European Journal of Operational Research 252 (2016) 27–38

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests

Yuan Li*, Haoxun Chen, Christian Prins

ICD-LOSI, UMR CNRS 6281, Université de Technologie de Troyes 12 rue Marie Curie, CS 42060, 10004 Troyes Cedex, France

ARTICLE INFO

Article history:
Received 27 August 2015
Accepted 16 December 2015
Available online 23 December 2015

Keywords:
Vehicle routing problem
Pickup and delivery problem
Time window
Profit
Adaptive large neighborhood search

ABSTRACT

This paper addresses the Pickup and Delivery Problem with Time Windows, Profits, and Reserved Requests (PDPTWPR), a new vehicle routing problem appeared in carrier collaboration realized through Combinatorial Auction (CA). In carrier collaboration, several carriers form an alliance and exchange some of their transportation requests. Each carrier has reserved requests, which will be served by itself, whereas its other requests called selective requests may be served by the other carriers. Each request is a pickup and delivery request associated with an origin, a destination, a quantity, two time windows, and a price for serving the request paid by its corresponding shipper. For each carrier in CA, it has to determine which selective requests to serve, in addition to its reserved requests, and builds feasible routes to maximize its total profit. A Mixed-Integer Linear Programming (MILP) model is formulated for the problem and an adaptive large neighborhood search (ALNS) approach is developed. The ALNS involves ad-hoc destroy/repair operators and a local search procedure. It runs in successive segments which change the behavior of operators and compute their own statistics to adapt selection probabilities of operators. The MILP model and the ALNS approach are evaluated on 54 randomly generated instances with 10–100 requests. The computational results indicate that the ALNS significantly outperforms the solver, not only in terms of solution quality but also in terms of CPU time.

1. Introduction

In recent years, the fierce competition in global markets, the introduction of products with shorter life cycles, the increasing fuel costs and labor prices, the growing transport legislation and the heightened expectations of customers have shrunk profit margins of shippers and carriers (Cruijssen, Cools, & Dullaert, 2007). Thus, as an effective way to cut empty backhauls and to increase vehicle utilization rate, Collaborative Logistics (CL) attracted a growing interest from industrial practitioners and academic research (Dai & Chen, 2009b). In CL, shippers, carriers, contractors and even competitors can be partners if their collaboration can create a win-win outcome.

The collaboration among small or medium sized enterprises (SME) plays a growing role in their daily operation/management. Participation in a network and collaboration with other enterprises has now become an inevitable strategy for them to gain competitive advantages in current severe environment. To achieve economies of scale, more and more SMEs have formed collaborative networks by sharing distribution tasks and resources, in order to reduce costs, improve responsiveness to the evolution of market demands, and capture more business opportunities.

In our study, we assume that several carriers form an alliance. This alliance aims at maximizing the total profit so as to generate more profit for each carrier. Inside this alliance, the carriers achieve collaboration by exchanging part of their transportation requests. We are developing a general two-step approach for carrier collaboration (*e.g.* Dai and Chen, 2009a; 2009b). The first step is reassignment/reallocation of requests among carriers, whereas the second aims at sharing the resulting profits among them (Dai & Chen, 2012, 2015). The task reassignment is realized through Combinatorial Auction (CA). In each iteration of CA, the auctioneer sets/updates a service price (revenue) for each request to be exchanged among carriers (Dai, Chen, & Yang, 2014), then each carrier determines which requests to bid for (to serve), in order to maximize its own profit. The latter problem is referred to Bid Generation Problem (BGP) (*e.g.* Wang and Xia, 2005; Lee, Kwon, and Ma, 2007; Buer, 2014; Triki, Oprea, Beraldi, and Crainic, 2014) for each carrier.

In this paper, we consider the BGP (or request selection problem) for each carrier in CA (Dai & Chen, 2011). It is assumed that each carrier has a set of *reserved requests* (*i.e.*, not proposed for

* Corresponding author. Tel.: +33 751610975.
E-mail addresses: yuan.li@utt.fr, yuan296103@gmail.com (Y. Li), haoxun.chen@utt.fr (H. Chen), christian.prins@utt.fr (C. Prins).

http://dx.doi.org/10.1016/j.ejor.2015.12.032
0377-2217/

exchange in CA) and can serve additional requests (*selective requests*) from other carriers. Each request is a pickup and delivery request associated with an origin, a destination, a quantity, two time windows, and a price (revenue) for serving the request paid by its corresponding shipper (a customer of the carrier). Two different decisions have to be simultaneously taken by the carrier: Which requests to bid for (to serve) and how to build routes for maximizing its own profit, equals to the sum of collected revenues minus the total cost of the routes. This raises a new Pickup and Delivery Problem (PDP) with Time Windows, Profits, and Reserved Requests (PDPTWPR). To the best of our knowledge, this problem was rarely studied in the literature.

The rest of the paper is organized as follows. A brief literature review on CL and Vehicle Routing Problems (VRP) with Profits (VRPP) is provided in Section 2. Section 3 defines the problem and provides a mathematical model. An adaptive large neighborhood search (ALNS) is developed in Section 4. Section 5 proposes a set of randomly generated instances and compares the results of our ALNS with the ones obtained by the CPLEX solver on the mathematical model. Finally, Section 6 provides a quick conclusion with some remarks for future research.

2. Literature review

Horizontal collaborative logistics refers to the collaboration among multiple actors at the same level in logistics operations such as the collaboration among shippers (manufacturers) and the collaboration among carriers. Two types of horizontal collaborative logistics are studied in the literature: shipper collaboration and carrier collaboration. Shipper collaboration considers a single carrier and multiple shippers. The collaboration among shippers is realized by consolidation of their transportation requests to be offered to carriers. Through collaboration, shippers are able to reduce "hidden costs" such as asset reposition costs (Ergun, Kuyzu, & Savelsbergh, 2007). However, more attention has been focused on carrier collaboration. Differing from shipper collaboration, carrier collaboration considers how to provide opportunities for LTL carriers to exploit synergies in operations (such as excess capacity), reduce costs associated with fleet operation, decrease lead times, increase asset utilization (power units), and enhance overall service levels (Esper & Williams, 2003; Hernández, Peeta, & Kalafatas, 2011).

Analyzing the existing scientific literature on carrier collaboration reveals that the majority of logistics cooperation research can be classified into two main categories according to the two ways of cooperation among carriers (Verdonck, Caris, Ramaekers, & Janssens, 2013). One way is to share resources, like vehicle capacities. The other is order sharing which is known to be able to improve efficiency and profitability owing to increment of capacity utilization and reduction of asset reposition by re-drafting the route planning (Dai & Chen, 2011). Verdonck et al. (2013) provide a detailed literature review of the two above carrier collaboration approaches.

Moreover, some interesting opportunities for CL are identified. Cruijssen et al. (2007) present the results of a large-scale survey on the potential benefits and impediments for horizontal cooperation and emphasize its importance. Krajewska and Kopfer (2009) apply a tabu search metaheuristic to solve an Integrated Operational Transportation Planning problem (IOTP) in which a carrier has the possibility of outsourcing its requests by involving subcontractors. Wang, Kopfer, and Gendreau (2014) use the ALNS framework proposed by Ropke and Pisinger (2006) and two other heuristics to solve the previous IOTP and its generalization to a centralized planning for all carriers. References (Krajewska & Kopfer, 2009) and (Wang et al., 2014) differ from our work, because, in addition to each request, complete routes can be subcontracted with either a payment per route or on a daily basis. Moreover, reserved requests are not considered.

Our problem is related to the Pickup and Delivery Problem with Time Windows (PDPTW), which itself is a generalization of the Vehicle Routing Problem with Time Windows (VRPTW). The PDPTW involves three main constraints: time window constraints, capacity constraints and coupling constraints (the delivery node of each request must be visited after its corresponding pickup node in the same route). The PDPTW has been well studied in the literature and due to its complexity, metaheuristic algorithms have become dominating methods for its resolution. Nanry and Wesley Barnes (2000) propose a reactive tabu search and test it on instances with up to 50 requests. Li and Lim (2003) create a set of benchmark instances and propose a hybrid metaheuristic. Hosny and Mumford (2012) compare sequential and parallel insertion heuristics to provide metaheuristics with high quality initial solutions. Bent and Hentenryck (2006) apply Variable Neighborhood Search (VNS) to the PDPTW and their computational results show promising performance of their algorithm, compared with the previous PDPTW metaheuristics. Ropke and Pisinger (2006) design an ALNS algorithm which is probably the most effective metaheuristic for the PDPTW so far, with results reported for up to 1000 locations.

Our PDPTWPR displays important differences with the PDPTW: i) serving all requests is not mandatory (provided all reserved requests are treated), ii) a profit is associated with each request, and iii) the objective function, to be maximized, is the sum of the revenues minus the routing costs. We found no reference on this problem in the literature, although a growing number of publications deals with Vehicle Routing Problems with Profits (VRPP) in general.

