Computers and Operations Research 88 (2017) 1–14

Contents lists available at ScienceDirect

Computers and Operations Research

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

An adaptive large neighborhood search for the full truckload pickup and delivery problem with resource synchronization

Axel Grimault[a], Nathalie Bostel[b], Fabien Lehuédé[a,*]

[a] IMT Atlantique, LS2N CNRS, Nantes, France
[b] Université de Nantes, LS2N CNRS, Nantes, France

ARTICLE INFO

Article history:
Received 3 July 2015
Revised 26 September 2016
Accepted 16 June 2017
Available online 17 June 2017

Keywords:
Vehicle routing
Full truckload
Adaptive large neighborhood search
Synchronization

ABSTRACT

This paper presents the Full Truckload Pickup and Delivery Problem with Resource Synchronization (FT-PDP-RS). It consists of optimizing the transport of materials between sites, using a heterogeneous fleet of trucks, in the context of public works. Full truckload pickup and delivery requests have to be served within time windows. Trucks are synchronized on pickup or delivery locations based on unitary loading and unloading resources. We propose an Adaptive Large Neighborhood Search (ALNS) to solve this problem. Custom destroy and repair operators and an efficient feasibility insertion procedure have been designed to solve it. The method has been evaluated on real instances from the literature and on real case instances from a public works company. Computational experiments confirm the efficiency of the method.

1. Introduction

This paper presents a truck routing and scheduling problem arising in the public works sector. The distribution of materials in this sector involves a large number of distances travelled each year. The problem is studied in partnership with Luc Durand,[1] a medium size company, whose travelled distance is estimated as ten millions kilometers per year, among which 30% are empty truck trips. This suggests that transportation may be a source of significant cost savings for this company.

We consider an operational problem for the daily transportation of materials. It consists in optimizing the transportation of materials between several different sites, using a heterogeneous fleet of vehicles. Transportation supports the construction of road networks, levelling works and deconstruction operations, supplying sites with materials and handling wastes. Fig. 1 presents three layers of different materials that are transported in our context. Construction sites, in particular, need deliveries of materials from quarries or asphalt concrete plants. Waste has to be transported from construction sites to waste lands or waste recycling plants. Transportation demands are significantly larger than the capacity of a truck and mixing two different products in a truck is generally not possible. As a result, all transportations can be seen as full truckloads and a demand generates several transportation requests.

In a first step, presented in Grimault et al. (2014), demands are split into requests according to truck capacities and characteristics. This step also determines a set of compatible trucks for each request. Time windows on pickup and deliveries are determined integrating that, for loading and/or unloading operations, vehicles may share a common resource (excavator, paver, weigh-bridge). A resource is assigned to a site and stays on this site all along a working day. When two vehicles compete to access a resource, one has to wait for it to become available. Hence, resources may generate waiting times. We distinguish two kinds of deterministic demands: *Scheduled* demands correspond to operations with known start and end times which must be supplied with materials without interruption. *Unscheduled* demands are planned with more flexibility. For both kind of demands, routes have to be synchronized with pickup or delivery locations due to loading or unloading resource availabilities (Drexl, 2012). Fig. 2 illustrates the result of the request splitting step for these two kinds of demands at two delivery locations with resource synchronization constraints. A scheduled demand corresponds to the delivery of asphalt concrete that constitutes the upper layer of the road (see layer 1 in Fig. 1). Discontinuances in application of asphalt result in imperfections on the road. The resource for asphalt concrete demands is a paver that lays the asphalt concrete on the road. It is paired with a truck which unloads progressively during application. Unscheduled demands include supplying a construction site in gravel from a quarry, or removing waste from a land to a recycling waste plant

* Corresponding author.
E-mail addresses: axel.grimault@imt-atlantique.fr (A. Grimault), nathalie.bostel@univ-nantes.fr (N. Bostel), fabien.lehuede@imt-atlantique.fr (F. Lehuédé).

[1] http://www.durandtp.fr

http://dx.doi.org/10.1016/j.cor.2017.06.012
0305-0548/

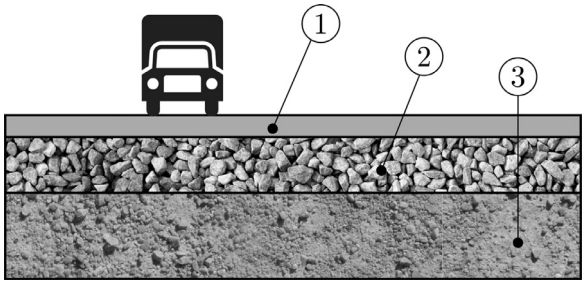

Fig. 1. The construction of a route is divided in three kinds of materials which are supplied separately: the asphalt layer (1) is a *scheduled* demand while the gravel layer (2) and the soil layer (3) are *unscheduled* demands.

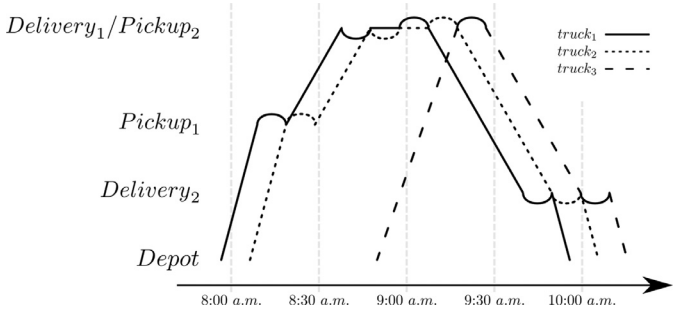

Fig. 3. Representation of a solution. Three trucks serve two demands: demand 1 and demand 2. The delivery point of demand 1 and the pickup point of demand 2 share the same resource. An arc sloping to the top (respectively to the bottom) represents a collection (resp. a delivery) of material. The continuance of the delivery for demand 2 is seen on the graph with the successive deliveries performed between *9:30 a.m.* and *10:00 a.m.* by the three trucks.

(see layers 2 and 3 in Fig. 1). In this case, trucks are loaded and unloaded by a single loader. The utilization of the resource is also unitary. For these requests, large time windows allow the demand to be served throughout the day.

In this paper we focus on designing a minimum cost set of routes to serve a given set of requests, satisfying capacities, time windows and resource synchronization constraints. We denote this problem as the Full Truckload Pickup and Delivery Problem with Resource Synchronization (FT-PDP-RS).

