality. Furthermore, $C1$ is the worst configuration in terms of computing time. This shows the importance of choosing an adequate number of customers to remove for the studied problem.

5.4. Configuration of sALNS

In the present subsection, we run, using the parameter values of configuration $P1$ (see Tab. 9), several versions of sALNS in order to show the importance of the new components. After that, we select the best versions and combine them in a single algorithm. We finally tune the parameters of the final algorithm. Table 8 is provided to facilitate the understanding of the next sections. This table describes the parameters of the proposed approach.

TABLE 8. Description of sALNS parameters.

Parameter	Description
p_r	The randomization control parameter of the related removal
p_w	The randomization control parameter of the worst removal
p_{hn}	The randomization control parameter of the historical node-pair removal
p_{hr}	The randomization control parameter of the historical request-pair removal
p_c	The randomization control parameter of the cluster removal
c	The cooling rate
T_0	The temperature threshold
ml	The number of iterations performed before the scores are reset
σ_1	The scores of insertion operators using the noise (not using the noise) are incremented with σ_1 if the use (non-use) of the noise resulted in a new global best solution
σ_2	The scores of insertion operators using the noise (not using the noise) are incremented with σ_2 if the use (non-use) of the noise resulted in a solution with an objective value better than the objective value of the current solution
σ_3	The scores of the insertion operators using the noise (not using the noise) are incremented with σ_3 if the use (non-use) of the noise resulted in a solution with an objective value worse than the objective value of the current solution, but the solution is accepted
ρ	The reaction factor used to control how quickly the algorithm reacts to changes in the scores (as in the classical ALNS but in sALNS, it only deals with the noise terms)
η	The noise rate: that controls the amount of noise
l	The constant that controls the duration of an evaluation phase
r	The number of removed customers

5.4.1. Evaluation of the new components

In order to evaluate the added components and to create an efficient variant of sALNS, we run eight versions of the algorithm. The first version represents the case where all the components are used. *alg* refers to this version. In the other versions, we remove, each time, a component from *alg*. We replace our operator selection and score update by those of the classical ALNS. This version is called *alg-OpSel*. We remove the solution evaluation after the removals from *alg*. This version is called *alg-Eval*. Each insertion operator is removed from *alg* and we obtain *alg-BG1*, *alg-BG2*, *alg-Reg* and *alg-H* that refer to *alg* without the *basic greedy 1*, *alg* without the *basic greedy 2*, *alg* without the *regret* operator and *alg* without *H_insert* respectively. We also evaluate the algorithm with and without the *related removal*. The version without the *related removal* is called *alg-Rel*. In addition, we consider the deletion of our diversification *via* temperature update from *alg*. This version is denoted *alg-Temp*. We also study a version of *alg* that only uses the removal/insertion operators presented in [17]. For the *Shaw removal*, we have just removed the time window terms. This version is denoted *Li* and it is compared to the other versions. Furthermore, we add a version called *mdfBG2* that represents *alg* without the *basic greedy 2*, the *cluster removal* and the *historical request-pair removal*. We will explain later the reasons that lead us to generate the *mdfBG2* version. Finally, we run the classical ALNS described in Section 5.3. All the studied versions are run 5 times during 200000 iterations.

On the left side of Figure 1, we report the average gap of each version from CPLEX among all instances. On the right side of Figure 1, we give the average computing time in seconds for each version. More specifically, we show on both sides of this figure, the best, the average and the worst results. We start by comparing the different versions of our algorithm without considering *Li* and the classical ALNS. The latter versions will be studied at the end of the current subsection.

From Figure 1, we can see that *alg* slightly outperforms *alg-Rel*, *alg-Reg* and *alg-BG1* in terms of solution quality in the best, the average and the worst cases. Indeed, the best, the average and the worst gaps of *alg* are -25.7%, -24.25% and -22.71% respectively, while the best, the average and the worst gaps take the values

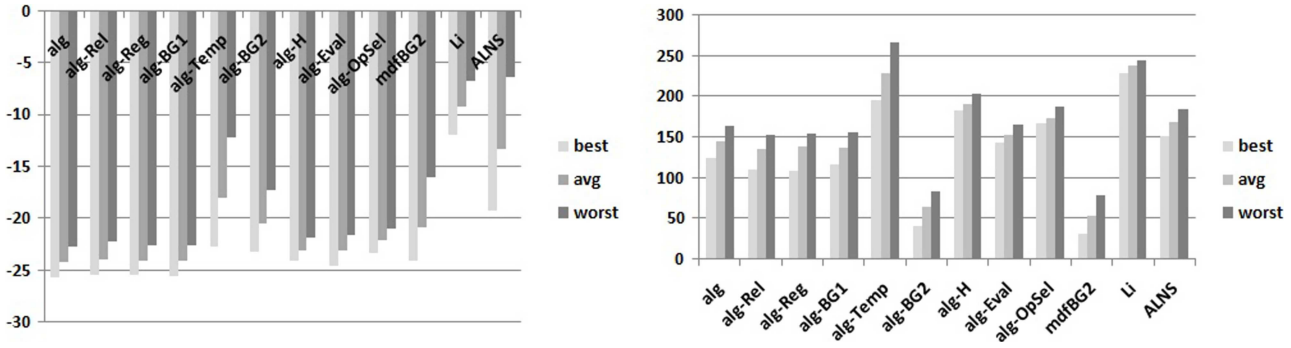

FIGURE 1. Evaluation of sALNS components in terms of percentage deviation (*left*) and computing time (*right*).

−25.39%, −23.99% and −22.27% respectively for *alg-Rel*, −25.38%, −24.07%, −22.6% respectively for *alg-Reg*, and −25.54%, −24.06% and −22.59% respectively for *alg-BG1*. We also remark that *alg* outperforms the remaining versions in terms of solution quality in the best, the average and the worst cases. In addition, from Figure 1, we remark that *alg* provides reasonable computing time. Indeed, disregarding *alg-BG2* and *mdfBG2* which are the fastest versions, the computing time of *alg* is not too far from the computing time of *alg-Rel*, *alg-Reg* and *alg-BG1*. We remark that *alg* is faster than the remaining versions. The fact that the gaps of *alg-Rel*, *alg-Reg* and *alg-BG1* are worse than the gaps of *alg* encourages us to maintain the *related removal*, the *regret* and the *basic greedy 1* operators in the main algorithm. We hope that the algorithm escapes local optima through the random aspect of these operators.

When evaluating the temperature update, we see that both the solution quality and the computing time of *alg-Temp* are degraded in comparison with *alg*. This proves that the restarts of the *alg* version from higher starting temperatures incorporate a balanced amount of intensification and diversification to the approach. We think that when our temperature update is used, the computing time decreases because good quality solutions are reached quickly. Indeed, after a good quality solution is reached, the probability of having a better quality solution either after a removal or an insertion decreases. This results in less updates of the current and the best solutions and their metrics (metrics are vectors used to quickly check a solution feasibility). If a solution S is not improved neither after a removal nor after an insertion, there would be a single update of the obtained solution S' ($S' = S$) and its metrics at the end of the current iteration. However, if both S and the global best solution S^{best} are improved after both removal and insertion, there would be an update of S, S^{best} and their metrics after both removal and insertion, which would result in an increase of the computing time.

Regarding *alg-BG2*, we see that deletion of the *basic greedy 2* from *alg* generates bigger gaps in comparison with *alg*. This highlights the importance of this operator within the *alg* version. On the other hand, the *alg-BG2* version together with *mdfBG2* constitute the fastest versions. This proves that, when *basic greedy 2* is used, the algorithm favors it over other insertion operators due to its efficiency. More specifically, the parallel aspect (several routes are evaluated at each iteration) of the *basic greedy 2* involves bigger values of computing time when the algorithm favors it, while a more balanced selection of insertion operators generates a faster approach, especially when using a rapid operator like *H_insert*.

Concerning *alg-H*, it is obvious that the deletion of *H_insert* worsens the solution quality. This is highlighted by the difference between the gaps of *alg-H* and *alg*. Also, *alg-H* requires more computational effort. This is because *H_insert* is a sequential insertion operator, and the deletion of *H_insert* involves the use of parallel insertions only (several routes are evaluated at each iteration), which requires more computing time.

By considering the *alg-Eval* version, we see that the solution evaluation after removal operators leads to the exploration of some missed areas of the search space. Indeed, this is stressed by the gaps obtained when deleting this evaluation from the main algorithm. Furthermore, the execution time of this version is larger than the execution time of *alg*.

It is obvious that our operator selection contributes significantly in the performance of the *alg* version. This can be observed in the difference between the gaps of *alg* and *alg-OpSel*. In addition, the deletion of this component involves more computational efforts.

Despite the fact that *alg-BG2* provides, on average, low quality solutions in comparison with *alg*, the deletion of the *basic greedy 2* from *alg* allows the *alg-BG2* version to reach better solutions on 18 instances. For six of these instances, even when *alg* is run with a bigger number of iterations, it stays trapped in local optima. We have remarked that the *mdfBG2* version is also able to reach good quality solutions for these instances. Additionally, *mdfBG2* provides better gaps and is faster than *alg-BG2*.