Single-vehicle problems with profits are surveyed in Feillet, Dejax, and Gendreau (2005). Tour costs and collected profits can be expressed in the objective function, by minimizing the travel costs minus the profits, giving the Profitable Tour Problem (PTP). The profits collected can be maximized, subject to a maximum tour length, which defines the Orienteering Problem (OP). Conversely, in the Prize-Collecting Traveling Salesman Problem (PCTSP) (Balas, 1989), the travel costs are minimized but the collected profits cannot be less than a given constant.

Among these problems, the PTP has the same objective function as our PDPTWPR. Only heuristics are available to solve it. Nguyen and Nguyen (2010) develop an approximation algorithm, based on the heuristic from Frieze, Galbiati, and Maffioli (1982) for the Asymmetric Traveling Salesman Problem (ATSP), and a method to round fractional solutions of a linear programming relaxation for the asymmetric PTP. Goemans and Bertsimas (1990) solve an undirected version of the PTP.

Routing problems with multiple vehicles and profits are much less studied. Butt and Cavalier (1994) define the Multiple Tour Maximum Collection Problem (MTMCP), a generalization of the OP where the same maximum tour length is applied to several vehicles. Chao, Golden, and Wasil (1996) study the same problem but introduce a nowadays standard name, the Team Orienteering Problem (TOP). A few recent papers have tackled the TOP with Time Windows (TOPTW), see for instance (Labadie, Mansini, Melechovsky, & Wolfler-Calvo, 2012) who develop a granular variable neighborhood search. The TOPTW is close to our PDPTWPR but does not distinguish between pickup and delivery nodes. A recent paper by Archetti, Corberan, Sanchis, Plana, and Speranza (2014) presents the Team Orienteering Arc Routing Problem (TOARP), but in a truckload (TL) context: since each vehicle can transport one request at a time, each request can be modeled by one arc, which leads to an Arc Routing Problem (ARP).

For more details on VRPPs, we refer readers to the technical report written by Archetti, Speranza, and Vigo (2013).

3. Problem definition and mathematical model

The PDPTWPR is based on a complete undirected graph $G = (N, E)$. The node-set is defined as $N = \{0, \ldots, 2n+1\}$, where n denotes the number of requests. Nodes 0 and $2n+1$ represent the depot of the carrier, hosting a set $K = \{1, \ldots, m\}$ of m identical vehicles of capacity Q. It is assumed that each vehicle route begins at node 0 and ends at node $2n+1$. Each node i has a time window $[a_i, b_i]$ to begin service, while each edge (i, j) in E has a travel cost c_{ij} and a travel time t_{ij}. The service times at node i is included in the t_{ij}'s. Like in the VRPTW, a vehicle can wait at customer i if it arrives before a_i. The subset $P = \{1, \ldots, n\}$ contains the pickup nodes of all requests, while $D = \{n+1, \ldots, 2n\}$ gathers delivery nodes. Request i, $i = 1, \ldots, n$, is associated with a pickup node i, a delivery node $n+i$, a demand $d_i > 0$ and a price p_i (customer payment). For the delivery node, we set $d_{n+i} = -d_i$. The set R of all requests includes the subset of reserved requests R_r and the subset of selective requests R_s.

The goal is to determine the selective requests to be served, in addition to the reserved requests, and to determine the associated vehicle routes, to maximize the total profit which is equal to the sum of collected payments minus the total cost of the routes. The demand served in a route cannot exceed vehicle capacity, the time window at each node must be respected, and the delivery node of each request must be visited after its corresponding pickup node, in the same route.

The problem is NP-hard in strong sense like the PDPTW which is the particular case where R_s is empty and all prices p_i are equal to a large positive constant M (to ensure that all requests are served). It can be formulated as a Mixed-Integer Linear Program (MILP). In addition to previous data, we need two symbols to write the model more easily: $T_{ij} = b_j - a_i$ plays the role of a big-M constant in the time window constraints, while $Q_i = Q + d_i$ is used in the capacity constraints. The following decision variables are used:

- x_{ij}^k, binary variable equals to 1 if and only if vehicle k travels directly through arc (i, j),
- y_i^k, binary variable equals to 1 if and only if request i is served by vehicle k,
- T_i^k, time at which vehicle k begins service at node i,
- Q_i^k, load of vehicle k when leaving node i.

Resulting model:

$$\max \sum_{k\in K}\sum_{i\in R} p_i \cdot y_i^k - \sum_{k\in K}\sum_{i\in N}\sum_{j\in N} c_{ij} \cdot x_{ij}^k \tag{1}$$

Subject to:

$$\sum_{j\in N, j\neq i} x_{ji}^k - \sum_{j\in N, j\neq i} x_{ij}^k = 0 \qquad \forall i \in P \cup D, \forall k \in K \tag{2}$$

$$\sum_{j\in P, j\neq 0} x_{0j}^k = 1 \qquad \forall k \in K \tag{3}$$

$$\sum_{i\in D, i\neq 2n+1} x_{i,2n+1}^k = 1 \qquad \forall k \in K \tag{4}$$

$$\sum_{k\in K} y_i^k = 1 \qquad \forall i \in R_r \tag{5}$$

$$\sum_{k\in K} y_i^k \leq 1 \qquad \forall i \in R_s \tag{6}$$

$$\sum_{j\in N, j\neq i, 2n+1} x_{ij}^k = y_i^k \qquad \forall i \in P, \forall k \in K \tag{7}$$

$$\sum_{j\in N, j\neq i, 0} x_{j,n+i}^k = y_i^k \qquad \forall i \in P, \forall k \in K \tag{8}$$

$$T_i^k + t_{i,n+i} \leq T_{n+i}^k \qquad \forall i \in P, \forall k \in K \tag{9}$$

$$T_j^k \geq T_i^k + t_{ij} \cdot x_{ij}^k - T_{ij} \cdot (1 - x_{ij}^k) \qquad \forall i, j \in N, \forall k \in K \tag{10}$$

$$a_i \leq T_i^k \leq b_i \qquad \forall i \in N, \forall k \in K \tag{11}$$

$$Q_j^k \geq Q_i^k + d_j - Q_j \cdot (1 - x_{ij}^k) \qquad \forall i, j \in N, \forall k \in K \tag{12}$$

$$\max\{0, d_i\} \leq Q_i^k \leq \min\{Q, Q + d_i\} \qquad \forall i \in N, \forall k \in K \tag{13}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall i, j \in N, \forall k \in K \tag{14}$$

$$y_i^k \in \{0, 1\} \qquad \forall i \in R, \forall k \in K \tag{15}$$

$$T_i^k \geq 0 \qquad \forall i \in N, \forall k \in K \tag{16}$$

$$Q_i^k \geq 0 \qquad \forall i \in N, \forall k \in K \tag{17}$$

The objective function (1) represents the total profit of the carrier, equals to the difference between the sum of payments of served requests and the total transportation cost. Constraints (2) ensures that a vehicle arriving at a pickup or delivery node has to leave it. Constraints (3) and (4) indicate that each vehicle leaves the depot and returns to it. Constraints (5) guarantees that all reserved requests must be served once, whereas in constraints (6) selective requests are served at most once. Constraints (7) and (8) ensure that if a request is served, there must be a vehicle leaving its pickup node and arriving at its paired delivery node. Time windows and precedence relations are respected via constraints (9)–(11). Constraints (12) and (13) concern vehicle capacity. Finally, constraints (14)–(17) define the variables.

4. Solution approach

For classical vehicle routing problems (without profits), the small-scale moves used in local search procedures affect the partition of customers in routes and the sequence of these routes. In problems with profits, other moves are required to modify the set of served requests since it is not mandatory to serve all of them. As the possibility of choosing requests tremendously expands solution space, the two kinds of moves must be combined in a clever way to avoid excessive running time. For instance, Labadie et al. (2012) propose for the TOPTW a VNS where a local search procedure focuses on route sequences, while the shaking step changes the subset of served requests. For our PDPTWPR, we selected the *adaptive large neighborhood search* (ALNS) framework as another way to remedy the weak efficiency of small-scale neighborhoods.

The precursor of the ALNS is Large Neighborhood Search (LNS), introduced by Shaw (1998) for the Capacitated Vehicle Routing Problem (CVRP). LNS begins with an initial solution and improves the objective value gradually, by applying one destroy and one repair operator at each iteration. The destroy operator is a randomized heuristic removing a small subset of customers. The repair operator reinserts these customers optimally, using constraint programming and branch-and-bound, see Bent and Hentenryck (2006) for the VRPTW. The destroy and repair operators are also called ruin and recreate operators, or removal and insertion operators.

The application of a destroy/repair pair can be viewed as a move that implicitly defines a very large neighborhood. However, only one move is randomly selected at each iteration instead of exploring the neighborhood completely. LNS is conceptually simple but has some known drawbacks. The search is a bit blind because the destroy/repair moves sample a very small fraction of the large neighborhood. This can be compensated by more iterations but, added to the exact method used to reinsert customers, the metaheuristic becomes time-consuming.