To solve it, we propose a metaheuristic based on Adaptive Large Neighborhood Search (ALNS). A representation of a solution of the FT-PDP-RS is given in Fig. 3.

This paper is organized as follows. Section 2 reviews related papers from the literature. Section 3 presents the FT-PDP-RS and a mathematical model for this problem. Section 4 presents the ALNS and the destroy and repair operators developed to solve the problem. The insertion feasibility test is introduced in Section 5. The description of instances from the literature and from real cases is presented in Section 6. Results on these instances are presented in Section 7. A conclusion and perspectives are given in Section 8.

2. Literature review

The presented problem can be seen as a variant of the Pickup and Delivery Problem with Time Windows (PDPTW) (Desaulniers et al., 2001; Dumas et al., 1991). Its main additional characteristics are a heterogeneous fleet of vehicles, full truckload transportation (FTL) and resource synchronization.

Mathematical models for a combined scheduling-transportation problem for an asphalt concrete company are presented in Rubasheuskaya (2012). The objective is to supply teams of workers with asphalt in route fixing operations in order to minimize the drivers working time. Truck routes perform multiple deliveries from a single depot. Operation synchronization appears between the routes of teams of workers and trucks. The author proposes different models to handle synchronization between a team of workers and a fleet of heterogeneous trucks for laying the asphalt concrete. To solve this problem, a simplified version with uncapacitated vehicles is considered by Brachner in Brachner (2013). Relatively small instances are solved with a commercial solver and a ruin and recreate heuristic.

Problems related to the delivery of Ready-Mixed Concrete (RMC) are the closest to our application. In particular, Asbach et al. (2009) consider the problem of delivering large orders of RMC from plants to customers with a heterogeneous fleet of vehicles. Several FTL trips are generally needed to deliver an order, which can be supplied within a given time window from several plants of limited capacities. Minimum and maximum time lags between consecutive deliveries at the customer and loading operations at each plant imply synchronizing routes on resources. Multiple deliveries to a customer can be pre-ordered, but the order of loading operations at plants should be determined. The problem is modeled on a very large graph which makes it unsolvable by a

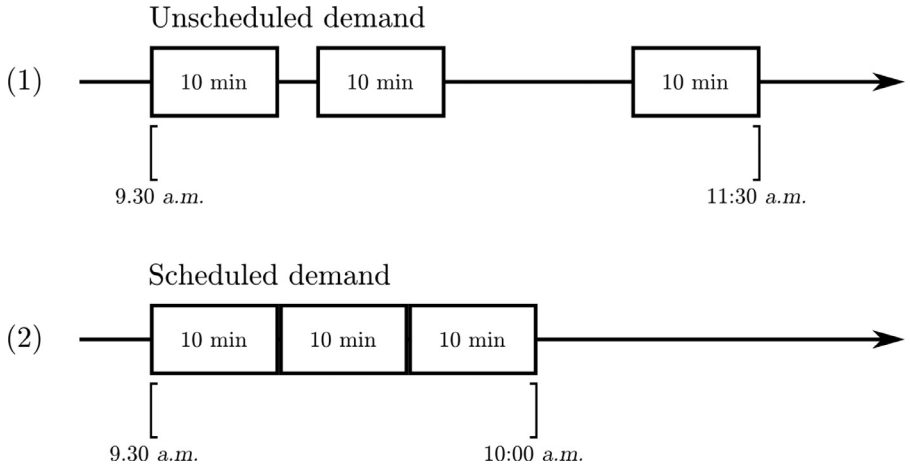

Fig. 2. This figure represents two different demands at their delivery points: an *unscheduled* demand (1) and a *scheduled* demand (2). Demand (1) is split into 3 requests (3 full truckload deliveries) and must be served between a large time windows [9:30;11:30]. As the service takes only 10 min and there is no need for continuity, the 3 deliveries can be placed everywhere between 9:30 and 11:30. Demand (2) is also split into 3 requests and the 3 deliveries must be done without interruption, beginning every 10 min after 9:30. The time windows are adjusted such that no interruption occurs during this operation.

MIP solver. As a result, a local search based algorithm is used to solve real-world instances. A similar problem has been recently discussed in Schmid et al. (2010, 2009). In particular, Schmid et al. (2010), share most characteristics with Asbach et al. (2009) and integrate trucks with specialized instrumentation. Plants are uncapacitated but pre-ordered successive deliveries to each customer have to be scheduled according to time-lags. An initial solution is obtained by a variable neighborhood search (VNS). Then, an algorithm iteratively unfixes some decision variables of the solution with shaking operators. A mixed integer linear program is solved for a subset of unfixed variables. The resulting solution is then improved with different neighborhoods. This process iterates until a run time limit is reached. This method provides good solutions for a real case study of a company. Liu et al. (2014) integrate the routing of pumps and trucks and present a genetic algorithm to solve relatively small instances of the integrated problem. Truck routes are created according to priority rules. Other problems related to the delivery of RMC are presented in the recent literature review provided by Liu et al. (2014).

FTL transportation problems are also met in forestry operation planning with the collection and delivery of round timbers. It implies multiple FTL trips between harvesting sites and customers to satisfy orders. One of the most studied problems is the Log-Truck Scheduling Problem (LTSP) (Palmgren et al., 2004). A simplified version, the Timber Transport Vehicle Routing Problem (TTVRP) (Gronalt and Hirsch, 2007; Hirsch, 2011), considers that the pickup location and quantities for customer orders are given. Flisberg et al. (2009) propose a two phase approach for the LTSP. A first phase creates transport requests using a heterogeneous fleet of trucks. The second phase uses an extended version of the Unified Tabu Search (UTS) method (Cordeau et al., 2001) to design truck routes. A variant of the UTS is proposed by Hirsch (2011) to solve the TTVRP. El Hachemi et al. (2014, 2013) consider a synchronized version of the LTSP where a single log-loader ensures the loading and unloading of all trucks at each pickup or delivery location. Hence, two operations at a given location cannot overlap in time. The authors propose two methods based on constraint programming to solve a problem with synchronization between log loaders and trucks. The method is applied on two industrial instances with 400 requests and 700 requests per week to be transported between forests and woodmills. Bredström and Rönnqvist (2008) present a mathematical model with precedence and synchronization constraints for the simultaneous routing of loaders and trucks, as well as other applications in home-care staff scheduling and airline scheduling. A heuristic based on the solving of the linear relaxation of the model as well as restricted parts of its mixed integer version is applied to real home-care staff scheduling instances.