Regarding the *Li* version, it is clear that the replacement of our removal/insertion operators by those presented in [17] deteriorates both solution quality and computing time. Indeed, *Li* provides the worst results among all the studied versions. This proves that our operators are adequate for the PTPSPD.

When comparing ALNS with *Li*, we remark that ALNS gives better results in terms of solution quality and computing time. However, ALNS is worse than all the other versions in term of solution quality.

The study of all these versions encourages us to create an algorithm that contains both *alg* and *mdfBG2*, in such a way that, at the beginning of the search, this algorithm randomly chooses which version to execute. In what follows, sALNS will refer to this algorithm.

5.4.2. Parameter tuning for the final algorithm

In order to finely tune the parameters for our final algorithm, we start from sALNS using an arbitrary configuration denoted $P1$ then, we repeatedly modify one parameter at once from the best configuration uncounted so far. Each configuration is run 10 times on all the instances during 2000 iterations.

Table 9 provides the parameter values for each configuration. In this table, $v1$, $v2$ and $v3$ are the same parameter values as those described in Section 5.3.1. The parameter c varies in the set $\{V_1', V_2', V_3'\}$, where $V_1' = 0.99$, $V_2' = 0.98$ and $V_3' = 0.9998$. The parameter η varies in the set $\{v_1, v_2, v_3\}$, where $v_1 = 0.025$, $v_2 = 0.03$ and $v_3 = 0.02$. Px is the name of the xth configuration, where $x \in \{1, 2, \ldots, 21\}$.

In Table 10, we give the results of each configuration. The rows of this table are defined in the same manner as the rows of Table 7.

From Table 10, we remark that configuration $P20$ leads to the best results in term of percentage deviation. However, the computing time of $P20$ is about 4 times bigger than the computing time of configuration $P1$. $P1$ only differs from $P20$ in the value of parameter r. So, we decided to increase the iteration number of $P1$ and compare the obtained results with those of $P20$. We remark that the increase in $P1$ iteration number leads to better results in comparison with $P20$. Indeed, $P1$ reaches the values -21.86%, -15.57% and -6.57% for the best, the average and the worst gaps respectively. While the computing time of $P1$ reaches the values 3.34, 6.00 and 9.09 seconds for the best, the average and the worst computing time respectively. As we can see, the computing time of $P1$ is still smaller.

Disregarding $P20$, we note that no configuration dominates the others in terms of best, average and worst gaps simultaneously. Hence, we choose the configuration that outperforms the others in term of average gap. The $P1$ configuration has the best average gap. Thus, it is retained for the final algorithm.

We recall that in Section 4.4.2, parameters α_1, α_2, α_3 and λ are used in the selection criteria of the insertion operators. The parameter values of α_1, α_2, α_3 are all set to 1, and λ is randomly selected from the interval $[0, 1]$. The reason is that we started from the parameter values $\alpha_1 = \frac{1}{2}$, $\alpha_2 = \frac{1}{2}$, $\alpha_3 = 0$ and $\lambda = 0$, that lead to the selection of customers whose insertions result in better objective function values in comparison with other insertions, and we tried to improve these values.

Table 11 reports the results of the studied parameter configurations that we tried, in terms of best, average and worst gaps with respect to CPLEX solutions, and in terms of best, average and worst computing time in seconds.

Let Con_1 be the configuration with parameter values $\alpha_1 = \frac{1}{2}$, $\alpha_2 = \frac{1}{2}$, $\alpha_3 = 0$ and $\lambda = 0$. We supposed that the diversification induced by other values for parameter λ could provide better solutions. Hence, we tested a configuration Con_2 that differs from Con_1 in the value of parameter λ. In Con_2, λ is randomly selected from the interval $[0, 1]$.

TABLE 9. Parameter setting for sALNS.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
p_r	0	0	6	6	6	50	0	0	0	0	0
p_w	6	6	3	6	6	50	6	0	0	6	6
p_{hn}	50	50	0	0	6	50	50	50	0	50	50
p_{hr}	6	6	0	0	6	50	50	50	0	6	6
p_c	6	6	0	0	6	50	50	50	0	6	6
c	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_2{}'$	$V_3{}'$
T_0	1	0.5	1	1	1	1	1	1	1	1	1
ml	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$
σ_1	33	33	33	33	33	33	33	33	33	33	33
σ_2	9	9	9	9	9	9	9	9	9	9	9
σ_3	0	0	0	0	0	0	0	0	0	0	0
ρ	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
l	5	5	5	5	5	5	5	5	5	5	5
η	v_1	v_1	v_1	v_1	v_1	v_1	v_1	v_1	v_1	v_1	v_1
r	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$

	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21
p_r	0	0	0	0	0	0	0	0	0	0
p_w	6	6	6	6	6	6	6	6	6	6
p_{hn}	50	50	50	50	50	50	50	50	50	50
p_{hr}	6	6	6	6	6	6	6	6	6	6
p_c	6	6	6	6	6	6	6	6	6	6
c	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$	$V_1{}'$
T_0	1	1	1	1	1	1	1	1	1	1
ml	$\frac{itr}{8}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$	$\frac{itr}{4.5}$
σ_1	33	33	33	33	33	33	33	33	33	33
σ_2	9	9	0	9	9	9	9	9	9	9
σ_3	0	13	0	0	0	0	0	0	0	0
ρ	0.2	0.2	0.2	0.1	0.2	0.2	0.2	0.2	0.2	0.2
l	5	5	5	5	10	1	5	5	5	5
η	v_1	v_1	v_1	v_1	v_1	v_1	v_2	v_3	v_1	v_1
r	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v2$	$v1$	$v3$

As the obtained results confirm our assumption (see Tab. 11), we continued the tests from Con_2, we tried configuration Con_3, which has the following parameter values $\alpha_1 = \alpha_2 = \alpha_3 = \frac{1}{3}$ and λ is randomly chosen from the interval $[0, 1]$. As the results of Con_3 are better than those of Con_2 (see Tab. 11), we tried another configuration (denoted Con_4), in which the sum of α_1, α_2 and α_3 is greater than 1. The idea behind Con_4 is that we want the amount of diversification induced by λ to be more balanced. Hence, Con_4 has the following parameter values $\alpha_1 = \alpha_2 = \alpha_3 = 1$ and λ is randomly chosen from the interval $[0, 1]$. From Table 11 we can see that Con_4 leads to the best results.

In Section 4.2, we say that the metropolis rule is only applied after insertion operators because this leads to better results. In order to prove that, some experimental tests are carried out. Indeed, a parameter $metro \in \{0, 1\}$ is added to sALNS, in such a way that, if $metro$ takes the value 0, sALNS is run without the metropolis rule after the removal operators. Otherwise, sALNS uses the metropolis rule after the removal operators.

The above described tests show that when $metro = 0$, sALNS obtains the values -18.92%, -10.61% and 0.92% for the best, the average and the worst gaps respectively. When $metro = 1$, sALNS obtains the values

TABLE 10. Comparison between parameter configurations for the final sALNS.

		P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
Avg gap	Best	−18.92	−18.97	−15.74	−16.66	−18.19	−17.63	−18.90	−18.84	−17.80	−18.22	−18.87
	Avg	−10.61	−10.30	−5.44	−4.11	−7.57	−8.31	−9.42	−9.98	−6.15	−9.49	−8.38
	Worst	0.92	2.27	6.06	15.32	6.64	5.11	1.06	3.03	9.14	2.95	6.81
Avg CPU (s)	Best	1.05	1.15	0.91	1.06	1.05	1.02	1.12	1.15	1.16	0.94	1.67
	Avg	1.65	1.78	1.50	1.69	1.67	1.66	1.74	1.85	1.73	1.69	2.26
	Worst	2.43	2.63	2.22	2.45	2.54	2.55	2.53	2.79	2.53	2.70	3.14

		P12	P13	P14	P15	P16	P17	P18	P19	P20	P21
Avg gap	Best	−18.66	−18.90	−18.74	−18.78	−18.78	−18.49	−18.46	−18.89	−17.68	−14.60
	Avg	−10.24	−10.51	−10.39	−10.07	−9.69	−9.66	−10.17	−9.45	−12.03	1.21
	Worst	1.11	1.87	1.35	3.25	2.12	2.04	2.55	2.90	−6.31	25.02
Avg CPU (s)	Best	1.29	1.05	1.02	1.03	1.06	1.05	1.06	1.01	4.81	0.65
	Avg	1.80	1.64	1.66	1.67	1.60	1.62	1.60	1.53	6.94	1.11
	Worst	2.50	2.46	2.43	2.51	2.24	2.37	2.30	2.10	9.78	1.75

TABLE 11. Comparison between parameter configurations for insertion operators.