Ropke and Pisinger (2006) proposed an ALNS to improve LNS. The ALNS involves several destroy and repair operators, which are all heuristics to achieve a time-saving purpose. At each iteration, a pair of operators is randomly chosen to make a move and statistics are computed to favor the most efficient pairs. The method is adaptive since the most frequent pairs may change during the search. The ALNS has been successfully applied to the PDPTW (Ropke & Pisinger, 2006) and later to various rich vehicle routing problems (*e.g.* Pisinger and Ropke, 2007; Aksen, Kaya, Salman, and Tüncel, 2014).

Our ALNS is inspired by Ropke and Pisinger (2006) but we brought six important modifications to cope with the peculiarities of our problem and achieve a good efficiency. The first one is the design of specific destroy/repair operators, which acts both on the sequence of the routes and on the selection of served requests. The second change is to restart the ALNS from several initial solutions. The iterations for one initial solution define a *run*. The third modification is to organize the search in what we call successive *segments*. The behavior of operators is modified at each new segment. At the beginning of the ALNS, both reserved and selective requests can be removed and inserted, even when this is not profitable. Then, gradually, less and less reserved requests may be removed and more and more profitable insertions are preferred. The fourth change is a diversification technique called meta-destroy: When the search is stalled, moves combining two destroy and one repair operators are tried. The two last changes are not really new since they are used in some recent ALNS implementations like Aksen et al. (2014): a local search procedure is added for intensification and the simple descent is replaced by a Simulated Annealing (SA) loop.

The general structure of our algorithm is sketched in Algorithm 1 and its components are detailed in the following sections. The main loop performs *nruns* independent runs to find a global best solution S^* and its costs $f(S^*)$. Each run calls a sequential insertion heuristic: *SIH* (Section 4.1) which initializes the current best solution S_{best} of the run, sets the SA temperature T to its initial value T_{beg} and initializes the weights and scores of operators. The weight and score system is explained in Section 4.2 while the different destroy/repair operators used are presented in Section 4.4. The second loop executes *nsegs* successive segments. Each segment selects different possible behaviors of destroy/repair operators, as explained in Section 4.3, performs *niters* the ALNS iterations (third loop) and ends by calling a local search procedure *LS* (Section 4.6) and updating the global best solution in case of improvement. Each ALNS iteration selects a combination of operators and executes the corresponding move and improves the incumbent solution or satisfies the SA criterion, where $r^{U(0,1)}$ is random value between 0 and 1 generated according to the uniform distribution $U(0, 1)$. The SA scheme is commented in Section 4.5.

Algorithm 1 – Pseudo-code of our ALNS metaheuristic: *ALNS*(S^*).

```
f(S*) ← ∞
for run ← 1 to nruns do
  call SIH(S) (Algorithm 2)
  S_best ← S
  T ← T_beg
  initialize weights and scores of operators (Section 4.2)
  for seg ← 1 to nsegs do
    select the behavior of operators (Section 4.3)
    for iter ← niters do
      select one destroy operator and one repair operator or
      two destroy operators if S_best is not improved in the last
      consecutive δ iterations of the run (Section 4.4)
      apply selected move S → S'
      if S' feasible and (f(S') > f(S) or r^U(0,1) <
      exp^((f(S')−f(S))/T) (Section 4.5) then
        S ← S'
        update performance scores of selected operators
        (Section 4.2)
        if f(S) > f(S_best) then S_best ← S endif
      end if
      T ← T × θ
    end for
    update the weights of the ALNS moves and reset scores
    (Section 4.2)
    call LS(S) (Section 4.6)
  end for
  if f(S_best) > f(S*) then S* ← S_best endif
end for
```

4.1. Initial solution construction

According to our experimental results, the quality of the initial solution may produce a crucial impact on the final outcome of the ALNS and its computational time. Consequently, we developed an effective sequential insertion heuristic (*SIH*) to provide quickly each run with a high-quality initial solution. *SIH* can be controlled using three insertion policies:

- *Policy 1*: Only the reserved requests are served.
- *Policy 2*: After the insertion of all reserved requests, only profitable selective requests can be inserted. The insertion procedure will stop if no profitable selective request can be found any more.
- *Policy 3*: After the insertion of all reserved requests, selective requests are inserted until infeasibility.

Policy 1 is the most basic way to construct a feasible initial solution, by including the whole set of reserved requests. In general, *Policy 2* produces a better initial solution than the others, but we observed in a few cases that a near-optimal initial solution may lead to be trapped in a local optimum at the very beginning of the ALNS searching process. In comparison with the two first policies, *Policy 3* tends to exhaust vehicle fleet capacity. For each run of the ALNS, *SIH* randomly select Policy 1, 2 or 3 with respective probabilities γ_1, γ_2 and γ_3, with $\gamma_1 + \gamma_2 + \gamma_3 = 1$.

As shown in Algorithm 2, *SIH* builds one route at a time. The reserved requests are first sorted in decreasing order of price and marked as unserved. The sorted list is browsed and the existence of at least one feasible insertion is checked for the incumbent request i. If feasible insertions exist in the current route, the request is marked as served and its two nodes i and $n+i$ are inserted in the most profitable position. If no feasible insertion is possible, the request remains unserved in the list and will be tested again in a new route. When no request can be inserted, a new route is initiated. This process is repeated until all reserved requests are served. The selective requests are treated in the same way but taking Policy 2 or 3 into account.

Algorithm 2 – Sequential insertion heuristic – *SIH*(*S*).

```
sort reserved requests in decreasing order of prices in a list L
set current route index r to 0
mark all reserved requests as unserved
repeat
  r ← r + 1
  initialize the new route using the 1st unserved request of L
  for all unserved request i in L do
    if feasible insertions exist in route r for i then
      insert i in the most profitable position
      mark i as served
    end if
  end for
until all reserved requests are served or r = m (maximum fleet capacity reached)
if Policy ≠ 1 then
  repeat steps 1–13 but for the selective requests and follow the Policy 2 or 3
end if
```

Table 1
Adaptive adjustment of the operator scores.

Increment	Conditions on the solution obtained by the operators
ϱ_1	A new best solution is obtained
ϱ_2	The solution was not found before and improves the incumbent solution
ϱ_3	The solution is not a new one but improves the incumbent solution
ϱ_4	The solution is worse than the incumbent solution but is accepted by the SA scheme

4.2. Adaptive selection of destroy/repair operators

At each iteration, our ALNS algorithm employs one or two removal operators to partially destroy the current solution and then repairs it by utilizing one insertion operator. One question is how to select these operators more effectively. Like all ALNS implementations, the algorithm chooses the most suitable combination of operators depending on their past performance. For diversification purposes, poor-performance operators still need to have a low selection probability to be selected during the search. We use a roulette-wheel mechanism. Assuming that n operators are available, each operator is associated with a weight ω_i which reflects its performance during its previous outcomes. Each operator j is randomly selected with probability $\omega_j / \sum_{i=1}^{n} \omega_i$ in the current iteration.

The weight ω_i of each operator i is set to 1 at the beginning of each ALNS run. It remains fixed during the iterations of a segment but it is adjusted at the end of the segment on the basis of a performance score. At the beginning of each segment, all scores are initialized to 0, for the reason that low performance operators can still have a chance to be selected even if they were seldom selected in the previous segment. For feasible moves, the scores are updated by adding either ϱ_1, ϱ_2, ϱ_3 or ϱ_4 according to the four different situations in Table 1.

In practice the score adjustment parameters should be set such that $\varrho_1 > \varrho_2 > \varrho_3 > \varrho_4$. The first situation is highly rewarded since it yields a new best solution. The second and third situations are still interesting because the current solution is improved. We prefer to favor the second condition because finding a solution with new characteristics means that the search is driven to an unexplored area of solution space. To detect a new solutions, the past solutions are stored. To achieve efficient comparison, only the following characteristics of a past solution are stored: total profit, the number of vehicle used and the number of customers served by each vehicle. The pool is kept sorted in increasing profit order and the existence of a solution with a given cost is checked using dichotomic search. The awarding of score ϱ_4 based on the SA acceptance criterion is used to prevent the search from looping on the same operators and also to bring some diversification.