Other FTL transportation problems generally involve a homogeneous fleet of vehicles. Desrosiers et al. present a vehicle routing problem with full loads and a homogeneous fleet of vehicles in Desrosiers et al. (1988). When several loads are required to serve a demand between two points, the demand is split into a number of arcs between the same points. The problem is solved by a branch and bound algorithm. Arunapuram et al. define the Vehicle Routing Problem with Full Truckloads (VRPFL) in Arunapuram et al. (2003) and propose a branch and price algorithm to solve it. This problem is similar to a Multi Depot Vehicle and Scheduling Problem with Time Windows (MDVSPTW) but in the VRPFL, a demand can be served by many full truckloads. Since all trucks are equivalent, minimizing the cost is equivalent to minimizing empty travels. Explicit formulation of the Full Truckload Pickup and Delivery Problem with Time Windows (FTPDPTW) has been firstly presented by Gronalt et al. (2003). Four savings heuristics are proposed to solve the problem. Heuristics are compared over randomly generated problems with up to 512 demands. Currie et al. propose a Tabu Search (TS) to solve the delivery of materials in road construction in Currie and Salhi (2004). Deliveries are seen as full truckloads with a heterogeneous fleet of vehicles. Allocations of demands to vehicles are predetermined by the company. Full truckload problems are close to applications in intermodal container transportations. The goal is to transport containers between shippers and terminals. Intermodal transport of containers is modeled as a FTPDPTW in Caris and Janssens (2009). In this paper, a two phase insertion heuristic constructs an initial solution. This solution is improved in a local search descent with three neighborhoods. To minimize empty trucks movements in a full truckloads problem in carrier collaboration, Liu et al. (2010) propose a two phase greedy algorithm. The first phase constructs cycles of deliveries and the second phase generates vehicle routes among these cycles.

An originality of the proposed problem relies on the synchronization between routes on resources. Synchronization in vehicle routing has been recently surveyed by Drexl (2012). A classification of five synchronization constraints (task synchronization, operation synchronization, movement synchronization, load synchronization and resource synchronization) underlines the recent interest on this field. Our problem involves *resource synchronization* which is defined as:

> *"At any point in time, the total utilization or consumption of a specified resource by all vehicles must be less than or equal to a specified limit."*

This limit is one in our case. Synchronization on resources in VRP are the focus of few recent studies. The principle was first introduced by Hempsch and Irnich (2008) under the name *"inter-tour resource constraint"*. The authors present a generic model for the VRP with inter-tour constraints based on a giant-tour representation. The problem is solved using local search. The routing of Automated Guided Vehicles (AGVs) with multiple resource constraints is introduced in Ebben et al. (2005). Several heuristics, based on scheduling rules, are used to solve a real case study from an underground transportation system. As pointed out before, resource synchronization constraints have also been considered in log truck scheduling (El Hachemi et al., 2014, 2013) or in long-haul freight transportation (Neves-Moreira et al., 2016).

3. Formulation of the FT-PDP-RS

We define the FT-PDP-RS on a complete oriented graph $G = (V, A)$ where V is the set of vertices and A the set of arcs.

We consider a set R of n FTL requests to be served by a fleet K of heterogeneous trucks. Each request $r \in R$ is represented by a pickup point $p_r \in P$, a delivery point $d_r \in D$ and a set of compatible trucks K_r. The service time (pickup or delivery) at a point $i \in P \cup D$ must start within a time window $[e_i, l_i]$. The duration of the service at a point $i \in P \cup D$ is denoted by s_i. A truck $k \in K$ starts from its depot $o(k) \in O_1$ and terminates at depot $o'(k) \in O_2$. The set of depots is denoted by $O = O_1 \cup O_2$. Each depot $o \in O$ is open in the time window $[e_o, l_o]$. A truck is fully loaded at a collection point and fully unloaded at the corresponding delivery point. The cost of using vehicle $k \in K$ is denoted by $\mathcal{CD}^k$ which includes the equipment cost and the structural cost (i.e. human resource cost). Traveling times and costs depend on the type of vehicle and the characteristics of the route (length, speed limitation, declivity). The travel time between two locations $(i, j) \in A$ by a truck k is denoted by t_{ij}^k and the distance between these locations by d_{ij}^k. The kilometric cost of using truck k is denoted by $\mathcal{CK}^k$ and the hourly cost by $\mathcal{CH}^k$.

The set of resources is denoted by Π and the set of operations that belong to a common resource $\pi \in \Pi$ is denoted by $V_\pi \subset P \cup D$. Only one operation can be performed at a given time by a given resource. A truck has to wait at a point if it arrives before the

Table 1
Sets for the FT-PDP-RS.

Set	Description
V	Set of vertices $V = O \cup P \cup D$
O	Sets of departure and arrival depots $O = O_1 \cup O_2$
A	Set of arcs
K	Set of trucks
R	Set of transportation requests with $\|R\| = n$
K_r	Set of compatible trucks for request $r \in R$
Π	Set of resources
V_π	Set of vertices that share a resource π with $V_\pi \subset P \cup D$

opening of the time window or before the resource at this point is available. The FT-PDP-RS consists of determining a set of minimum cost FTL pickup and delivery routes such that all requests are served with compatible trucks, respecting time windows and resource synchronization constraints, and such that the sum of fixed costs and traveling costs is minimal.

To simplify the presentation, we define $V = O \cup P \cup D$ and $A = O_1 \times P \cup \{(p_r, d_r) | r \in R\} \cup D \times P \cup D \times O_2 \cup \{(o(k), o'(k)) | k \in K\}$. Note that the set of edges A can be refined to integrate truck and request compatibilities. The sets used in the model are summarized in Table 1.

We present a mathematical formulation of the FT-PDP-RS problem based on the PDP formulation of Desaulniers et al. (2001).

We define the binary variable y^k which is equal to one if truck $k \in K$ performs a tour in the solution and zero otherwise. Variable x_{ij}^k is equal to one if truck $k \in K$ travels through arc $(i, j) \in A$ and zero otherwise. The time variable h_i represents the beginning of service at point $i \in V$. The variable z_{ij}^π is equal to one if point $i \in V_\pi$ is scheduled before point $j \in V_\pi$ on resource π.