		Con_1	Con_2	Con_3	Con_4
Avg gap	Best	−5.94	−14.95	−15.74	−18.92
	Avg	2.80	−5.38	−5.44	−10.61
	Worst	11.52	8.02	6.06	0.92
Avg CPU (s)	Best	0.74	0.93	0.91	1.05
	Avg	1.11	1.49	1.50	1.65
	Worst	1.60	2.16	2.22	2.43

−2.15%, 10.73% and 23.08% for the best, the average and the worst gaps respectively. Regarding the average computing time, when $metro = 0$, sALNS obtains the values 1.05, 1.65 and 2.43 seconds for the best, the average and the worst computing time respectively. When $metro = 1$, sALNS obtains the values 0.53, 0.83, 1.28 seconds for the best, the average and the worst computing time respectively.

These results confirm that, even if the computing time is slightly better when $metro = 1$, much better results are obtained when the metropolis rule is only applied after insertion operators.

5.5. Study of the performance of sALNS

We evaluate, on the new generated instances, the performance of sALNS, ALNS and CPLEX using our mathematical formulation of the PTPSPD. In addition, we run our construction heuristic using the criterion $cr_1^{\text{global}}(i, u, j)$ as defined in expressions (4.9)–(4.13), and with all possible combinations of the parameter values. The best encountered solutions are returned.

In Table 12 the results of all the approaches are compared. The detailed results are reported in Appendix B. In Table 12, *sALNS* refers to our algorithm and *ALNS* to the classical ALNS. CPLEX refers to CPLEX12.2 results. $H_{\text{const}}^{\text{1run}}$ and H_{const} stand for our construction heuristic executed one time using randomly generated parameter values and our construction heuristic using all possible combinations of parameter values (as described above) respectively. The instances are divided into 13 sets according to their vehicle number and capacity bound. In each set, there are nine instances with 50–199 customers. On column *Set*, the sets of types "*Q*-*m*" are presented, where Q stands for the capacity bound and m is the number of vehicles. When Q takes the value X, this means that the original capacity bound is used (see [5]). Similarly, when m takes the value Y this means that the

TABLE 12. Heuristic evaluation according to the number of vehicles and the capacity bound.

Set	CPLEX	H^{1run}_{const}		H_{cosnt}		ALNS		sALNS			
	$\%^{UB}$	$\%^{cpx}$	CPU	$\%^{cpx}$	CPU	$\%^{cpx}$	CPU	$\%^{cpx}$	$\%^{UB}$	$\%^{ALNS}$	CPU
X-Y	36.02	−110.9	0.06	−163.11	603.09	−197.56	182.23	−213.33	6.95	−9.39	149.41
50-2	10.42	137.52	0.01	36.91	90.89	5.57	589.74	−3.11	8.01	−9.52	167.93
50-3	14.11	142.53	0.01	44.19	127.73	4.60	927.26	−3.12	11.70	−8.46	367.1
50-4	18.36	149.65	0.01	50.62	158.07	5.00	1033.14	−3.93	15.46	−9.42	710.39
75-2	12.07	93.19	0.01	20.77	118.02	−1.2	630.23	−4.12	8.71	−2.87	274.18
75-3	15.20	102.67	0.01	29.41	179.84	−1.35	831.44	−5.64	10.66	−4.5	723.67
75-4	20.83	99.55	0.02	31.97	216.39	−3.79	1062.02	−10.53	13.18	−7.7	793.11
100-2	13.41	66.58	0.01	14.58	162.98	−5.19	590.49	−7.38	7.72	−2.06	444.9
100-3	15.36	67.76	0.02	19.58	223.92	−.8	886.37	−5.99	10.49	−5.33	702.83
100-4	20.63	82.26	0.02	24.29	268.25	−5.38	931.22	−11.43	11.92	−6.1	1344.18
X-2	11.37	49.43	0.02	9.24	328.87	−4.21	643.49	−5.49	6.78	−1.29	638.52
X-3	18.13	45.71	0.03	7.36	583.93	−11.02	695.28	−14.24	7.44	−2.92	654.13
X-4	28.73	26.04	0.06	−7.53	568.00	−18.22	669.69	−25.75	7.89	−8.26	601.66
Avg	18.05	73.23	0.02	9.1	279.23	−17.97	744.05	−24.16	9.76	−5.99	582.46

original vehicle number is used. For each approach, we give the average gap from CPLEX in column $\%^{cpx}$ and the average computing time CPU in seconds. $\%^{UB}$ stands for the average gap from the upper bound provided by CPLEX12.2 (only 8 of CPLEX solutions are proven to be optimal). $\%^{alns}$ refers to the average gap between sALNS and ALNS. Finally, *avg* refers to the average results on all sets. From Table 12 and Appendix B, we remark that $H^{1\text{run}}_{\text{const}}$ provides better quality solutions than CPLEX in two cases for instances with big (original) capacity bounds and number of vehicles. For the remaining sets, $H^{1\text{run}}_{\text{const}}$ provides better quality solution than CPLEX solutions in only one case (last instance in Appendix B). However, the heuristic is very quick.

Regarding H_{const}, we note that it gives, on average, better results than the CPLEX for instances with big capacity bounds and a number of vehicles greater than or equal to 4. The computing time of this heuristic is smaller than the computing time of both ALNS and sALNS.

The classical ALNS heuristic is slower than other heuristics. This heuristic outperforms CPLEX in all sets except when the capacity bound equals 50.

sALNS provides the best results among all the studied approaches, especially for the sets with a big number of vehicles and a large capacity bound. More precisely, our heuristic gives a better gap from the upper bound than CPLEX. sALNS provides better results than CPLEX in 97 cases and identical results in 17 cases (including the optimal solutions). sALNS does not reach CPLEX solution for only three instances, namely 9-150-50-3, 9-150-50-4 and 15-150-50-4. The gaps of sALNS from CPLEX on these instances are equal to 0.44%, 3.64% and 0.77% respectively (see Appendix B for detailed results). The gap between sALNS and ALNS shows that the former heuristic is more efficient. Furthermore, sALNS is generally faster than ALNS except for the set with 4 vehicles and a capacity bound of 100.

In Figure 2, we report the average results of the heuristics according to the number of customers for all the studied instances. Actually, the instances are divided into five sets according to their number of customers (50, 75, 100, 120, 150 and 199 customers). On the left side of the figure, the average gap from CPLEX of our construction heuristic executed one time ($H^{1\text{run}}_{\text{const}}$), our construction heuristic using all possible combinations of parameter values (H_{const}), ALNS and sALNS is reported. On the right side of the figure, the average computing time in seconds of these heuristics is presented.

From Figure 2, we remark that the gap value of sALNS is always negative. In addition, sALNS always reaches the minimum values in comparison with the other heuristics. We also remark that the deviation of sALNS from CPLEX increases with the number of customers. This is because CPLEX fails to find good quality solutions within the available time limit for these instances.

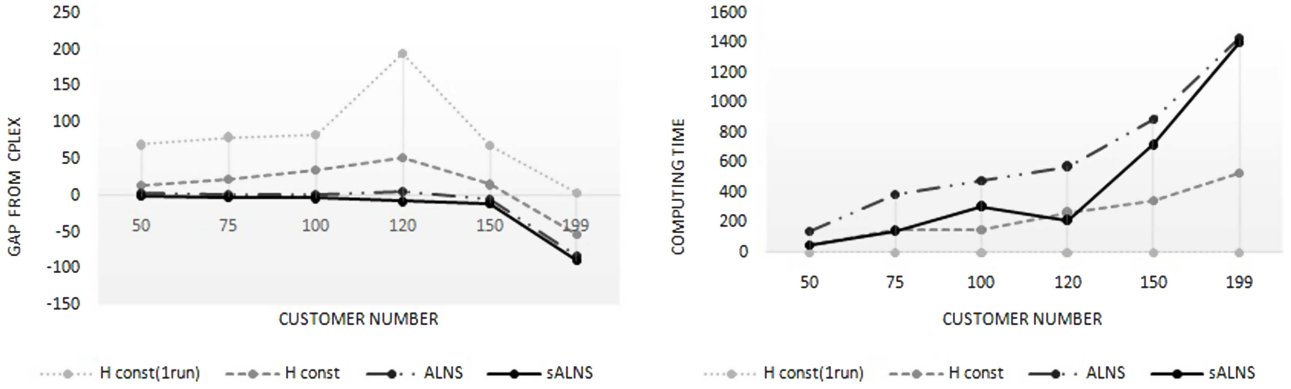

FIGURE 2. Heuristic evaluation according to the number of customers.

Regarding the classical ALNS, its gap with respect to CPLEX reaches negative values only when the customer number is greater than or equal to 150. We think that this is because the heuristic cannot visit some regions of the search space due to the following reasons. (1) ALNS does not evaluate solutions after the removals. Hence, the diversification mechanism *via* removal operators cannot be used. As a result, ALNS cannot accept solutions with less customers, which may be a characteristic of an optimal solution. (2) As the insertion operators of ALNS do not manage the load fluctuation within the solutions, ALNS cannot create more space in the solutions and insert additional customers. ALNS may miss some good quality solutions, especially if there are customers with big profits that do not significantly degrade the total traveling cost if inserted in the right positions. (3) The non-use of the diversification *via* temperature update, leads ALNS to stay trapped in local optima. (4) The operator selection of ALNS is not adequate for the PTPSPD.