To update the operator weights after each segment *seg*, let $\omega_{i,\,seg}$ be the weight of operator i used in the segment, $\alpha_{i,\,seg}$ the number of times the operator was called, $\beta_{i,\,seg}$ its resulting score, and $\eta \in [0, 1]$ a reaction factor representing how quick the weights react to performance. The weights are adjusted using formula (18). Note that if $\eta = 1$ then the previous weight is completely ignored and the new weight solely depends on the score achieved in the last segment. The other extreme is $\eta = 0$ which preserves the current weight while ignoring the score.

$$\omega_{i,seg+1} = \begin{cases} \omega_{i,seg} & \text{if } \beta_{i,seg} = 0 \\ (1-\eta)\cdot\omega_{i,seg} + \eta\cdot\beta_{i,seg}/\alpha_{i,seg} & \text{otherwise} \end{cases} \tag{18}$$

4.3. Dynamic adjustment of operator behavior (DAOB)

The PDPTWPR is highly complex since it combines the choice of selective requests and routing decisions. Even a minor modification on the current solution might deeply affect final results. To improve final solution quality and to vary the number of served requests, we implemented a technique called *Dynamic Adjustment of Operator Behavior* (DAOB). The basic idea is to modify progressively the behavior of operators over the successive segments of the ALNS. Firstly, we develop one group of removal policies and two groups of insertion policies based on the specific features of the PDPTWPR.

Request Remove Policies (RRP):

1. Both selective requests and reserved requests are removable.
2. Only selective requests are removable.

Request Insertion Priority Policies (RIPP):

1. Selective requests and reserved requests have the same priority.
2. All reserved requests must be served before treating selective requests.

Insertion Threshold of Selective Requests (ITSR):

1. Selective requests are inserted regardless of profitability.
2. Selective requests are inserted only if they are profitable (insertion cost $<$ service payment).

We would refer to the policies marked by '1.' as Code-1 policies, and the policies marked by '2.' as Code-2 policies.

As explained in the next Section describing each operator, each ALNS iteration applies one or two destroy operators to remove a given number of requests, and then one repair operator. The destroy operators can remove any request in Policy RRP1 but only reserved requests in RRP2. While the ALNS for the PDPTW (Ropke & Pisinger, 2006) reinserts all removed requests (since all requests must be served), our repair operators try to insert unserved requests according to the selected insertion policies (RIPP1/RIPP2 and ITSR1/ITSR2), as long as feasible insertions exist. An infeasible solution, *i.e.*, a solution violating certain constraints or with negative profit, can be obtained if some reserved requests cannot be served, due to time windows or lack of vehicle capacity.

Clearly, Code-1 policies allow deep modifications in the incumbent solution and induce long-range moves while the more conservative policies with Code-2 favor the generation of feasible solutions and a faster improvement of the objective function. At the beginning of our ALNS (first segment) it is worthwhile to widen the search by using more frequently Code-1 policies. However, because of Policy RIPP1, infeasible solutions are frequent since selective requests can exhaust vehicle capacity and leave some reserved

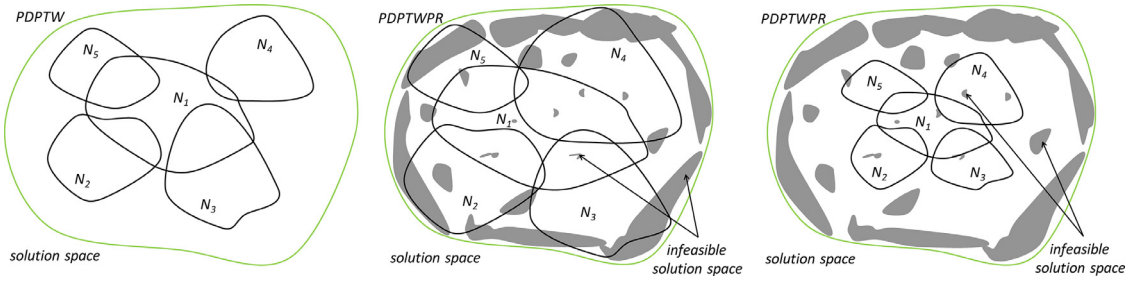

Fig. 1. Illustration of neighborhoods in a classical ALNS and in our DAOB version.

requests unserved. Also, Policy ITSR1 results in very slow improvements of the objective function.

Hence, the probability of using Code-2 policies is gradually augmented at the beginning of each segment. In the last segments, the ALNS generates more feasible solutions and tends to faster improve the total profit. Let P_1 and P_2 denote the probabilities of using policies with Code-1 or Code-2, respectively. They are simply computed as $P_1 = 1 - P_2$ and $P_2 = seg/nsegs$ (where seg is the current segment number and $nsegs$ the total number of segments, $nsegs > 2$).

To better understand the DAOB, the change of neighborhoods $N_1, N_2, \ldots$ for three cases is depicted in Fig. 1. The first one illustrates a traditional ALNS applied to the PDPTW: most of the solution space is searched. The second one corresponds to the first segment of our DAOB for the PDPTWPR. The destroy/repair operators define wider neighborhoods which include many infeasible solutions (shown in grey). The last situation corresponds to the more conservative operators in the last segments: neighborhood are reduced and focus on feasible solutions.

4.4. Description of destroy/repair operators

Although some operators in our ALNS are similar to the ones designed in Ropke and Pisinger (2006) for the PDPTW, they must be adapted to serve a variable number of selective requests and deal with profits. For non-selective VRPs, all requests must be served and one destroy operator removes a certain number of requests which must be reinserted by one repair operator. Conversely, in the PDPTWPR, partial solutions are already feasible as long as all reserved requests are served. Moreover, the behavior of our destroy and repair operators (different treatment for reserved and selective requests) is affected by the DAOB policies explained in the previous section. Finally, the repair operators perform as many feasible request insertions as possible.

We implemented also a *meta-destroy operators* to diversify the search after a maximum number δ of successive iterations without improving the best solution of the current run. This mechanism is independent from the decomposition of the search into segments: The number of iterations without improvement is counted from the beginning of each ALNS run and the maximum number is checked at each iteration of each segment. The meta-destroy consists in applying two destroy operators instead of one.

The number of routes, or vehicles actually used, $\overline{m}$ can be modified by our operators. When one destroy operator removes the only request from a route, this route is closed. When looking for a best insertion for a request, the repair operators consider the $\overline{m}$ non-empty routes plus, if $\overline{m} < m$, one "empty" route reduced to the two depot nodes 0 and $2n + 1$.

4.4.1. Destroy operators

Our ALNS involves six destroy operators described. Given the number $\overline{n}$ of the requests in the incumbent solution and a removal fraction $\rho \in [0, 1]$, each of the operators applies a strategy to select

Algorithm 3 – Least profit removal.

1: $removed \leftarrow 0$
2: **repeat**
3: sort all requests (Policy RPP1) or all selective requests (Policy RPP2) served in S in increasing order of profit in an array L
4: compute a random request index $j = \lceil (r^{U(0,1)})^{100 \cdot \rho} |L| \rceil$
5: remove request L_j from S
6: $removed \leftarrow removed + 1$
7: **until** $removed = \lceil \rho \cdot \overline{n} \rceil$

$\lceil \rho \cdot \overline{n} \rceil$ requests (among all requests in Policy RPP1, or only among selective requests in RPP2). These requests are then removed from the routes. Only one destroy operator is executed in each ALNS iteration, except in the meta-destroy scheme (applied after δ successive non-improving iterations) where two destroy operators are applied to bring diversification.

Random removal. This operator randomly selects the $\lceil \rho \cdot \overline{n} \rceil$ requests to be removed. Depending on the value of ρ, it may significantly modify the incumbent solution.

Least profit removal. The profit of a request i served in the incumbent solution S is defined as $f(S) - f(S')$, where $f(S')$ is the objective function without request i. The least profit removal sketched in Algorithm 3 removes $\lceil \rho \cdot \overline{n} \rceil$ requests with low profits, because they might often be reinserted in more profitable positions. The operator is randomized to avoid repeatedly removing the $\lceil \rho \cdot \overline{n} \rceil$ requests with lowest profits.

Least paid removal. The least paid removal operator has the same algorithm as the least profit removal, except that array L is now sorted in increasing order of prices p_i. Considering request prices is essential in the PDPTWPR: removing the cheapest requests and putting them in the request pool (case of selective requests) or reinserting them in other positions may lead to a better solution.

Most expensive removal. Given a request i served in the incumbent solution S, we define its cost as $f(S) - f(S')$, where $f(S) - f(S')$ represents the difference of transportation cost with or without request i. This operator is widespread in ALNS metaheuristics for general VRPs. It works like the least profit and least paid removal (Algorithm 3), except that array L is sorted in decreasing order of requests' cost.

Shaw removal. We use the same way of implementing Shaw removal (Shaw, 1998) as Ropke and Pisinger (2006). Firstly, a seed request is chosen randomly and the heuristic removes similar requests in terms of distance (a request whose pickup and delivery nodes are close to those of the seed request is favored), time (starts of service at the two nodes are similar in the two requests), and demands. It is also applied $\lceil \rho \cdot \overline{n} \rceil$ times with the same randomization as in Algorithm 3. The underlying idea is that similar requests less frequently violate capacity and time window constraints when they are reshuffled around in groups.

Algorithm 4 – Price similarity removal.