The mathematic model of the FT-PDP-RS is the following:

$$\min \quad \sum_{k \in K} \mathcal{CD}^k * y^k + \sum_{k \in K} \sum_{(i,j) \in A} (\mathcal{CK}^k * x_{ij}^k * d_{ij}^k + \mathcal{CH}^k * x_{ij}^k * t_{ij}^k) + \sum_{i \in P} s_i * \left(\sum_{k \in K} \sum_{j \in D} \mathcal{CH}^k * x_{ij}^k \right) \tag{1}$$

$s.t.$

$$\sum_{(o(k),i) \in A} x_{o(k)i}^k = 1 \qquad \forall k \in K \tag{2}$$

$$\sum_{(i,o'(k)) \in A} x_{io'(k)}^k = 1 \qquad \forall k \in K \tag{3}$$

$$\sum_{(j,i) \in A} x_{ji}^k = \sum_{(i,j) \in A} x_{ij}^k \qquad \forall k \in K, \forall i \in P \cup D \tag{4}$$

$$\sum_{k \in K_r} x_{p_r d_r}^k = 1 \qquad \forall r \in R \tag{5}$$

$$\sum_{i \in P} x_{o(k)i}^k \leq y^k \qquad \forall k \in K \tag{6}$$

$$h_j + M_{ij}(1 - x_{ij}^k) \geq h_i + s_i + t_{ij}^k \qquad \forall (i, j) \in A, \forall k \in K \tag{7}$$

$$e_i \leq h_i \leq l_i \qquad \forall i \in V \tag{8}$$

$$h_j + M_{ij}(1 - z_{ij}^\pi) \geq h_i + s_i \qquad \forall \pi \in \Pi, \forall i \in V_\pi, \forall j \in V_\pi \tag{9}$$

$$z_{ij}^\pi + z_{ji}^\pi = 1 \qquad \forall \pi \in \Pi, \forall i \in V_\pi, \forall j \in V_\pi \tag{10}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall (i, j) \in A, \forall k \in K$$

$$y_k \in \{0, 1\} \qquad \forall k \in K$$

$$h_i \in \mathbb{R}^+ \qquad \forall i \in V$$

$$z_{ij}^\pi \in \{0, 1\} \qquad \forall \pi \in \Pi, \forall i \in V_\pi, \forall j \in V_\pi$$

The objective function (1) minimizes the sum of traveling costs (kilometric and hourly), the hourly cost of service times and vehicle utilization costs. Constraints (2) and (3) state that each truck starts at its origin depot and ends at its destination depot. Flow conservation is ensured by constraints (4). Constraints (5) impose that all FTL requests are served by compatible trucks. Constraints (6) state that a truck is used as soon as it visits one request. Constraints (7) and (8) ensure the consistency of time variables and the satisfaction of time windows. Constraints (9) and (10) state that it exists an order on each resource between two vertices. Big M value defined is $M_{ij} = \max\{0, l_i + s_i + t_{ij} - e_j\}$ with $t_{ij} = \max_{k \in K} t_{ij}^k$.

The FT-PDP-RS is **NP**-hard since it generalizes the multiple traveling salesman with time windows (Bektas, 2006). The objective of this paper is to develop a fast solving method which provides good solutions. For this reason, we propose a metaheuristic to tackle this problem.

4. An ALNS for the FT-PDP-RS

In this section, we present the solution method used to solve the FT-PDP-RS. ALNS was introduced in Ropke and Pisinger (2006). It extends Large Neighborhood Search (LNS), introduced by Shaw (1998) and Ruin and Recreate, proposed by Schrimpf et al. (2000).

ALNS showed good performances on a large class of classical VRP (Pisinger and Ropke, 2007; Ropke and Pisinger, 2006) with relatively low computational times. It is still one of the best metaheuristics for the PDPTW (Ropke and Pisinger, 2005). ALNS also proved to be very efficient to solve several VRP with synchronization constraints: the Pickup and Delivery Problem with Transfers (PDPT) (Masson et al., 2012; 2013), the Dial-A-Ride Problem with Transfers (DARPT), the Two-Echelon Vehicle Routing Problem (2E-VRP) (Hemmelmayr et al., 2012) and the Two-Echelon Multiple-Trip Vehicle Routing Problem with Satellite Synchronization (2E-MTVRP-SS) (Grangier et al., 2016). A LNS metaheuristic has been proposed in Goel and Meisel (2013) to solve a workforce routing and scheduling problem including a complex optimization criterion and synchronization constraints. ALNS has been used to solve problems with complex objective function such as the Pollution Routing Problem (Demir et al., 2012) or complex loading or scheduling constraints (Bortfeldt et al., 2015; Kovacs et al., 2012).

The general principle of ALNS is described in Algorithm 1.

It is based on successive destruction and reconstruction operations to improve a solution. At each iteration, the number q of requests to remove is chosen randomly in the interval $[\xi_{min}{}^*|R|, \xi_{max}{}^*|R|]$ according to parameters ξ_{min} and ξ_{max}, the minimum and maximum percentage of requests to remove, respectively. A roulette-wheel mechanism selects a destroy operator in set $\mathcal{N}^-$ and a repair operators in set $\mathcal{N}^+$. The adaptive layer is similar to Ropke and Pisinger (2005). It influences the probability of selection of operators according to their ability to generate interesting solutions, which is assessed in procedure `updateOperatorsScores`. The produced solution is accepted as the new current solution according to a simulated annealing criterion.

Algorithm 1: Framework of the ALNS.

input : an initial solution $s_{initial}$, a set of destroy operators $\mathcal{N}^-$, a set of repair operators $\mathcal{N}^+$
output: **return** the best solution s^*

begin
  $s^* \leftarrow s_{initial}$
  $s_{current} \leftarrow s_{initial}$
  while *stopping criterion is not met* **do**
    `/* Select the number of requests q to remove */`
    destroy $\leftarrow$ `chooseDestroyOperator`$(\mathcal{N}^-)$
    repair $\leftarrow$ `chooseRepairOperator`$(\mathcal{N}^+)$
    $s_{new} \leftarrow destroy(s_{current})$
    $s_{new} \leftarrow repair(s_{new})$
    if *transition is accepted* **then**
      $s_{current} \leftarrow s_{new}$
    end
    if $z(s_{new}) < z(s^*)$ **then**
      $s^* \leftarrow s_{new}$
    end
    `updateScoresOperators`$(\mathcal{N}^-, \mathcal{N}^+)$
  end
  return s^*
end

We present the main characteristics of the proposed ALNS in the following sections. The parametrization of our ALNS is discussed in Section 7.2.