Regarding the two versions of the construction heuristic, we remark that H_{const} reaches a negative value only when the number of customers is equal to 199. $H_{\text{const}}^{\text{1run}}$ does not reach CPLEX solutions on any set.

Concerning the computing time, we remark that $H_{\text{const}}^{\text{1run}}$ is the fastest heuristic. sALNS has quite similar results with H_{const} when the number of vehicles equals 50, 75 and 120. However, sALNS is slower than H_{const} for the other sets of instances. When compared with ALNS, we remark that sALNS is faster in all the studied sets.

In Figure 3, we report the average gap from CPLEX (left) and the average computing time in seconds (right) of the studied heuristics according to the number of vehicles. All the instances are studied. Six sets of instances are formed depending on the number of vehicles (with 2, 3, 4, 10, 15 and 20 vehicles). We remark that sALNS outperforms the other heuristics in term of average gap, for every set of instances. Regarding the computing time, we can see that sALNS is faster than ALNS. sALNS is slower than H_{const} for instances with small numbers of vehicles (2, 3 and 4). However, when the number of vehicles increases, sALNS becomes faster.

We remark that ALNS always provides negative gaps with respect to CPLEX. In addition, ALNS outperforms the two versions of the construction heuristic, for all sets of instances. Regarding the computing time, as sALNS, ALNS is slower than H_{const} for instances with small numbers of vehicles (2, 3 and 4). ALNS is faster for instances with 10 vehicles and more.

We think that the computing time of both ALNS and sALNS decreases for instances with a number of vehicle greater than or equal to 10 because such instances have original values of the number of vehicles and the capacity bound (see Sect. 5.1). Solutions of such instances generally contain all the customers. Hence, the problem becomes easier to solve as ALNS and sALNS only have to reorder the customers instead of selecting the best set of customers and then reordering that set. On the one hand, there would be no update of the current and the best solutions and their metrics after the removals. On the other hand, as good quality solutions are reached quickly, there would also be less updates after the insertions. This would result in less computing time.

We remark that the gaps of H_{const} from CPLEX reach negative values only when the number of vehicles is greater than or equal to 10. The gaps of $H_{\text{const}}^{\text{1run}}$ with respect to CPLEX reach negative values for instances with 15 and 20 vehicles.

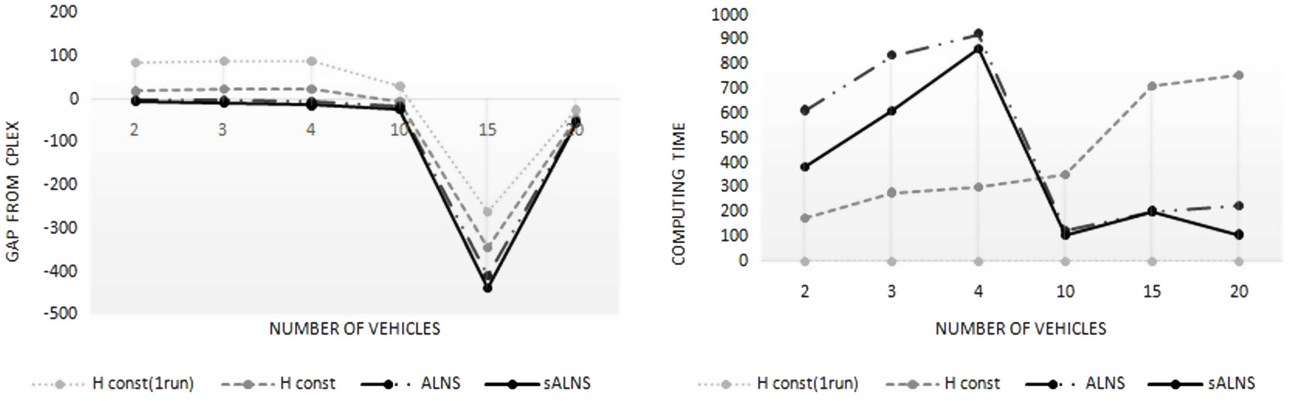

FIGURE 3. Heuristic evaluation according to the number of vehicles.

From Table 12, Figures 2 and 3, we can say that sALNS efficiency in terms of solution quality is not affected by the changes in the number of customers, the number of vehicles or the capacity bound. However, the computing time of sALNS depends on the values of the latter data.

5.6. Evaluation of the performance of sALNS on VRPSPD instances

As we said in Section 3, our model can be equivalent to the one proposed in [6] if all the profits are set to a very large number in comparison with the traveling costs (or distances). So, we want to evaluate the performance of sALNS on the instances with symmetric traveling cost matrix proposed in [6] by fixing the profits to a large number.

In the following, we describe the used benchmark instances. Then, we report the results of sALNS on these instances. sALNS results are compared to those reported in [6] using a *branch-and-price* algorithm with a time limit of one hour.

5.6.1. Description of VRPSPD benchmark instances

We use the instances of Class 1 presented in [6] for the VRPSPD, which were based on 3 VRPTW instances (r101, c101 and rc101) proposed in [27]. To adapt VRPTW instances to the VRPSPD, the authors in [6] considered the first 20 and the first 40 customers for each instance. The time windows were neglected. The demands of the original instances were considered as delivery demands. For each customer i, its pickup demand p_i was generated from its delivery demand d_i according to expressions $p_i = \lfloor (1-\gamma) \cdot d_i \rfloor$ if i is even, and $p_i = \lfloor (1+\gamma) \cdot d_i \rfloor$ if i is odd. The parameter γ takes either the value 0.2 or the value 0.8. This results in 12 different instances. The authors in [6] rounded up the Euclidean distance between customers.

We also use the instances of Class 3S and Class 3C. Class 3S instances are based on the *Capacitated Vehicle Routing Problem* (CVRP) instances available on VRPLIB [30], with 20 and 40 customers. For each CVRP instance, the fraction of delivery customer can take the values $\frac{1}{2}$, $\frac{2}{3}$ or $\frac{4}{5}$, and the capacity bound varies in the set $\{150, 200, 300\}$. A total of 18 instances are generated.

Class 3C is generated from Class 3S as follows: Each demand on Class 3S is considered as a delivery demand. For each customer i, its pickup demand p_i is computed according to the expression $p_i = (0.5 + r) \cdot d_i$, where d_i is the delivery demand of i and r is randomly chosen using a uniform distribution in the interval $[0, 1]$. Class 3C contains 18 different instances. In instances of Class 3S and Class 3C, the number of vehicles was computed as the minimum number of vehicles for which a feasible solution exists. See [6] for more details.

5.6.2. Behavior of sALNS on the benchmark instances

To test sALNS on VRPSDP instances, we consider the instances proposed in [6] as PTPSPD instances by adding a profit to each customer. For enforcing sALNS to favor solutions containing all the customers and hence,

TABLE 13. Behavior of sALNS on VRPSPD instances of Class 1.

ins	m	B&P	sALNS		
		Cost	Cost	Gap	CPU
c101_20_02	4	272	272	0.00	35.12
c101_20_08	4	279	279	0.00	34.42
r101_20_02	3	329	329	0.00	27.59
r101_20_08	3	342	342	0.00	30.42
rc101_20_02	5	428	428	0.00	41.55
rc101_20_08	5	458	458	0.00	40.54
c101_40_02	8	551	551	0.00	65.92
c101_40_08	8	569	589	3.40	68.14
r101_40_02	6	601	596	−0.84	64.23
r101_40_08	6	629	627	−0.32	70.55
rc101_40_02	9	886	886	0.00	72.02
rc101_40_08	9	926	926	0.00	79.26
Avg				0.19	52.48

TABLE 14. Behavior of sALNS on VRPSPD instances of Class 3S and Class 3C.