1: randomly select one seed-request r from solution S and put it in a set Z
2: **while** $|Z| < \lceil \rho \cdot \overline{n} \rceil$ **do**
3: sort requests of $S \backslash Z$ such as $i < j \Rightarrow P(r,i) < P(r,j)$ in a list L
4: compute a random request index $j = \lceil (r^{U(0,1)})^{100 \cdot \rho} |L| \rceil$
5: $Z = Z \cup \{L_j\}$
6: **end while**
7: remove all requests of Z from S

Price similarity removal. This operator is similar to the previous one but it removes requests which are similar in terms of price. Then a repair operator will exchange their locations or directly abandon them to increase total profit. We use in fact a dissimilarity measure for two requests i and j, defined as their price difference $P(i,j) = |p_i - p_j|$: growing values correspond to more and more dissimilar prices. This operator is outlined in Algorithm 4.

4.4.2. Repair operators

Two repair operators were utilized in our ALNS. Their behavior depends on the policies RIPP1/2 and ITSR1/2 selected by the DAOB. They insert unserved requests (not only those removed by the destroy operators) as long as feasible insertions are possible.

Basic greedy insertion heuristic. This greedy heuristic inserts one by one unserved requests. The two nodes i and $n+i$ of request i are inserted in order to achieve the largest increase in total profit.

Regret insertion heuristic. The basic greedy heuristic seems quite myopic as it only considers the profit change of one request: the later an attempt of a request insertion is made, the more difficult it is to insert this request at a good insertion position (slot), because the insertion of other requests reduced the number of possible insertion slots. The regret insertion heuristic tries to anticipate by computing for each unserved request a regret equal to the total profit difference between the best insertion and the second best one. Thus, requests with a high regret will be inserted first. The regrets must be recomputed after each insertion, because some insertion positions are no longer available.

4.5. Diversification via simulated annealing

The simulated annealing (SA) scheme appears clearly in the main algorithm. Its goal is to avoid to be trapped in a local optimum. Compared with a descent heuristic which only accepts improved solutions, the SA accepts a degrading move $S \rightarrow S'$ (when $f(S') < f(S)$) with a probability $e^{-(f(S)-f(S'))/T}$. The probability decreases with the profit disparity and with parameter T called temperature.

At the beginning of each run, T is set to T_{beg}. A number computed to accept a solution 30% worse than the initial solution with a given probability τ. The temperature is reduced after each iteration (over successive segments) by multiplying T by a cooling factor $\theta \in (0,1)$. In practice, θ must be close to one to achieve a slow cooling.

4.6. Local search procedure

We observed that embedding a local search procedure in our ALNS is beneficial to improve the outcome of metaheuristic. Consequently, we decided to implement six moves in a local search procedure called only at the end of each segment to keep running time at a reasonable level.

The moves are inspired by the ones of the classical VRP literature (Toth & Vigo, 2014) with the difference that we relocate or exchange pairs of nodes. The moves are illustrated in Figs. 2–7. Circles represent pickup nodes while triangles denote delivery nodes. Reserved requests are filled in black while selective requests have an empty interior. The classical *2-Opt* move is not included because time windows make it infeasible in most cases.

The local search works as follows. Three types of moves are randomly selected, including at least one of the two selective moves (selective request removal and selective request insertion) for the reason that they are more effective than the other four in most cases. The neighborhoods defined by the three types are searched in the order of description below. All feasible moves in the incumbent neighborhood are tested. If improving moves are found, the best one is executed and the neighborhood is examined again, otherwise the search proceeds with the next neighborhood type. The local search stops when the last type yields no improvement.

- *Intraroute relocate:* one pickup node or a delivery node is removed to be reinserted in another position of the same route (Fig. 2).
- *Interroute relocate:* one request is removed from one route and reinserted in another (Fig. 3).
- *Intraroute exchange:* two requests are exchanged in the same route (Fig. 4).
- *Interroute exchange:* two requests are exchanged between two routes (Fig. 5).
- *Selective request removal:* one selective request is removed from its route and becomes unserved (Fig. 6).
- *Selective request insertion:* one unserved selective request is inserted in a route (Fig. 7).

5. Computational results

To evaluate the performance of our ALNS algorithm, we designed 54 the PDPTWPR instances partitioned in small size ($n \in \{10, 20\}$), medium size ($n \in \{30, 40, 50\}$) and large size instances ($n = 100$). The ALNS has been compared with the CPLEX MILP solver (version 12.6).

The following sections describe the generation of instances, list the parameter values used in our algorithm, and provide test results and optimality gaps which are reported separately for small, medium and large size instances. An analysis on the percentage of infeasible solutions and the impact of the *Dynamic Adjustment of Operator Behavior* (DAOB) close the section.

5.1. Generation of instances

The instances of this study are generated based on the Euclidean benchmark instances proposed by Ropke and Pisinger (2006) for the PDPTW, available at URL http://www.diku.dk/~sropke/. We copy the instances with the coordinates of each node, the demand and time windows of each request.

Each instance name has a format $n - |R_r| - |R_s| - source$. Consider instance 10-5-5-50a as an example. There are 10 requests in total, including 5 reserved requests (1–5) and 5 selective requests (6–10). The code 50a means this instance is derived from the original one, prob50a: Only the 10 first requests appearing in prob50a are copied in instance 10-5-5-50a.

Some specific data required for the PDPTWPR are added. For each instance size (10, 20, 30, 40, 50 or 100 requests), we build nine instances using nine PDPTW files. These nine instances can be decomposed in three types: three with roughly one-third of reserved requests, three with 50% of reserved requests, and three with two-thirds of reserved requests.

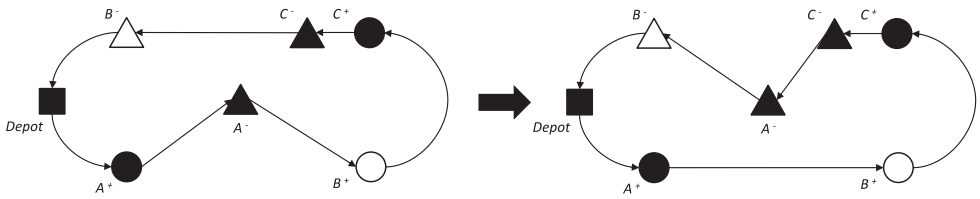

Fig. 2. Intraroute relocate: delivery node of request A is relocated.

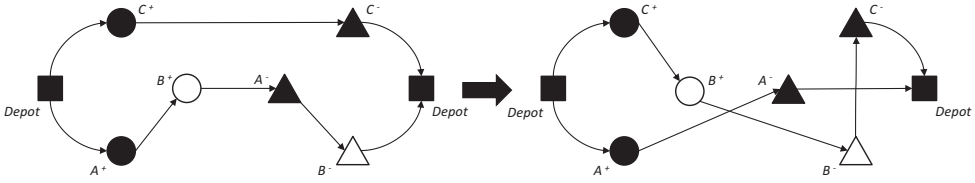

Fig. 3. Interroute relocate: request B is relocated.

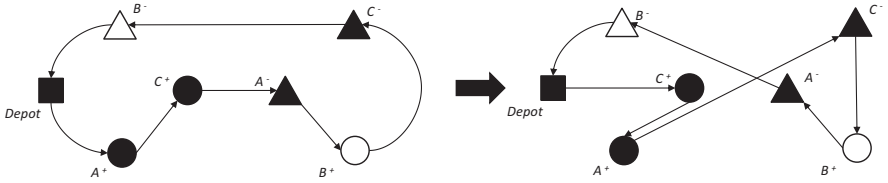

Fig. 4. Intraroute exchange: request A and request C are exchanged.

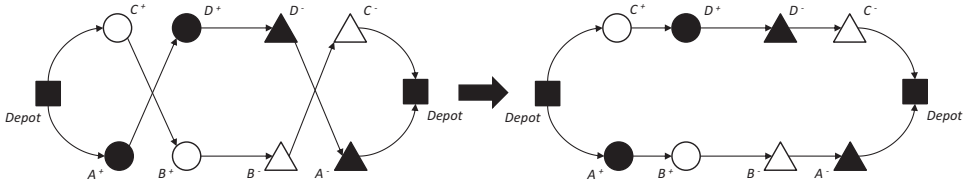

Fig. 5. Interroute exchange: request B and request D are exchanged.

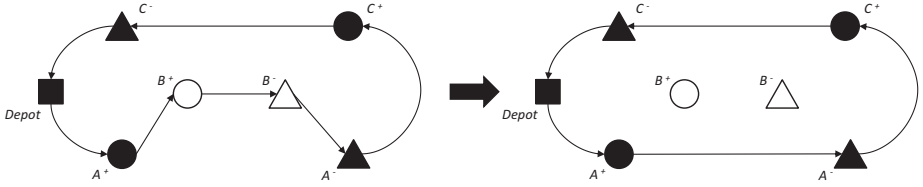

Fig. 6. Selective requests removal: request B is removed.