4.1. Destroy operators

We present in this part the destroy operators used in the algorithm. A destroy operator removes a number q of requests from the list of inserted requests $\mathcal{L}$ of a solution s and adds them to the request bank $\mathcal{B}$. The destroy operators are the following:

1. **Random Destroy Operator (RDO)**: This operator randomly removes q requests from the solution. The complexity of this operator is $O(q)$.
2. **Worst Destroy Operator (WDO)**: This operator removes the q most expensive requests. We denote by $\Delta(i, s)$ the difference in terms of costs between the solution s and the solution s without request i. We define $\Delta(i, s) = f(s) - f_{-i}(s)$ where $f_{-i}(s)$ is the cost of solution s without request i. The method repeatedly removes the request i with the highest cost $\Delta(i, s)$. The complexity of this operator is $O(q \times |\mathcal{L}|)$.
3. **Distance-Related Destroy Operator (DRDO)**: This operator removes a set of similar requests. For each request i, one evaluates the *distance-relatedness* r_{ij}^{dist} of i with other request $j \neq i$. We define a FTL version of the *distance-relatedness* measure as $r_{ij}^{dist} = \frac{1}{2}(d_{p_i d_j} + d_{d_i p_j})$ with d_{ij} the distance from i to j. The operator first randomly selects a request $i \in |\mathcal{L}|$. Then, it removes request j with the lowest *distance-relatedness* to request i, i.e. the request j closed to request i. The operator iterates until q requests have been removed. The complexity of this operator is $O(q \times |\mathcal{L}|)$.
4. **Time-Related Destroy Operator (TRDO)**: This operator belongs to the same family as **DRDO**. The *time-relatedness* of a request $i \in \mathcal{L}$ with another request $j \in \mathcal{L}$ is defined by $r_{ij}^{time} = |h_{p_i} - h_{p_j}| + |h_{d_i} - h_{d_j}|$. The complexity of this operator is $O(q \times |\mathcal{L}|)$.
5. **Route Destroy Operator (RODO)**: This operator removes all the requests from a route. The route is chosen randomly in the set of non-empty routes. **RODO** performs until q requests or more have been removed. This operation is performed in $O(q)$.
6. **Resource Destroy Operator (REDO)**: This operator removes all requests from a resource until q requests or more have been removed. The resource is selected randomly between the set of resources Π. This operation is performed in $O(q)$.
7. **Critical Destroy Operator (CDO)**: This operator removes a set of q requests with respect to their Forward Time Slack F, further detailed in Section 5. The *forward time slack* F_i of a vertex i is the quantity of time the service time of this vertex can be postponed without any time window violation. Requests are ordered in increasing order of criticality. The criticality of a request i is defined as $\min(F_{p_i}, F_{d_i})$. Its complexity is $O(|\mathcal{L}| \times \log(|\mathcal{L}|))$.

The **RDO, WDO** operators were introduced in Ropke and Pisinger (2005). Relatedness operators **DRDO** and **TRDO** are adapted from Shaw removal and used in Shaw (1998). **RODO** is presented in Lehuédé et al. (2013) and Demir et al. (2012). The **REDO** and **CDO** are new destroy operators fitted for our problem.

4.2. Repair operators

Repairing a solution implies inserting the requests from the request bank $\mathcal{B}$ in the solution routes and, if necessary, on resources. Hence, a repair operator has to decide at each iteration which request to insert, in which route, at which position in this route, and at which position on its associated resources. In this section we first propose a general framework for our repair operators. It is based on Ebben et al. (2005) and it inserts requests in routes and resources according to priority rules. The various kinds of priority rules are then discussed as well as the combination of rules that have been implemented to design our repair operators.

4.2.1. General framework of the repair operators

The proposed framework combines the principle of classical insertion heuristic for the VRPTW (Solomon, 1987) and serial scheduling heuristics (Kolisch, 1996). It consists of three decisions: (i) a request selection `RequestSelection`, (ii) a request insertion on route `RequestInsertion` and (iii) a request scheduling on resource `ResourceScheduling`. For each decision, we propose several rules. Some of them are inspired from the literature and others have been designed for the problem. The objectives of the decisions are the following:

- `RequestSelection`: determines the order in which the requests are selected for insertion in routes.
- `RequestInsertion`: determines in which route, and at which position in this route, the selected request should be inserted.
- `ResourceScheduling`: determines at which position on the used resource the pickup or delivery point of the selected request should be inserted.

We consider a partial solution s which is a solution to the problem with a set of non inserted requests ($\mathcal{B} \neq \emptyset$). The framework of the repair operators is proposed in Algorithm 2 .

In the following, we describe the rules proposed for each decision.

4.2.2. Selection of a request

In this section, we present five rules to select a request for insertion (decision `RequestSelection`).

- Earliest Due Date (**EDD**): select the request that has to be finished at the earliest time ($arg\,min_{r \in \mathcal{B}} l_{d_r}$). EDD is a rule that is often used in scheduling, see Brucker (2007).

Algorithm 2: Repair operators framework.

input : a partial solution s, a set of non inserted requests $\mathcal{B}$
output: a complete solution

```
1 begin
2   while no more feasible insertion do
      /* Select a request to insert                    */
3     r ← RequestSelection(B) ;
      /* Insert r into a route                         */
4     s ← RequestInsertion(r) ;
      /* Insert r on resources                         */
5     ResourceScheduling(r) ;
6     B ← B − {r} ;
7   end
8 end
```

- Earliest Delivery Release Date (**EDRD**): the request that has the smallest value for $e_{d_r}, r \in \mathcal{B}$ is selected for insertion.
- Number of Similar Requests (**NSR**): for a request $r_1 \in \mathcal{B}$, we define by NSR_{r_1} the number of similar requests, i.e. belonging to the same demand. Requests with the highest NSR are selected in priority.
- Cheapest Insertion (**CI**): select the request which has the cheapest cost of insertion comparing insertion cost in all positions in all routes.
- 2-Regret (**2R**): the regret value c_i of a request i is defined as $c_i = \Delta f_{i2} - \Delta f_{i1}$ with Δf_{i1} the cost of inserting i in its best route, Δf_{i2} the cost of inserting i in its second best route. The operator chooses the request i with the highest regret value c_i.
- Critical Resource (**CR**): the motivation of this operator is to reorder requests from a resource which contains a large number of requests in order to schedule them in a better way. Requests are sorted with respect to the resource they need. The requests associated with the most occupied resource are inserted in priority.