ins	m	B&P	sALNS			ins	m	B&P	sALNS		
		Cost	Cost	Gap	CPU			Cost	Cost	Gap	CPU
3S_20_50_1	6	8769	8769	0.00	32.82	3C_20_50_1	11	12720	12720	0.00	56.56
3S_20_50_2	4	7986	7986	0.00	28.07	3C_20_50_2	7	11559	10461	−10.50	47.85
3S_20_50_3	3	6445	6445	0.00	23.70	3C_20_50_3	6	8387	8387	0.00	36.00
3S_20_66_1	7	9129	9129	0.00	37.16	3C_20_66_1	12	14578	14578	0.00	80.68
3S_20_66_2	5	7470	7470	0.00	33.30	3C_20_66_2	8	11178	11176	−0.02	46.76
3S_20_66_3	3	8346	7036	−18.62	26.87	3C_20_66_3	5	8160	8160	0.00	33.03
3S_20_80_1	8	10707	10707	0.00	52.23	3C_20_80_1	11	12802	12802	0.00	59.31
3S_20_80_2	6	10093	9024	−11.85	37.33	3C_20_80_2	8	10087	10087	0.00	46.10
3S_20_80_3	4	7058	7058	0.00	27.63	3C_20_80_3	5	8317	8317	0.00	33.59
3S_40_50_1	10	18282	18282	0.00	89.41	3C_40_50_1	22	27559	27245	−1.15	134.04
3S_40_50_2	8	14603	14603	0.00	66.40	3C_40_50_2	16	21773	21773	0.00	129.70
3S_40_50_3	5	11610	11343	−2.35	55.04	3C_40_50_3	10	15629	15523	−0.68	80.71
3S_40_66_1	12	18370	18003	−2.04	78.88	3C_40_66_1	22	25981	25981	0.00	126.07
3S_40_66_2	9	15307	15307	0.00	67.95	3C_40_66_2	15	21319	21377	0.27	106.91
3S_40_66_3	6	11725	11725	0.00	61.72	3C_40_66_3	10	15293	15293	0.00	79.43
3S_40_80_1	17	20665	20665	0.00	120.32	3C_40_80_1	21	26273	26617	1.29	121.09
3S_40_80_2	12	17599	17503	−0.55	112.29	3C_40_80_2	16	20652	20652	0.00	108.24
3S_40_80_3	8	13317	13221	−0.73	81.83	3C_40_80_3	10	15365	15365	0.00	82.41
Avg				−2.01	57.39	Avg				−0.60	78.25

provides feasible solutions for the VRPSPD, we add big profit values to the studied instances. The profits are set to 10 000. When sALNS stops, we check the feasibility of the obtained solutions.

The obtained results are reported in Tables 13 and 14. Table 13 displays results for Class 1, while results of Class 3S and Class 3C are displayed in Table 14.

In these tables, *B&P* refers to the best solution reported using the *branch-and-price* algorithm presented in [6]. *sALNS* refers to our heuristic. *cost* stands for the best traveling cost (best distance) obtained, and *CPU* is the corresponding computing time in seconds. *ins* refers to the name of the instances, where an instance $tw\_n\_val_\gamma$ of Class 1 is an instance generated using the VRPTW instance tw with n customers and γ is equal to val_γ. In an instance $Class\_n\_frac\_Q$ of Class 3S or Class 3C, $Class$ refers to the class name (3S or 3C), n is

the number of customers, $frac \in \{50, 66, 80\}$ is the fraction of delivery customer, where 50 refers to $\frac{1}{2}$, 66 refers to $\frac{2}{3}$ and 80 refers to $\frac{4}{5}$, and Q stands for the capacity bound. m refers to the number of vehicles. *gap* refers to the gap of sALNS from the best solution reported in [6]. The gap of a sALNS solution from a *branch-and-price* (B&P) solution is computed following the expression $gap = 100 \cdot (z_{sALNS} - z_{B\&P})/|z_{sALNS}|$, where z_{sALNS} and $z_{B\&P}$ are the objective values given by sALNS and B&P respectively.

From Table 13, we remark that, for instances of Class 1, sALNS reaches almost all the solutions provided by the *branch-and-price* algorithm in reasonable computing time. sALNS does not reach the B&P solution for instance c101_40_08 and reports a gap of 3.40%. We remarked that for this instance, sALNS uses one more vehicle (9 instead of 8) than the B&P. On the other hand, sALNS provides a better solution than the *branch-and-price* algorithm for both instances r101_40_02 and r101_40_08. Note that, in instance r101_40_08, sALNS solution uses more vehicles than the B&P (7 instead of 6).

From Table 14, we can see that sALNS gives better or equal solutions than the B&P algorithm for instances of Class 3S. On the other hand, for instances in Class 3C, our approach fails to reach B&P solutions for both instances 3C_40_66_2 and 3C_40_80_1.

6. Conclusion

In the present paper, we describe the *Profitable Tour Problem with Simultaneous Pickup and Delivery services* (PTPSPD) and we present a mathematical formulation. We propose a new construction heuristic together with a *selective Adaptive Large Neighborhood Search* (sALNS) heuristic. sALNS makes use of several components. The most important ones are the solution evaluation, the operator selection, the tuned removal/insertion operators and the diversification *via* the temperature update.

sALNS is tested on 117 instances based on benchmark ones. Some parameters are added to the benchmark instances to make them compatible with the PTPSPD. To prove the effectiveness of our heuristic, we first evaluate its components. Then, we implement a version of sALNS (denoted *Li*) that uses the removal/insertion operators presented in [17]. *Li* solutions are compared with sALNS solutions. We also run both the classical *Adaptive Large Neighborhood Search heuristic* (ALNS) and our mathematical formulation using the CPLEX solver. The computational results show that, sALNS performs well in comparison with *Li*, ALNS and CPLEX in both solution quality and computing time. Furthermore, we evaluate the performance of sALNS on VRPSPD instances from the literature. The experimental tests show that our heuristic gives good quality solutions in reasonable computing time.

A future work may consider the application of sALNS to other variants of *Pickup and Delivery Problems with Profits*, including different realistic constraints as tour-duration, heterogeneous fleet and time windows. This could be done by incorporating these constraints into the selection criteria of the insertion operators for instance.

On the other hand, the PTPSPD could be studied with some other real-life constraints as time windows or, with the objective of profit maximization subject to tour-duration constraints. This would result in a *Team Orienteering Problem with Simultaneous Pickup and Delivery services*.

Appendix A. Sub-tour elimination proof

Constraints (3.5), (3.6), (3.8) and (3.9) state that the number of arcs entering a node $i \in N$ must be equal to the number of arcs leaving i. In addition, this number has to be equal to 0 or 1. Hence, if customer $i \in N$ is included in the solution, there will be exactly one arc entering and one arc leaving i. Otherwise, no arc can neither enter nor leave i. As a result, the proposed mathematical model prohibits solutions (and hence, sub-tours) including different numbers of arcs entering and leaving the solution's nodes. An example of such sub-tours is given in Figure A.1.

In that figure, two graphs with five customers (1, 2, 3, 4 and 5) and a depot (0) are given. On the left side of the figure there are different numbers of arcs entering and leaving node 5, which violate constraint (3.6). On the right side of the Figure, same numbers of arcs are entering and leaving node 5 but these numbers are greater than 1. According to constraints (3.5) and (3.8), if such solutions exist, this involves that there exists t_i which is greater than 1. Hence, constraints (3.9) will be violated.

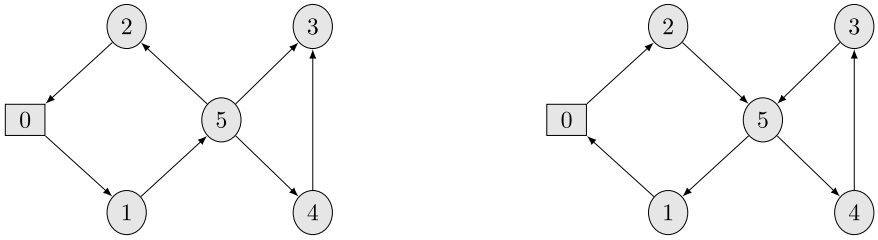

FIGURE A.1. Examples of prohibited solution configurations using constraints (3.5), (3.6), (3.8) and (3.9).

FIGURE A.2. Example of prohibited solution configurations that satisfy constraints (3.5), (3.6), (3.8) and (3.9) but still contain sub-tours.

Note that not all sub-tours are handled by constraints (3.5), (3.6), (3.8) and (3.9). An example of a sub-tour that is not handled by these constraints is given in Figure A.2. constraints (3.2) and (3.3) handle the remaining sub-tours by assuming that, for each customer $i \in N$, the pickup and the delivery demands of i cannot be both equal to 0. This sub-tour elimination is demonstrated as follows.

Let us consider constraints (3.2) and (3.3) from our mathematical model. We suppose that, for each customer $i \in N$ in the PTPSPD, the pickup demand p_i and the delivery demand d_i of i cannot be both equal to zero.

We consider a solution containing a sub-tour $a_1 - a_2 - a_3 - \ldots - a_{k-1} - a_k - a_1$ with k customers: a_1, a_2, a_3, $\ldots$, a_{k-1} and a_k. From constraints (3.2) and (3.3) of the mathematical model, we will have the two systems $(S1)$ and $(S2)$ respectively.

$$(S1)\begin{cases} y_{a_1a_2} - y_{a_ka_1} = p_{a_1} \\ y_{a_2a_3} - y_{a_1a_2} = p_{a_2} \\ \vdots \\ y_{a_ka_1} - y_{a_{k-1}a_k} = p_{a_k}. \end{cases} \qquad (S2)\begin{cases} z_{a_ka_1} - z_{a_1a_2} = d_{a_1} \\ z_{a_1a_2} - z_{a_2a_3} = d_{a_2} \\ \vdots \\ z_{a_{k-1}a_k} - z_{a_ka_1} = d_{a_k}. \end{cases}$$

The sum of the equations in $(S1)$ and the sum of the equations in $(S2)$ give $\sum_{l=1}^{k} p_{a_l} = 0$ $\sum_{l=1}^{k} d_{a_l} = 0$ respectively. As, $p_i \geq 0$ and $d_i \geq 0$ for all $i \in N$, then $p_i = 0$ and $d_i = 0$ for all $i \in N$, which contradicts our assumption. Therefore, the solutions of our mathematical model cannot contain sub-tours.