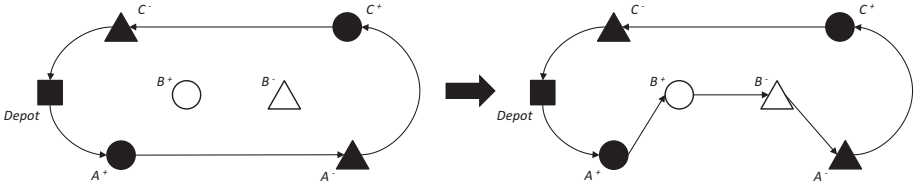

Fig. 7. Selective requests insertion: request B is inserted.

The fleet size of original instances is regarded as a reference. It is adjusted in accordance with the proportion of the number of requests extracted from the original instance.

The service price of each request is set according to the coordinates of its nodes. Take request i as an example, let $d_{i,n+i}$ denote the distance from its pickup node i to delivery node $n+i$, then this request is given a service price $p_i = d_{i,n+i} \cdot \lambda$, $\lambda \in [3, 5]$. This formula generates a large proportion of profitable requests and a small proportion of non-profitable requests.

Finally, since some original instances have multiple depots, we select in such cases one of the depots arbitrarily.

Table 2
Parameter tuning according to instance size.

Symbol	Role	Small	Medium	Large
nruns	Number of runs	10	10	10
nsegs	Segments per run	10,000	20,000	50,000
niters	Iterations per segment	100	100	100
γ_1	SIH Policy 1 probability	0.15	0.15	0.20
γ_2	SIH Policy 2 probability	0.70	0.70	0.60
γ_3	SIH Policy 3 probability	0.15	0.15	0.20
ϱ_1	Operator score increment case 1	10	10	10
ϱ_2	Operator score increment case 2	5	5	5
ϱ_3	Operator score increment case 3	3	3	3
ϱ_4	Operator score increment case 4	1	1	1
η	Score reaction factor	0.8	0.8	0.8
δ	Unfruitful iterations for meta-destroy	100	150	200
ρ	Removal fraction	0.2	0.2	0.3
τ	To set SA initial temperature	0.3	0.4	0.4
θ	SA cooling factor	0.9995	0.9996	0.9997

5.2. Parameter setting

5.2.1. Computational environment and CPLEX setting

All experiments were conducted on a desktop equipped with Intel(R) Core (TM) i7-2600 3.40 gigahertz processor and 8 Gigabyte RAM. The operating system of this PC is 64-bit Window 7. The ALNS algorithm was coded in C++ using the development platform Visual Studio 2013. The Optimization Programming Language (OPL) and the CPLEX solver 12.6 were used to solve the MILP model. CPLEX 12.6 was called with the following option settings: $nodefileind = 2$, $workmem = 128$, $memoryemphasis = 1$, $threads = 8$, $nodesel = 2$ and $varsel = 3$ (see CPLEX 12.6 Solver Manual). With these options, the computation load of CPLEX is distributed over the four cores (8 threads) of the Intel(R) Core (TM) i7-2600 processor.

We solved the MILP model with a preset time of 0.5, 1, 1.5, 2, 4 hours (1800, 3600, 5400, 7200, 14,400 seconds) with 10, 20, 30, 40, 50 requests, respectively. For the large size instances with 100 requests we set the time limit to 10 hours. The long preset time aims to ensure that the resolution of the MILP model can obtain at least one feasible solution served as a comparison indicator with our ALNS algorithm, although in most cases it failed to achieve such a goal. To further evaluate the performance of our ALNS algorithm, we present the upper bounds found by CPLEX as well.

5.2.2. Parameter setting for the ALNS

The ALNS parameters were determined in preliminary experiments, since the ALNS algorithm is composed of several procedures and each procedure has its own parameters, parameter setting was tuned by concerning a tradeoff between solution quality and CPU time. The values used are gathered in Table 2.

Observe that large size instances require more balanced probabilities (smaller selecting probability difference between $\{\gamma_1, \gamma_2, \gamma_3\}$) to provide the ALNS with more diverse initial solutions. The number of iterations without improvement δ before calling the meta-destroy mechanism must be increased on medium and large size instances, to give more time to the ALNS to explore its large neighborhoods. For the same reason, ρ, τ and θ are tuned in keeping with the size of instances.

5.3. Test results and optimality gaps

Two key indicators were used to evaluate the performance of our ALNS algorithm:

i) The upper bound produced by the CPLEX solver for the MILP model of a PDPTWPR instance input, which indicates the upper bound of the optimal objective function value (profit).

Table 3
Abbreviation of experiment indicators and definition.

Abbreviation	Definition
UB_{MILP}	The upper bound of the MILP model obtained by CPLEX in a preset running time
LB_{MILP}	The best feasible objective value found by CPLEX solver in a preset running time
LB_{ALNS}	The best feasible objective value obtained by the ALNS after a preset number of iterations
Gap_{MILP}	The gap between LB_{MILP} and UB_{MILP}. It is calculated by: $\frac{UB_{MILP}-LB_{MILP}}{UB_{MILP}}$
Gap_{ALNS}	The gap between LB_{ALNS} and UB_{MILP}. It is calculated by: $\frac{UB_{MILP}-LB_{ALNS}}{UB_{MILP}}$
$Imp_{ALNS-MILP}$	The improvement of LB_{ALNS} over LB_{MILP}. It is calculated by: $\frac{LB_{ALNS}-LB_{MILP}}{LB_{MILP}}$
CPU_{ALNS}	The computing duration of the ALNS algorithm
CPU_{MILP}	CPU time for solving the MILP model by CPLEX

ii) The best feasible solution of the MILP model found by CPLEX, which is marked as the lower bound of the optimal objective function value (profit).

For ease of reading, the abbreviations of the experiment indicators and corresponding definition are listed in Table 3.

Table 4 compares the performance of our ALNS algorithm and the CPLEX solver on small size instances. For the instances with 10 requests, both solution approaches were able to solve the problem to optimality, but the ALNS algorithm consumed less CPU time than CPLEX. When the number of requests increases to 20, no proven optima were obtained, so we compare the near-optimal solutions of the two methods using the three above-mentioned indicators (Gap_{MILP}, Gap_{ALNS} and $Imp_{ALNS-MILP}$) and the running time of both. Observe that the ALNS algorithm found better objective value than CPLEX for 7 out of 9 instances and the average Gap_{ALNS} is only 6.11%. Furthermore, our ALNS algorithm supersedes CPLEX in terms of running time, the longest CPU time being only 23.2 seconds compared with the limit of 3600 seconds reached by CPLEX.

Table 5 gives in the same format the results for medium size instances. The improvement $Imp_{ALNS-MILP}$ increases quickly with the number of requests. For the group of instances with 30 requests, Gap_{ALNS} is on average 6.17 percent which maintains at the same level of instances with 20 requests, whereas Gap_{MILP} increases from 12.14percent to 24.65 percent. For 40 requests instances, CPLEX failed to identify feasible solutions, in spite of a larger time limit of 2 hours. In contrast, our ALNS algorithm always returned good quality feasible solutions. For $n = 40$ and $n = 50$, the average Gap_{ALNS} is equal to 8.96 percent and 9.98 percent and $Imp_{ALNS-MILP}$ increases to 30.87 percent and 69.77 percent, respectively, excluding the cases for which CPLEX failed to obtain any feasible solution. In parallel, the running time of the ALNS grows naturally with instance size but still represents a small fraction of the CPU time consumed by CPLEX.

Table 6 summarizes the results for 100 requests instances. CPLEX achieved to find feasible solution for only 2 out of 9 instances. In most cases, CPLEX terminated because of lack of memory, so we did not try longer time limit. For this reason, we only report UB_{MILP}, LB_{ALNS} and Gap_{ALNS} for the remaining 7 instances to serve as a benchmark for future comparisons. Our algorithm produce an average 12.80 percent Gap_{ALNS}. Since the best upper bound might be reduced by more sophisticated techniques, the actual optimality gap is possibly less than Gap_{ALNS}.

Table 7 concludes the average value of Gap_{MILP}, Gap_{ALNS}, $Imp_{ALNS-MILP}$, CPU_{MILP} and CPU_{ALNS} by instance size, respectively.

5.4. Impact of the DAOB mechanism

In this section, we will analyze the gain of our proposed DAOB (see Section 4.3) from two perspectives. Firstly we present the

Table 4
Computational results for small size instances.