4.2.3. Insertion of a request in a route

Two rules are proposed to select the route and the position in the route to insert the selected request (`RequestInsertion`).

- Cheapest Insertion (**CI**): inserts the selected request at the position that generates the smallest increase in cost. For two identical costs, the request is inserted in the route that gives the earliest service time for the pickup.
- Earliest Truck Appending (**ETA**): inserts the selected request at the end of the route that implies the earliest service time for the pickup of this request.

4.2.4. Schedule of a request on a resource

Two methods are proposed to determine the position of inserted operations on the pickup or delivery resources (`ResourceScheduling`).

- Earliest Feasible (**EF**): inserts the operation on the resource at its earliest feasible position.
- Latest Feasible (**LF**): inserts the operation of the request on the resource at its latest feasible position.

4.2.5. Set of repair operators

The set of repair operators ($\mathcal{N}^+$) includes selected combinations of these three decisions. Benchmarks were conducted with all these repair operators. Preliminary experiments showed that combining **CI** with **LF** and **ETA** with **EF** induces an average decrease of the quality of the solution by 0.35%. The produced solution may be caught in a local extremum due to a bad scheduling on resources. Indeed, it is contradictory to insert a request at the end of a route (**ETA**) and its pickup and delivery nodes at the beginning of theirs resources (**EF**). The same observation can be made for the pair **CI+LF**.

Table 2
Combination of repair operators used.

`RequestInsertion`		CI	ETA
`ResourceScheduling`		EF	LF
`RequestSelection`	**EDD**	√	√
	EDRD	√	√
	NSR	√	√
	CI	√	
	2R	√	
	CR	√	√

As a result, we consider a set of ten repair operators which are presented in Table 2.

As in Ropke and Pisinger (2005), noise is introduced in repair operators to randomize the insertions. When a repair operator is selected, we randomly decide if noise should be used based on the probability of success of insertion with noise. For an insertion of cost C, the corresponding noised evaluation is defined by $C' = \max\{0, C + noise\}$. The amount of noise is a random number $noise$ in the interval $[-0.025 * d_{max}, 0.025 * d_{max}]$ where d_{max} is the maximum distance between two points of V. The value 0.025 is taken from Ropke and Pisinger (2005).

4.3. Stopping and acceptance criteria

A solution is accepted as the new current solution according to a simulated annealing criterion (Kirkpatrick et al., 1983). The ALNS terminates the optimization when one of the following stopping criteria is met:

- the maximum number of iterations Φ has been reached,
- τ iterations were performed without improving the best known solution,
- the time t_{limit} allowed to the algorithm is elapsed.

5. Feasibility of insertions

The ALNS is an algorithm that iteratively removes and inserts requests in a graph. A very large number of insertions are evaluated at each iteration: first in terms of cost and then in terms of feasibility, considering insertions in increasing order of cost until a feasible one is found. In particular, precedences between points, both on resources and routes, mean that we must check that requests are served within their time windows. To test if a solution is feasible, methods have been proposed in the literature. An efficient feasibility test is a key element for the performance of the algorithm. In this section, we summarize some results from the literature and propose an extension of a feasibility algorithm designed for the PDPT to handle resource synchronizations.

5.1. Insertion feasibility for the TSPTW

Savelsbergh (1985, 1992) introduced *forward time slacks* for the Traveling Salesman Problem with Time Windows (TSPTW). They represent the maximum amount of time by which a vertex u can be postponed without any time window violation for its successors. In the TSPTW, the idea is to perform a constant time feasibility check for local search operators. We first introduce some notations that are shared in this paper. The direct predecessor and direct successor of a vertex i are denoted by $\rho(i)$ and $\sigma(i)$ respectively. The set of all successors of a vertex i is denoted by

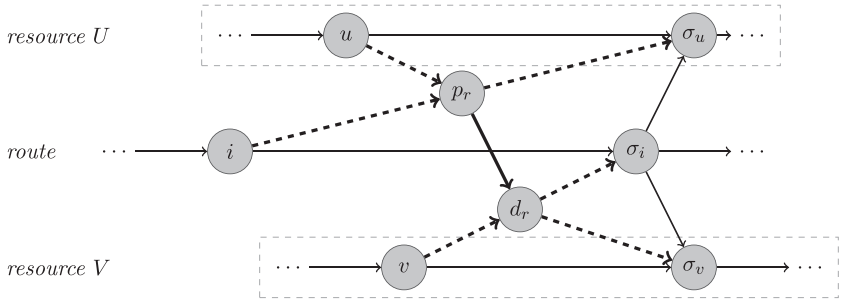

Fig. 4. Representation of the insertion of a request r in a route with precedences on resources U and V.

$\Gamma(i)$. $\psi_{u,v}$ is the ordered set of vertices on path $(u,\dots,v)$ and $\Omega_{u,v}$ is the set of all paths from u to v. We denote by $TWT_{\psi_{u,v}} = \Sigma_{i\in\psi_{\sigma(u),v}} w_i$, the total waiting time on path $(u,\dots,v)$. Cordeau et al. (2004) define the forward time slack F_u of a vertex u by:

$$F_u = \min_{i\in\{u\}\cup\Gamma(u)} \left\{ TWT_{\psi_{u,i}} + l_i - h_i \right\},$$

where h_i denotes the service time of a vertex i in an as early as possible schedule of the route.

5.2. Insertion feasibility for the PDPT

In the PDPT, different paths exist between two vertices because of precedence constraints in transfer nodes. The notion of slack time has been extended in this problem by Masson et al. (2013). First, the notion of a temporal graph is introduced. It represents all precedences between services at the various nodes of the problem as arcs. For every path $\omega \in \Omega_{u,v}$ in this graph, TWT_ω denotes the sum of waiting times along path ω in an as early as possible schedule of the solution. The slack time $ST_{u,v}$ between two vertices u and v is defined by:

$$ST_{u,v} = \min_{\omega\in\Omega_{u,v}} TWT_\omega.$$

If there is no path between u and v, $ST_{u,v}$ is equal to $+\infty$. According to this definition, the forward time slack for vertex u can be rewritten as follows:

$$F_u = \min_{i\in\{u\}\cup\Gamma(u)} \left\{ ST_{u,i} + l_i - h_i \right\}.$$

This framework has been proposed to model operations synchronization with precedences in the PDPT but, as shown in Drexl (2014), it can be extended easily to model other categories of synchronizations in VRPs. In particular, it was extended to model *exact operations synchronization* in Grangier et al. (2016). An equivalent framework was proposed by Afifi et al. (2016) in a simulated annealing algorithm to solve the VRP with time windows and synchronized visits.