APPENDIX B. DETAILED RESULTS

The results of all the studied algorithms are reported in Table B.1. The first column displays the instance name, where *ins* stands for the original instance name (see [1]), n is the number of customers, Q is the capacity bound and m is the number of vehicles. The remaining notations are defined in a similar manner as in Table 12. A gap value in boldface refers to the best gap for the studied instance.

TABLE B.1. Detailed results of all the studied algorithms.

ins-n-Q-m	CPLEX			H^{1run}_{const}		H_{const}		ALNS		sALNS			
	UB	$\%^{UB}$	CPU	$\%^{cpx}$	CPU	$\%^{cpx}$	CPU	$\%^{cpx}$	CPU	$\%^{cpx}$	$\%^{UB}$	$\%^{ALNS}$	CPU
6-50-160-10	320.31	**0.00**	911.98	58.69	0.00	16.11	83.62	11.45	88.64	**0.00**	**0.00**	−12.93	39.56
7-75-140-20	708.96	12.45	7200.00	52.34	0.01	14.42	136.84	−2.62	181.13	**−7.3**	**6.06**	−4.56	54.27
8-100-200-15	777.38	5.50	7200.00	56.99	0.03	21.56	278.91	8.22	148.36	**−0.48**	**5.05**	−9.48	137.34
9-150-200-10	1430.16	41.12	7200.00	17.90	0.06	−34.68	713.52	−53.53	175.52	**−61.81**	**4.72**	−5.39	192.47
10-199-200-20	2166.39	100.00	7200.00	**−100.00**	0.13	**−100.00**	1378.49	**−100.00**	267.39	**−100.00**	**6.69**	−3.69	161.49
13-120-200-15	610.96	37.27	–	97.12	0.05	12.51	463.60	9.61	176.74	**−23.5**	**22.52**	−36.64	355.19
14-100-200-10	1133.48	11.18	7200.00	19.58	0.03	7.32	264.18	−2.18	112.86	**−5.02**	**6.72**	−2.77	84.58
15-150-200-15	1388.34	21.90	7200.00	19.63	0.06	−0.49	725.98	−15.67	201.53	**−21.63**	**5.01**	−5.15	151.57
16-199-200-15	2163.25	94.77	7200.00	−1220.36	0.13	−1404.72	1382.69	−1633.37	287.93	**−1700.19**	**5.8**	−3.86	168.23
6-50-50-2	74.29	**0.00**	65.34	111.32	0.00	28.38	17.38	5.01	154.03	**0.00**	**0.00**	−5.27	55.20
7-75-50-2	92.11	**0.00**	3249.42	86.44	0.00	29.67	31.08	7.93	289.83	**0.00**	**0.00**	−8.61	24.71
8-100-50-2	91.51	12.17	7200.00	145.19	0.01	47.21	60.17	14.12	404.24	**−3.07**	**9.47**	−20.02	38.26
9-150-50-2	132.89	**10.29**	7200.00	138.82	0.00	19.62	121.39	15.30	727.73	**0.00**	**10.29**	−18.06	73.41
10-199-50-2	143.51	9.39	7200.00	105.46	0.02	48.76	152.32	6.03	964.45	**−0.6**	**8.85**	−7.05	551.21
13-120-50-2	137.36	**5.23**	–	289.03	0.00	40.09	90.97	1.29	500.71	**0.00**	**5.23**	−1.31	71.72
14-100-50-2	107.47	**8.86**	7200.00	196.32	0.02	84.28	42.82	8.48	328.58	**0.00**	**8.86**	−9.27	138.60
15-150-50-2	125.20	23.93	7200.00	96.73	0.00	13.81	133.41	1.84	819.85	**−10.23**	**16.15**	−12.29	301.23
16-199-50-2	151.95	23.92	7200.00	68.32	0.01	20.40	168.48	−9.87	1118.25	**−14.08**	**13.2**	−3.83	257.02
6-50-50-3	101.50	**0.00**	1370.81	133.79	0.00	28.99	24.52	3.67	191.75	**0.00**	**0.00**	−3.80	24.60
7-75-50-3	142.30	9.85	7200.00	141.43	0.00	45.01	113.30	6.22	532.88	**−1.65**	**8.36**	−8.39	30.28
8-100-50-3	141.63	15.78	7200.00	114.03	0.01	48.88	81.71	15.26	560.94	**−0.01**	**15.77**	−18.02	43.47
9-150-50-3	193.09	**12.13**	7200.00	118.84	0.00	42.69	153.38	11.93	950.32	**0.44**	12.52	−13.05	326.25
10-199-50-3	215.51	18.92	7200.00	127.23	0.02	49.37	205.94	2.21	1400.97	**−8.74**	**11.83**	−11.20	1554.35
13-120-50-3	180.66	8.23	–	236.55	0.00	20.09	126.41	2.00	735.96	**−0.05**	**8.18**	−2.10	68.45
14-100-50-3	163.29	**15.73**	7200.00	177.86	0.01	106.08	54.12	8.70	514.35	**0.00**	**15.73**	−9.53	386.71
15-150-50-3	184.86	20.16	7200.00	143.39	0.02	44.23	159.15	6.95	1163.33	**−1.5**	**18.97**	−9.08	619.08
16-199-50-3	222.55	26.16	7200.00	89.68	0.01	12.35	231.08	−15.52	2294.87	**−16.59**	**13.91**	−0.93	250.69
6-50-50-4	131.55	13.30	–	70.02	0.00	16.33	30.38	3.15	211.58	**−1.91**	**11.64**	−5.22	27.06
7-75-50-4	184.82	14.69	7200.00	160.51	0.02	48.93	54.99	7.58	528.80	**−0.75**	**14.05**	−9.02	149.21
8-100-50-4	184.22	16.99	7200.00	127.76	0.00	55.98	109.26	11.86	733.41	**−0.09**	**16.91**	−13.55	53.81
9-150-50-4	244.85	**12.22**	7200.00	113.82	0.02	45.58	196.97	9.45	1231.30	**3.64**	15.41	−6.42	1391.83
10-199-50-4	286.57	24.57	7200.00	143.41	0.03	42.77	262.27	−2.56	1779.49	**−13.98**	**14.02**	−11.13	1890.69
13-120-50-4	205.53	11.82	–	295.44	0.01	58.03	156.62	7.36	813.76	**−1.07**	**10.88**	−9.09	86.79
14-100-50-4	197.89	24.21	7200.00	220.24	0.00	124.79	67.97	3.23	627.07	**−4.63**	**20.7**	−8.13	444.00
15-150-50-4	240.54	**19.64**	7200.00	114.83	0.02	43.47	197.23	11.56	1269.55	**0.77**	20.26	−12.20	117.68
16-199-50-4	288.94	27.77	7200.00	100.78	0.03	19.72	346.98	−6.66	2103.29	**−17.31**	**15.27**	−9.98	2232.42
6-50-75-2	131.71	**2.22**	7200.00	87.95	0.00	6.60	25.21	**0.00**	153.33	**0.00**	**2.22**	0.00	58.41
7-75-75-2	175.91	**1.97**	7200.00	89.50	0.00	21.03	41.95	2.66	434.68	**0.00**	**1.97**	−2.73	63.74
8-100-75-2	188.40	11.63	7200.00	77.33	0.02	21.78	80.34	0.23	425.11	**−1.12**	**10.64**	−1.35	72.28
9-150-75-2	234.40	19.81	7200.00	59.50	0.01	6.81	149.55	−4.17	722.07	**−7.94**	**13.44**	−3.63	284.70
10-199-75-2	259.15	18.76	7200.00	83.85	0.02	12.94	209.64	−4.04	1062.37	**−11.00**	**9.83**	−6.69	597.95
13-120-75-2	200.64	7.95	–	191.76	0.00	20.72	124.46	0.11	536.49	**−1.06**	**6.97**	−1.18	137.26
14-100-75-2	207.02	12.24	7200.00	95.23	0.02	56.67	54.97	**−1.88**	337.41	**−1.88**	**10.59**	0.00	312.52
15-150-75-2	224.71	13.21	7200.00	95.30	0.00	34.77	152.80	3.16	743.90	**−2.59**	**10.96**	−5.94	727.41
16-199-75-2	261.15	20.85	7200.00	58.25	0.02	5.58	223.27	−6.89	1256.67	**−11.46**	**11.79**	−4.27	213.37
6-50-75-3	187.08	2.88	7200.00	64.73	0.00	10.32	85.15	**−0.07**	165.61	**−0.07**	**2.81**	0.00	77.43
7-75-75-3	262.35	7.80	7200.00	97.73	0.00	29.47	147.00	2.80	480.24	**−0.4**	**7.43**	−3.30	51.32
8-100-75-3	280.70	17.37	7200.00	84.36	0.01	21.01	107.08	−3.11	550.24	**−5.12**	**13.14**	−1.95	531.91
9-150-75-3	337.07	21.50	7200.00	55.29	0.02	20.22	203.10	−6.42	1030.84	**−12.57**	**11.64**	−5.78	957.41
10-199-75-3	381.17	18.21	7200.00	75.73	0.02	17.64	290.02	−6.9	1495.39	**−9.5**	**10.44**	−2.43	1623.35
13-120-75-3	244.19	11.44	7200.00	274.51	0.01	81.04	177.22	14.72	709.52	**−1.56**	**10.06**	−19.09	173.12
14-100-75-3	293.85	17.75	7200.00	98.30	0.02	44.83	81.20	−0.4	463.89	**−1.1**	**16.85**	−0.70	547.76
15-150-75-3	331.42	19.55	7200.00	88.40	0.01	29.46	215.90	−4.17	1025.20	**−10.1**	**11.42**	−5.69	1059.11
16-199-75-3	381.47	20.34	7200.00	84.95	0.02	10.68	311.89	−8.64	1562.04	**−10.31**	**12.13**	−1.54	1491.62
6-50-75-4	223.79	12.06	–	102.02	0.02	2.83	39.32	1.99	196.89	**−2.69**	**9.7**	−4.78	28.88
7-75-75-4	326.24	9.93	7200.00	94.86	0.00	32.30	180.40	3.95	452.75	**−0.69**	**9.3**	−4.83	51.48
8-100-75-4	350.81	19.73	7200.00	61.73	0.03	34.45	132.04	1.56	987.78	**−5.12**	**15.62**	−6.78	591.81
9-150-75-4	436.45	26.50	7200.00	66.05	0.01	13.59	252.43	−14.6	1237.55	**−18.62**	**12.81**	−3.51	1093.71
10-199-75-4	494.68	29.19	7200.00	71.57	0.02	17.43	367.43	−22.49	1835.17	**−25.53**	**11.11**	−2.48	1954.93
13-120-75-4	265.00	18.42	7200.00	261.64	0.03	103.00	218.34	27.39	880.01	**−0.82**	**17.75**	−38.85	239.66
14-100-75-4	359.96	22.46	7200.00	126.27	0.02	48.07	106.43	−6.08	918.87	**−6.72**	**17.24**	−0.60	532.04
15-150-75-4	432.34	17.18	7200.00	80.48	0.01	34.25	263.62	−1.65	1177.79	**−3.99**	**13.88**	−2.30	621.64
16-199-75-4	494.95	32.05	7200.00	31.34	0.03	1.77	387.54	−24.19	1871.36	**−30.62**	**11.24**	−5.18	2023.83
6-50-100-2	189.62	**3.78**	7200.00	37.60	0.00	6.06	33.09	0.14	145.06	**0.00**	**3.78**	−0.14	87.57
7-75-100-2	252.08	**0.00**	6443.98	55.06	0.01	15.30	134.34	0.86	264.61	**0.00**	**0.00**	−0.87	98.81
8-100-100-2	282.20	9.45	7200.00	61.79	0.00	11.85	102.45	0.02	406.08	**−3.64**	**6.15**	−3.66	373.40
9-150-100-2	331.86	14.83	7200.00	68.15	0.02	4.80	194.13	−4.2	717.37	**−6.5**	**9.3**	−2.21	763.87
10-199-100-2	363.83	21.52	7200.00	62.51	0.01	4.22	271.48	−17.26	1098.47	**−18.04**	**7.36**	−0.67	1149.03