Instances	UB_{MILP}	LB_{MILP}[a]	LB_{ALNS}[a]	Gap_{MILP}(percent)[b]	Gap_{ALNS}(percent)[b]	$Imp_{ALNS-MILP}$(percent)	CPU_{MILP}(second)	CPU_{ALNS}(second)
10-5-5-50a	965.2	965.2	**965.2**	0	0	0	47.4	4.2
10-5-5-50b	1235.8	1235.8	**1235.8**	0	0	0	256.8	3.1
10-5-5-50c	1415.0	1415.0	**1415**	0	0	0	123.3	3.2
10-3-7-50d	1100.7	1100.7	**1100.7**	0	0	0	134.0	4.1
10-3-7-50e	864.8	864.8	**864.8**	0	0	0	57.0	3.5
10-3-7-50f	1467.6	1467.6	**1467.6**	0	0	0	52.2	3.0
10-7-3-50g	1047.2	1047.2	**1047.2**	0	0	0	147.6	5.2
10-7-3-50h	756.2	756.2	**756.2**	0	0	0	177.6	2.7
10-7-3-50i	1226.4	1226.4	**1226.4**	0	0	0	101.2	4.5
20-10-10-50a	4116.2	3765.1	**3978.9**	8.53	*3.34*	5.68	3600	14.4
20-10-10-50b	3591.8	3123.3	**3475**	13.04	*3.25*	11.26	3600	12.2
20-10-10-50c	1999.7	**1863.6**	1854.7	*6.81*	7.25	−0.48	3600	9.3
20-5-15-50d	3432	3007.4	**3112.7**	12.38	*9.30*	3.52	3600	11.0
20-5-15-50e	3252.4	2766.6	**3018.2**	14.94	*7.2*	9.09	3600	10.3
20-5-15-50f	2555.7	**2365.0**	2334.2	*7.46*	8.67	−1.30	3600	23.2
20-15-5-50g	4086.5	3461.3	**3878.6**	15.30	*5.09*	12.06	3600	14.1
20-15-5-50h	3216.4	2682.0	**2994.4**	16.61	*6.90*	11.65	3600	17.0
20-15-5-50i	4164.7	3574.3	**4000.3**	14.18	*3.95*	11.92	3600	10.3

[a] The higher of the best feasible objective values found by the MILP (Column 3) and the ALNS (column 4) is indicated in boldface.
[b] The better Gap between the upper bounds found by the MILP in a given running time and the best feasible solutions found by the ALNS/the best feasible solutions found by the MILP is emphasized in italics among the columns 5 and 6.

Table 5
Computational results for medium size instances.

Instances	UB_{MILP}	LB_{MILP}[a]	LB_{ALNS}[a]	Gap_{MILP}(percent)[b]	Gap_{ALNS}(percent)[b]	$Imp_{ALNS-MILP}$(percent)	CPU_{MILP}(second)	CPU_{ALNS}(second)
30-15-15-50c	10336.9	**9456.4**	9356.8	*8.52*	9.48	−1.05	5400	35.7
30-10-20-50d	12783.3	7177.7	**11596.6**	43.85	*9.28*	61.56	5400	56.3
30-10-20-50e	11232.5	9632.4	**10763.2**	14.25	*4.18*	11.74	5400	46.0
30-10-20-50f	8564.2	5864.3	**7478**	31.53	*12.68*	27.52	5400	60.5
30-20-10-50g	10648.3	7364.0	**10056.2**	30.84	*5.56*	36.56	5400	41.1
30-20-10-50h	10326.7	8264.7	**10268.2**	19.97	*0.57*	24.24	5400	47.8
30-20-10-50i	8494.3	6002.0	**8278.9**	29.34	*2.54*	37.94	5400	42.0
40-20-20-50a	14527.0	10023.6	**12998.1**	31.00	*10.52*	29.67	7200	72.1
40-20-20-50b	15986.4	9552	**14756.3**	40.25	*7.69*	54.48	7200	102.0
40-20-20-50c	15268.1	11752.1	**13535.5**	23.03	*11.35*	15.18	7200	80.6
40-15-25-50d	12134.6	7531.8	**11136.4**	37.93	*8.23*	47.86	7200	118.0
40-15-25-50e	10134.2	7963.9	**9636.7**	21.42	*4.91*	21.00	7200	75.6
40-15-25-50f	10593.7	–	**9616.0**	–	*9.23*	–	7200	86.4
40-25-15-50g	11667.4	8567.8	**10589.3**	26.57	*9.24*	23.59	7200	89.0
40-25-15-50h	17868.5	–	**16069.9**	–	*10.07*	–	7200	84.3
40-25-15-50i	13244.2	9654.0	**12000.3**	27.11	*9.39*	24.30	7200	100.3
50-25-25-50a	26518.2	14421.2	**24738.6**	45.62	*6.71*	71.54	14,400	361.0
50-25-25-50b	21996.8	–	**18991.0**	–	*13.66*	–	14,400	258.0
50-25-25-50c	23644.1	–	**22695.9**	–	*4.01*	–	14,400	247.5
50-20-30-50d	22414.6	–	**19983.6**	–	*10.85*	–	14,400	577.1
50-20-30-50e	18649.9	10664.8	**16119.2**	42.82	*13.57*	51.14	14,400	246.4
50-20-30-50f	22378.0	–	**20347.5**	–	*9.07*	–	14,400	416.0
50-30-20-55g	19986.5	9894.5	**18465.2**	50.49	*7.61*	86.62	14,400	365.9
50-30-20-50h	23668.4	–	**20004.1**	–	*15.48*	–	14,400	345.0
50-30-20-50i	16986.0	–	**15478.6**	–	*8.87*	–	14,400	466.8

[a] The higher of the best feasible objective values found by the MILP (Column 3) and the ALNS (column 4) is indicated in boldface.
[b] The better Gap between the upper bound found by the MILP in a given running time and the best feasible solutions found by the ALNS/the best feasible solutions found by the MILP is emphasized in italics among the columns 5 and 6.

Table 6
Computational results for large instances test.

Instances	UB_{MILP}	LB_{MILP}[a]	LB_{ALNS}[a]	Gap_{MILP}(percent)[b]	Gap_{ALNS}(percent)[b]	$Imp_{ALNS-MILP}$(percent)	CPU_{MILP}(second)	CPU_{ALNS}(second)
100-50-50-100a	89554.8	–	**74431.9**	–	16.89	–	36,000	741.2
100-50-50-100b	94316.2	54549.7	**85631.4**	42.16	*9.21*	56.98	36,000	766.8
100-50-50-100c	127414.0	–	**111717.1**	–	12.32	–	36,000	515.2
100-25-75-100d	99874.7	–	**86041.3**	–	13.85	–	36,000	1023.0
100-25-75-100e	112084.5	–	**96327.0**	–	14.06	–	36,000	985.1
100-25-75-100f	96683.7	64493.4	**82667.6**	33.29	*14.50*	28.18	36,000	602.0
100-75-25-100g	81324.6	–	**68543.2**	–	15.72	–	36,000	866.9
100-75-25-100h	92333.1	–	**84667.9**	–	8.30	–	36,000	711.4
100-75-25-100i	13269.9	–	**11898.0**	–	10.34	–	36,000	1176.2

[a] The higher of the best feasible objective values found by the MILP (Column 3) and the ALNS (column 4) is indicated in boldface.
[b] The better Gap between the upper bound found by the MILP in a given running time and the best feasible solutions found by the ALNS/the best feasible solutions found by the MILP is emphasized in italics among the columns 5 and 6.

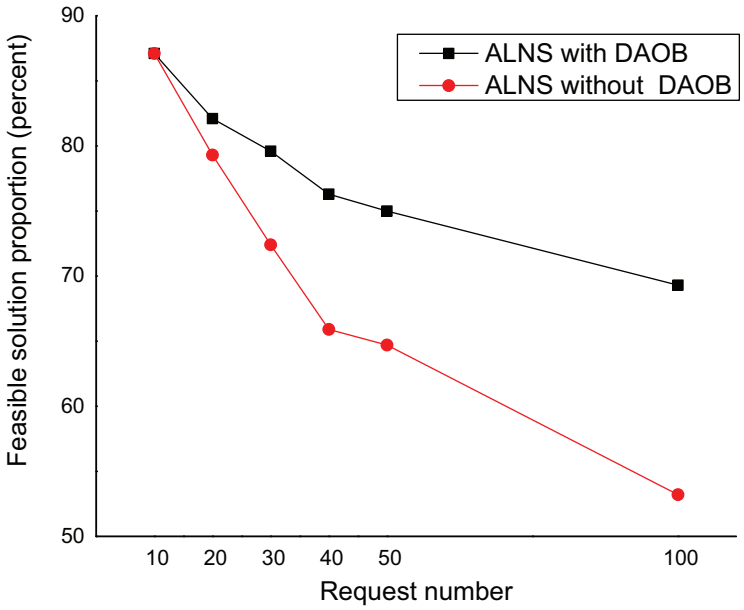

Fig. 8. Proportion of feasible solutions generated with or without DAOB.

Table 7
Statistical average value on the performance indicators,

Instance size	Gap_{MILP} (percent)	Gap_{ALNS} (percent)	$Imp_{ALNS-MILP}$ (percent)	CPU_{MILP} (second)	CPU_{ALNS} (second)
$n = 10$	0	0	0	121.9	3.72
$n = 20$	12.14	6.11	7.04	3600	13.53
$n = 30$	24.65	6.17	26.68	5400	44.61
$n = 40$	29.62	8.96	30.87	7200	89.81
$n = 50$	46.31	9.98	69.77	14,400	318.19
$n = 100$	37.73	12.80	42.58	36,000	820.87

Table 8
Average improvement of total profit with the DAOB.