5.3. Insertion feasibility for the FT-PDP-RS

We introduce a new temporal graph G_t which includes all vertices for the FT-PDP-RS. The set of vertices in G_t contains all the vertices of G. Each precedence between successive vertices in a route or on a resource is modeled as an arc in G_t. The weight of an arc between vertices i and j is the minimum time between the service at i and j.

Fig. 4 illustrates the insertion in G_t of a request r for which both pickup p_r and delivery d_r vertices are synchronized on their respective resources U and V. On resource U, two vertices u and σ_u are already scheduled. On resource V, two vertices are scheduled v and σ_v. The request r composed of pickup vertex p_r and delivery vertex d_r is inserted on the route after vertex i. The pickup vertex p_r is inserted after vertex u on resource U and the delivery vertex d_r is inserted after vertex v on resource V. Before the insertion of request r, precedences between σ_i and σ_u and between σ_i and σ_v can exist.

Table 3
New precedences introduced in the solution if the insertion of Fig. 4 is performed.

	σ_i	σ_u	σ_v
i		→	→
u	→		→
v	→	→	

To ensure the feasibility of an insertion, one has to verify two conditions (Masson et al., 2013):

1. The insertion does not create any cycle in the temporal graph G_t.
2. The insertion does not violate any time window.

5.3.1. Cycle detection

This insertion creates new precedences between the nodes of the current solution: To ensure the feasibility of an insertion, we have to first check if cycles have been introduced in the graph. A minimal set of new precedences for the insertion of Fig. 4 are represented in Table 3. The objective is to check that they do not contradict a precedence that already exists in the solution. Indeed, cycles are created if $i \in \Gamma(\sigma_u)$, $i \in \Gamma(\sigma_v)$, $u \in \Gamma(\sigma_v)$, $u \in \Gamma(\sigma_i)$, $v \in \Gamma(\sigma_i)$. With a matrix of successors (size $|V| \times |V|$), we can check in constant time if an insertion creates a cycle.

5.3.2. Temporal feasibility

In addition, the feasibility of the insertion is evaluated with respect to temporal constraints, testing if time windows are violated. We denote by $\bar{h}_i$ the time at which the service begins at point i after the insertion of a request r. Algorithm 3 presents the sequence of tests that involves the preprocessing variables and verifies the feasibility of the temporal constraints of the problem. This algorithm, which is an extension of the algorithm provided in Masson et al. (2013) and has a constant time complexity, describes the general case with synchronizations on pickup resource and delivery resource.

Hence, an insertion can be performed in constant time under two conditions: (i) all service times can be scheduled as early as possible, (ii) the value of F_v for all $v \in G_t$ and the matrix of successors have been updated after each removal or insertion. G_t corresponds to a PERT diagram. Before each request insertion, slack times are computed with the $O(n^3)$ Floyd–Warshall algorithm. This preprocessing considerably reduces the complexity and the computational time for the test of many insertions.

Algorithm 3: Feasibility of the insertion of a request r.

input : a request r to insert after vertex i with pickup point p_r to insert before σ_u and delivery point d_r to insert before σ_v
output: **return** TRUE if the insertion is feasible, FALSE otherwise

begin
  /* Insertion of point p_r */
  $\bar{h}_{p_r} = \max(e_{p_r}, h_i + s_i + t_{i,p_r}, h_u + s_u)$
  if $\bar{h}_{p_r} > l_{p_r}$ **then**
    return FALSE
  end
  /* Evaluate the shift at point σ_u */
  $\delta_{\sigma_u} = \bar{h}_{p_r} + s_{p_r} - h_{\sigma_u}$
  if $\delta_{\sigma_u} > F_{\sigma_u}$ **then**
    return FALSE
  end
  /* Evaluate the beginning of the service at point d_r */
  $\bar{h}_{d_r} = \max(e_{d_r}, \bar{h}_{p_r} + s_{p_r} + t_{p_r,d_r}, h_v + s_v)$
  if $\bar{h}_{d_r} > l_{d_r}$ **then**
    return FALSE
  end
  /* Evaluate the shift at point σ_v */
  $\delta_{\sigma_v} = \bar{h}_{d_r} + s_{d_r} - h_{\sigma_v}$
  if $\delta_{\sigma_v} > F_{\sigma_v}$ **then**
    return FALSE
  end
  /* Evaluate the shift at point σ_i */
  $\delta_{\sigma_i} = \bar{h}_{d_r} + s_{d_r} + t_{d_r,\sigma_i} - h_{\sigma_i}$
  if $\delta_{\sigma_i} > F_{\sigma_i}$ **then**
    return FALSE
  else
    return TRUE
  end
end

6. Instances

This section presents the parameters of the set of instances used in the computational experiments. The method has been used to experiment with two sets of instances. The first set is taken from the literature (Hirsch, 2011). The second set corresponds to real instances from the Luc Durand company,[2] which specializes in public works in France.

6.1. Instances from the literature

Our algorithm is first tested on the instances of Hirsch (2011).[3] The studied problem (LTSP) is quite similar to our problem since it deals with full truckloads collections and deliveries. This problem does not integrate synchronization on resources, which nevertheless appears in the formulations of El Hachemi et al. (2013) and Bredström and Rönnqvist (2008). The fleet of vehicles is heterogeneous with no fixed costs. 40 instances are proposed and divided in two sets of instances. Set 1 contains the instances from 1 to 20 (H1 - H20) and set 2 from 21 to 40 (H21 - H40).

[2] http://www.durandtp.fr

[3] The authors gratefully thank Mr. Hirsch for providing his data sets for our numerical experiments.

Table 4
Fleet of trucks for real-life instances.

Type	Number	Capacity (*tons*)
TT	18	25
8X4	9	16
6X4	13	12
4X2	5	10

Table 5
Main characteristics of the real-life instances. $|R_{sch}|$ (respectively $|R_{unsch}|$) represents the number of *scheduled* requests (resp. *unscheduled* requests). $|R|$ represents the total number of associated requests.