TABLE B.1. continued.

ins-n-Q-m	CPLEX			H^{1run}_{const}		H_{const}		ALNS		sALNS			
	UB	$\%^{UB}$	CPU	$\%^{cpx}$	CPU	$\%^{cpx}$	CPU	$\%^{cpx}$	CPU	$\%^{cpx}$	$\%^{UB}$	$\%^{ALNS}$	CPU
13-120-100-2	247.99	**8.9**	7200.00	130.69	0.00	50.97	168.03	3.39	539.02	**0.00**	**8.9**	−3.51	86.22
14-100-100-2	301.58	13.12	7200.00	65.73	0.00	35.23	75.54	**−0.42**	346.12	**−0.42**	**12.76**	0.00	351.65
15-150-100-2	324.59	19.57	7200.00	57.28	0.01	13.75	200.73	−8.32	716.00	**−11.93**	**9.97**	−3.33	779.67
16-199-100-2	367.92	29.50	7200.00	60.41	0.02	−10.99	287.00	−20.92	1081.64	**−25.89**	**11.24**	−4.11	313.92
6-50-100-3	251.97	**4.64**	7200.00	57.58	0.00	12.29	41.66	7.98	152.19	**0.00**	**4.64**	−8.67	28.00
7-75-100-3	369.46	8.92	7200.00	63.38	0.01	14.12	181.18	0.36	316.90	**−1.79**	**7.29**	−2.15	353.71
8-100-100-3	397.92	12.85	7200.00	49.37	0.02	14.80	138.92	2.16	516.27	**−2.61**	**10.57**	−4.87	448.71
9-150-100-3	483.41	20.58	7200.00	61.72	0.02	5.60	262.86	−10.86	992.21	**−14.69**	**8.91**	−3.46	1064.25
10-199-100-3	534.03	15.89	7200.00	70.75	0.03	12.81	379.44	−5.93	1532.58	**−9.08**	**8.25**	−2.97	1609.77
13-120-100-3	314.81	24.73	7200.00	132.89	0.01	68.26	229.85	8.61	634.27	**−5.98**	**20.23**	−15.97	199.71
14-100-100-3	422.27	17.78	7200.00	40.95	0.00	30.71	106.00	−0.74	677.68	**−1.29**	**16.72**	−0.54	365.88
15-150-100-3	475.73	12.40	7200.00	83.07	0.03	15.87	275.62	0.74	950.99	**−4.92**	**8.09**	−5.71	1102.24
16-199-100-3	534.67	20.46	7200.00	50.15	0.03	1.72	399.75	−9.55	2204.22	**−13.55**	**9.69**	−3.66	1153.19
6-50-100-4	285.70	12.37	–	82.35	0.00	17.25	59.21	−1.07	117.45	**−4.78**	**8.18**	−3.67	32.78
7-75-100-4	452.14	11.89	7200.00	60.86	0.01	11.15	93.94	−0.85	363.35	**−4.71**	**7.74**	−3.83	310.26
8-100-100-4	493.59	17.80	7200.00	57.56	0.02	15.10	183.98	−0.46	601.87	**−6.19**	**12.72**	−5.70	523.06
9-150-100-4	621.41	22.41	7200.00	63.48	0.03	6.73	346.95	−11.36	1126.15	**−14.71**	**11.00**	−3.00	983.30
10-199-100-4	691.78	28.93	7200.00	68.33	0.03	3.00	505.25	−24.33	1856.49	**−27.13**	**9.64**	−2.26	3987.88
13-120-100-4	343.80	30.72	7200.00	275.39	0.03	117.18	273.10	9.76	796.74	**−13.56**	**21.32**	−25.84	240.75
14-100-100-4	529.81	22.41	7200.00	42.55	0.01	26.67	130.44	−3.96	531.08	**−4.83**	**18.67**	−0.84	451.15
15-150-100-4	610.57	15.06	7200.00	58.64	0.03	16.28	330.23	−4.92	1133.26	**−7.42**	**8.76**	−2.37	1226.91
16-199-100-4	695.01	24.06	7200.00	31.20	0.03	5.23	491.18	−11.26	1854.61	**−19.52**	**9.24**	−7.42	4341.49
6-50-160-2	291.51	**0.00**	654.86	26.54	0.01	9.52	48.34	5.01	112.28	**0.00**	**0.00**	−5.27	71.23
7-75-140-2	375.96	3.17	7200.00	55.96	0.00	9.39	173.85	1.07	390.16	**−0.76**	**2.43**	−1.85	199.67
8-100-200-2	568.60	10.30	7200.00	42.22	0.02	6.91	205.47	**−5.29**	623.76	−4.27	**6.47**	0.97	434.40
9-150-200-2	700.42	14.59	7200.00	36.04	0.03	−2.64	403.70	−8.71	772.06	**−9.08**	**6.83**	−0.34	869.22
10-199-200-2	772.22	18.22	7200.00	40.34	0.03	−3.36	621.16	−13.88	1156.45	**−14.72**	**6.18**	−0.74	1257.43
13-120-200-2	475.20	24.46	7200.00	138.41	0.03	37.73	350.81	−9.13	472.45	**−9.84**	**17.03**	−0.65	497.88
14-100-200-2	646.13	11.27	7200.00	40.80	0.00	15.11	159.68	**−0.19**	334.11	**−0.19**	**11.1**	0.00	311.42
15-150-200-2	679.83	7.89	7200.00	29.40	0.03	8.88	416.60	−1.09	798.48	**−3.35**	**4.81**	−2.23	830.04
16-199-200-2	760.76	12.46	7200.00	35.13	0.05	1.62	580.23	−5.64	1131.63	**−7.21**	**6.15**	−1.48	1275.39
6-50-160-3	320.31	**0.00**	3248.91	26.26	0.00	15.16	53.92	5.35	97.41	**0.00**	**0.00**	−5.66	36.84
7-75-140-3	506.04	10.20	7200.00	42.97	0.02	12.33	231.44	−4.25	384.98	**−4.45**	**6.21**	−0.19	260.67
8-100-200-3	719.14	11.77	7200.00	54.38	0.02	9.74	252.36	−6.38	406.59	**−9.15**	**3.69**	−2.61	110.62
9-150-200-3	975.64	15.05	7200.00	42.73	0.03	4.88	518.02	−8.03	879.10	**−11.4**	**5.37**	−3.12	697.61
10-199-200-3	1094.48	25.08	7200.00	24.11	0.05	−4.81	1233.32	−20.5	1480.06	**−22.34**	**8.35**	−1.52	1507.86
13-120-200-3	595.72	35.95	7200.00	109.37	0.05	40.35	642.66	−16.67	343.44	**−23.69**	**20.78**	−6.01	370.13
14-100-200-3	885.82	14.67	7200.00	49.44	0.02	5.42	324.44	−2.71	443.57	**−3.84**	**11.39**	−1.10	355.40
15-150-200-3	943.14	20.44	7200.00	33.98	0.03	0.90	793.81	−16.17	817.76	**−20.57**	**4.07**	−3.79	982.33
16-199-200-3	1082.56	30.02	7200.00	28.16	0.05	−17.74	1205.36	−29.82	1404.58	**−32.77**	**7.09**	−2.27	1565.73
6-50-160-4	320.31	**0.00**	2956.28	53.73	0.01	16.11	85.45	11.45	49.16	**0.00**	**0.00**	−12.93	37.41
7-75-140-4	605.26	16.77	7200.00	45.15	0.02	6.44	399.91	−10.83	385.05	**−11.86**	**6.89**	−0.93	206.12
8-100-200-4	776.30	11.00	7200.00	37.88	0.03	15.56	442.58	1.23	218.38	**−8.2**	**3.71**	−9.55	107.98
9-150-200-4	1177.92	20.99	7200.00	20.52	0.06	−0.49	950.05	−16.79	918.31	**−18.63**	**6.27**	−1.57	705.87
10-199-200-4	1366.78	38.73	7200.00	6.04	0.06	−27.57	929.45	−47.93	1510.91	**−51.15**	**7.39**	−2.18	1530.40
13-120-200-4	610.15	33.63	7200.00	104.82	0.03	17.18	455.74	11.90	340.17	**−17.11**	**22.28**	−32.93	234.93
14-100-200-4	1064.50	17.95	7200.00	37.57	0.03	5.56	266.69	1.14	226.01	**−8.07**	**11.33**	−9.31	218.15
15-150-200-4	1141.46	19.49	7200.00	28.63	0.11	−0.55	648.02	−14.15	802.87	**−16.75**	**6.00**	−2.28	720.01
16-199-200-4	1361.27	100.00	7200.00	**−100.00**	0.16	**−100.00**	934.11	**−100.00**	1576.37	**−100.00**	**7.18**	−2.66	1654.06
Avg		18.05	6869.24	73.23	0.02	9.10	279.23	−17.97	744.05	−24.16	9.76	−5.99	582.46