Number of requests	Average improvement of total profit using the DAOB (percent)
10	0
20	5.7
30	7.0
40	10.3
50	9.8
100	14.6

proportion of feasible solutions as a function of instance size, with or without DAOB, in Fig. 8. The proportion of feasible solutions generated by our ALNS decreases sharply when the number of requests increases, whether the DAOB is activated or not, but clearly the DAOB technique looks effective since it ends with 70 percent of feasible solutions generated versus 50 percent when it is not activated.

Table 8 shows the average improvement of applying the DAOB for different size of instances. Here again, the DAOB leads to a better total profit (improve objective function value) on average.

6. Conclusions

This article introduces a new vehicle routing problem, the Pickup and Delivery Problem with Time Windows, Profits, and Reserved Requests (PDPTWPR). The PDPTWPR is an important subproblem of Collaboration Logistics (CL) in Less than Truck-Load transportation (LTL) mode. To get a near optimal solution of the PDPTWPR under tight time window and fleet size constraints, we developed specific techniques to improve the basic adaptive large neighborhood search (ALNS) method, such as the meta-destroy mechanism, the search organized in segments and the dynamic adjustment of operator behavior (DAOB). Moreover, 8 tailored destroy/repair operators are designed to cope with the peculiarities of the PDPTWPR and a local search procedure based on 6 effective moves is added for further improvement. To the best of our knowledge, this is the first time an ALNS is developed for a pickup and delivery problem with profits.

To evaluate the performance of our ALNS heuristic applied to the PDPTWPR, a Mixed-Integer Linear Programming (MILP) model is formulated and solved by CPLEX in a pre-specified time limit. For small to medium size instances (up to 50 requests), the upper and lower bounds achieved by CPLEX are compared with the lower bounds obtained by the ALNS. The test results show that our heuristic is able to retrieve the proven optima found by CPLEX. In the cases without proven optima, the ALNS significantly outperforms CPLEX both in terms of solution quality and CPU time. On the largest instances with 100 requests, even when CPLEX was not able to reach feasible solutions in 10 hours, the ALNS was still able to generate good feasible solutions in a reasonable computational time.

Our future work will consider a variant of the PDPTWPR that has new characteristics such as a heterogeneous vehicle fleet, maximum tour duration, multiple vehicle depots, etc. The design of a fair post-collaboration profit reallocation scheme will also be addressed.

Acknowledgments

This work is supported by ANR (the French National Research Agency) in the framework of the TCDU project (Collaborative Transportation in Urban Distribution). This project ANR-14-CE22-0017 is labeled by the “Pôle Véhicule du Futur”, and is jointly performed by four partners, the three French Universities of Technology (UTT, UTBM, UTC) and the company Share And Move Solutions (SAMS). The China Scholarship Council (CSC) grant no. 201304490068 is also acknowledged for Yuan Li’s Ph.D. grant.

References

Aksen, D., Kaya, O., Salman, F. S., & Tüncel, Ö. (2014). An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem. *European Journal of Operational Research, 239*(2), 413–426.

Archetti, C., Corberan, A., Sanchis, J., Plana, I., & Speranza, M. (2014). The team orienteering arc routing problem. *Transportation Science, 48*, 442–457.

Archetti, C., Speranza, M., & Vigo, D. (2013). Vehicle routing problems with profits. *Technical Report WPDEM2013/3*. University of Brescia.

Balas, E. (1989). The prize collecting traveling salesman problem. *Networks, 19*(6), 621–636.

Bent, R., & Hentenryck, P. V. (2006). A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research, 33*(4), 875–893.

Buer, T. (2014). An exact and two heuristic strategies for truthful bidding in combinatorial transport auctions. arXiv preprint arxiv:1406.1928.

Butt, S. E., & Cavalier, T. M. (1994). A heuristic for the multiple tour maximum collection problem. *Computers & Operations Research, 21*(1), 101–111.

Chao, I., Golden, B. L., & Wasil, E. A. (1996). The team orienteering problem. *European Journal of Operational Research, 88*(3), 464–474.

Cruijssen, F., Cools, M., & Dullaert, W. (2007). Horizontal cooperation in logistics: opportunities and impediments. *Transportation Research Part E: Logistics and Transportation Review, 43*(2), 129–142.

Dai, B., & Chen, H. (2009a). A benders decomposition approach for collaborative logistics planning with LTL transportation. In *Proceedings of the third international conference on operations and supply chain management, Wuhan, China*.

Dai, B., & Chen, H. (2009b). Mathematical model and solution approach for collaborative logistics in less than truckload (LTL) transportation. In *Proceedings of international conference on computers & industrial engineering (CIE 2009)* (pp. 767–772). IEEE.

Dai, B., & Chen, H. (2011). A multi-agent and auction-based framework and approach for carrier collaboration. *Logistics Research, 3*(2–3), 101–120.

Dai, B., & Chen, H. (2012). Profit allocation mechanisms for carrier collaboration in pickup and delivery service. *Computers & Industrial Engineering, 62*(2), 633–643.

Dai, B., & Chen, H. (2015). Proportional egalitarian core solution for profit allocation games with an application to collaborative transportation planning. *European Journal of Industrial Engineering, 9*(1), 53–76.

Dai, B., Chen, H., & Yang, G. (2014). Price-setting based combinatorial auction approach for carrier collaboration with pickup and delivery requests. *Operational Research, 14*(3), 361–386.

Ergun, Ö., Kuyzu, G., & Savelsbergh, M. (2007). Shipper collaboration. *Computers & Operations Research, 34*(6), 1551–1560.

Esper, T. L., & Williams, L. R. (2003). The value of collaborative transportation management (CTM): its relationship to CPFR and information technology. *Transportation Journal, 42*(4), 55–65.

Feillet, D., Dejax, P., & Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation Science, 39*(2), 188–205.

Frieze, A. M., Galbiati, G., & Maffioli, F. (1982). On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks, 12*(1), 23–39.

Goemans, M. X., & Bertsimas, D. J. (1990). On the parsimonious property of connectivity problems. In *Proceedings of the first annual ACM-SIAM symposium on discrete algorithms* (pp. 388–396). Society for Industrial and Applied Mathematics.

Hernández, S., Peeta, S., & Kalafatas, G. (2011). A less-than-truckload carrier collaboration planning problem under dynamic capacities. *Transportation Research Part E: Logistics and Transportation Review, 47*(6), 933–946.

Hosny, M. I., & Mumford, C. L. (2012). Constructing initial solutions for the multiple vehicle pickup and delivery problem with time windows. *Journal of King Saud University – Computer and Information Sciences, 24*(1), 59–69.

Krajewska, M. A., & Kopfer, H. (2009). Transportation planning in freight forwarding companies: Tabu search algorithm for the integrated operational transportation planning problem. *European Journal of Operational Research, 197*(2), 741–751.

Labadie, N., Mansini, R., Melechovsky, J., & Wolfler-Calvo, R. (2012). The team orienteering problem with time windows: an LP-based granular variable neighborhood search. *European Journal of Operational Research, 220*, 15–27.

Lee, C.-G., Kwon, R. H., & Ma, Z. (2007). A carriers optimal bid generation problem in combinatorial auctions for transportation procurement. *Transportation Research Part E: Logistics and Transportation Review, 43*(2), 173–191.

Li, H., & Lim, A. (2003). A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools, 12*(02), 173–186.

Nanry, W. P., & Wesley Barnes, J. (2000). Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological, 34*(2), 107–121.

Nguyen, V. H., & Nguyen, T. T. T. (2010). Approximating the asymmetric profitable tour. *Electronic Notes in Discrete Mathematics, 36*, 907–914.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research, 34*(8), 2403–2435.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science, 40*(4), 455–472.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and practice of constraint programming* (pp. 417–431). Springer.

Toth, P., & Vigo, D. (2014). *Vehicle routing: problems, methods and applications* (2nd ed.). Philadelphia: SIAM.

Triki, C., Oprea, S., Beraldi, P., & Crainic, T. G. (2014). The stochastic bid generation problem in combinatorial transportation auctions. *European Journal of Operational Research, 236*(3), 991–999.

Verdonck, L., Caris, A., Ramaekers, K., & Janssens, G. K. (2013). Collaborative logistics from the perspective of road transportation companies. *Transport Reviews, 33*(6), 700–719.

Wang, X., Kopfer, H., & Gendreau, M. (2014). Operational transportation planning of freight forwarding companies in horizontal coalitions. *European Journal of Operational Research, 237*(3), 1133–1141.

Wang, X., & Xia, M. (2005). Combinatorial bid generation problem for transportation service procurement. *Transportation Research Record: Journal of the Transportation Research Board, 1923*, 189–198.