Acknowledgements. The authors are very grateful to Pr. Noureddine Hannoun for the English editing and for Dr. Matteo Salani for kindly providing us with VRPSDP benchmark instances. The authors would also like to thank the reviewers for their valuable comments and suggestions.

REFERENCES

[1] C. Archetti, D. Feillet, A. Hertz and M.G. Speranza, The capacitated team orienteering and profitable tour problems. *J. Oper. Res. Soc.* **60** (2009) 831–842.

[2] R. Bent and P. Van Hentenryck, A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Comput. Oper. Res.* **33** (2006) 875–893.

[3] B. Çatay, A new saving-based ant algorithm for the vehicle routing problem with simultaneous pickup and delivery. *Expert Syst. Appl.* **37** (2010) 6809–6817.

[4] V. Černỳ, Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *J. Optim. Theory Appl.* **45** (1985) 41–51.
[5] N. Christofides, A. Mingozzi and P. Toth, The vehicle routing problem, in Combinatorial Optimization, edited by N. Christofides, A. Mingozzi, P. Toth and C. Sandi. Wiley, Chichester (1979) 315–338.
[6] M. Dell'Amico, G. Righini and M. Salani, A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. *Transp. Sci.* **40** (2006) 235–247.
[7] M. Gansterer, M. Küçüktepe and R.F. Hartl, The multi-vehicle profitable pickup and delivery problem. *OR Spectr.* **39** (2017) 303–319.
[8] V. Ghilas, E. Demir and T. Van Woensel, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. *Comput. Oper. Res.* **72** (2016) 12–30.
[9] I. Gribkovskaia, G. Laporte and A. Shyshou, The single vehicle routing problem with deliveries and selective pickups. *Comput. Oper. Res.* **35** (2008) 2908–2924.
[10] G. Gutiérrez-Jarpa, G. Desaulniers, G. Laporte and V. Marianov, A branch-and-price algorithm for the vehicle routing problem with deliveries, selective pickups and time windows. *Eur. J. Oper. Res.* **206** (2010) 341–349.
[11] M. Hifi and L. Wu, A hybrid metaheuristic for the vehicle routing problem with time windows, in *2014 International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE (2014) 188–194.
[12] S.C. Ho and W. Szeto, Grasp with path relinking for the selective pickup and delivery problem. *Expert Syst. Appl.* **51** (2016) 14–25.
[13] M.K. Jepsen, B. Petersen, S. Spoorendonk and D. Pisinger, A branch-and-cut algorithm for the capacitated profitable tour problem. *Discret. Optim.* **14** (2014) 78–96.
[14] C.B. Kalayci and C. Kaya, An ant colony system empowered variable neighborhood search algorithm for the vehicle routing problem with simultaneous pickup and delivery. *Expert Syst. Appl.* **66** (2016) 163–175.
[15] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing. *Science* **220** (1983) 671–680.
[16] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Am. Math. Soc.* **7** (1956) 48–50.
[17] Y. Li, H. Chen and C. Prins, Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests. *Eur. J. Oper. Res.* **252** (2016) 27–38.
[18] D. Männel and A. Bortfeldt, A hybrid algorithm for the vehicle routing problem with pickup and delivery and three-dimensional loading constraints. *Eur. J. Oper. Res.* **254** (2016) 840–858.
[19] H. Min, The multiple vehicle routing problem with simultaneous delivery and pick-up points. *Transp. Res. Part A: General* **23** (1989) 377–386.
[20] F.A.T. Montané and R.D. Galvao, A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *Comput. Oper. Res.* **33** (2006) 595–619.
[21] D. Mu, C. Wang, F. Zhao and J.W. Sutherland, Solving vehicle routing problem with simultaneous pickup and delivery using parallel simulated annealing algorithm. *Int. J. Shipp. Transp. Logist.* **8** (2016) 81–106.
[22] D. Pisinger and S. Ropke, A general heuristic for vehicle routing problems. *Comput. Oper. Res.* **34** (2007) 2403–2435.
[23] X. Qiu, S. Feuerriegel and D. Neumann, Making the most of fleets: a profit-maximizing multi-vehicle pickup and delivery selection problem. *Eur. J. Oper. Res.* **259** (2017) 155–168.
[24] S. Ropke and D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transp. Sci.* **40** (2006) 455–472.
[25] S. Ropke and D. Pisinger, A unified heuristic for a large class of vehicle routing problems with backhauls. *Eur. J. Oper. Res.* **171** (2006) 750–775.
[26] S. Salhi and G. Nagy, A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *J. Oper. Res. Soc.* **3**5 (1999) 1034–1042.
[27] M.M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.* **35** (1987) 254–265.
[28] A. Subramanian, L.M.D.A. Drummond, C. Bentes, L.S. Ochi and R. Farias, A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Comput. Oper. Res.* **37** (2010) 1899–1911.
[29] C.-K. Ting and X.-L. Liao, The selective pickup and delivery problem: formulation and a memetic algorithm. *Int. J. Prod. Econom.* **141** (2013) 199–211.
[30] VRPLIB, Available at: http://or.dei.unibo.it/library/vrplib-vehicle-routing-problem-library
[31] E.E. Zachariadis and C.T. Kiranoudis, A local search metaheuristic algorithm for the vehicle routing problem with simultaneous pick-ups and deliveries. *Expert Syst. Appl.* **38** (2011) 2717–2726.
[32] E.E. Zachariadis, C.D. Tarantilis and C.T. Kiranoudis, A hybrid metaheuristic algorithm for the vehicle routing problem with simultaneous delivery and pick-up service. *Expert Syst. Appl.* **36** (2009) 1070–1081.
[33] E.E. Zachariadis, C.D. Tarantilis and C.T. Kiranoudis, An adaptive memory methodology for the vehicle routing problem with simultaneous pick-ups and deliveries. *Eur. J. Oper. Res.* **202** (2010) 401–411.