Instance	$\lvert R_{sch}\rvert$	$\lvert R_{unsch}\rvert$	$\lvert R\rvert$
OS22	4	18	22
OS32	9	21	30
OS49	11	38	49
OU35	0	35	35
OU51a	0	51	51
OU51b	0	51	51
OU85	0	85	85

In both sets, 30 requests have to be delivered with ten trucks belonging to two different families. Because of weight limit consideration, there are compatibility restrictions between vehicles and requests. These instances represent a daily problem with a global time window of [0, 1440]. The network is composed of 589 sites and deliveries are gathered on three sites for set 1 and four sites for set 2. In set 2, requests have higher service times and vehicles have more operation time (780 min against 500 min).

6.2. Real case study

We tested our algorithm on real-life instances from the public works company. Seven instances are considered and divided in two groups. The instances involve 20 sites including one depot. Four categories of trucks are available to serve the demands (see Table 4).

Instances OS22, OS32, OS49 mix *scheduled* and *unscheduled* demands while instances OU35, OU51a, OU51b, OU85 contain only *unscheduled* demands. Table 5 presents the main characteristics of the instances. The considered period is one day from 6:00 to 18:00 which corresponds to a time horizon of 12 h. The depot and the sites are opened during the whole day. Demands have time windows and starting times in the case of *scheduled* demands. The service time for the collection and the delivery is ten minutes. Note that OU85 corresponds to an extreme case with regards to what can be met by the company for a single day demand.

7. Experiments

In the following, we present some numerical results for our algorithm. The initialization and the parametrization of the ALNS are first covered. To evaluate its performance, it is applied to solve instances from the literature, which are adapted to evaluate the impact of resource synchronization. Benchmarks on real-life instances are finally presented. The algorithms are coded in C++ and executed on a Xeon X5650 at 2.67 GHz with 64GB of RAM.

7.1. Initialization of the ALNS

The initial solution is obtained from a two-phase heuristic method considering the previous work presented in Grimault et al. (2014). According to the preliminary experiments presented

Table 6
Description of the four configurations of the adaptive layer tested with $t_{limit} = 600$ s.

Configuration	σ_1	σ_2	σ_3
*AD*0	–	–	–
*AD*1	1	0	5
*AD*2	33	9	13
*AD*3	33	20	13

in the previous paper, the initial solution is obtained using the **EDD+CI+EF** combination: requests are sorted using the Earliest Due Date (**EDD**) rule, the insertion operator on routes is Cheapest Insertion (**CI**) and requests are scheduled as early as possible on resources (**EF**). This heuristic provides a feasible initial solution for all instances. It shows good performances for *scheduled* requests, which represent a major difficulty in our problem.

7.2. Calibration of the ALNS

The literature on the ALNS proposes different tunings for the algorithm depending on the type of problem solved. Many parameters may be fitted to produce a better solution. To achieve this goal, we focus on the tuning on four set of parameters: (1) the adaptive layer, (2) the noise function, (3) the quantity of removable requests and (4) the performance of destroy operators. All tests were conducted on two subsets of instances with ten runs for each instance. The first set contains five instances from Hirsch (2011) (H6 - H10 - H26 - H29 - H34) and the second set contains three instances from our industrial partner (OS22 - OS32 - OS49). A time limit is used for the stopping criterion, $t_{limit} = 600s$, which represents a good compromise for the allowed time in an industrial application context. The choice of a fixed time allows a fair comparison of parameters and operators, independently of their respective complexity.

7.2.1. The adaptive layer on the ALNS

The initial selection probabilities of operators are set to $\frac{1}{|\mathcal{N}^-|}$ for destroy operators and $\frac{1}{|\mathcal{N}^+|}$ for repair operators. The probability for each operator is updated according to Ropke and Pisinger (2005). At each iteration of the ALNS, the score of the selected operator i is updated with either σ_1, σ_2, σ_3 according to score adjustments presented in Ropke and Pisinger (2005). The search is divided into time segments of 100 iterations. At the end of each segment, selection probabilities are updated and scores of operators are reset to zero. Based on papers from the literature, we test four configurations of the adaptive layer, presented in Table 6. The first configuration, named "AD0" does not implement the adaptive layer and the scores of the operator remain at their initial values. This setting is shown to be the most efficient in Lehuédé et al. (2013). The second configuration "AD1" is obtained from Demir et al. (2012). The third configuration "AD2" s the original configuration proposed by Ropke and Pisinger (2005). The last configuration "AD3 is derived from the settings of Masson et al. (2014).

Table 7 compares the four configurations on the test instances, based on the percentage deviation from the best known solution.

The results confirm a slight influence of the parameters of the adaptive layer on the efficiency of the ALNS. Configuration "AD3" presents the best results compared to other configurations. This configuration proposes the best average deviation and is the best four times out of eight.

7.2.2. Influence of noise

We propose to study the impact of the noise function in the repair operators. The cost of an insertion can be computed without noise (C) and with noise (C'). We will look at the percentage of deviation of the average objective from the best objective for each instance.

Table 7
Results for four configurations of the adaptive layer: the table reports the average percentage deviation from the best known solution with $t_{limit} = 600$ s for each configuration on eight test instances.

Instance	Dev_{AD0}	Dev_{AD1}	Dev_{AD2}	Dev_{AD3}
OS22	**0.25**	0.90	0.77	0.65
OS32	**0.01**	**0.01**	0.06	0.17
OS49	0.98	1.44	**0.67**	0.68
H6	0.70	1.06	1.07	**0.64**
H10	0.73	1.09	0.74	**0.63**
H26	2.22	2.69	2.44	**1.94**
H29	1.21	1.19	**0.92**	1.10
H34	1.26	1.08	0.97	**0.85**
average	0.92	1.18	0.96	**0.83**

Table 8
Results of the insertion cost without and with the noise as the average percentage deviation from the best known solution with $t_{limit} = 600s$.

Instance	Dev_{with} (%)	$Dev_{without}$ (%)
OS22	1.04	**0.65**
OS32	**0.13**	0.23
OS49	0.38	0.38
H6	**0.68**	0.92
H10	**0.53**	1.03
H26	2.83	**2.43**
H29	**0.48**	0.54
H34	0.82	**0.70**
average	0.86	0.86

Table 9
Results for five configurations of the number of requests removed: the table presents the average deviation percentage from the best known solution with $t_{limit} = 600$ s.

Instance	[5, 10]	[10, 20]	[10, 35]	[10, 40]	[10, 50]
OS22	0.82	0.51	0.82	0.79	**0.43**
OS32	0.14	0.06	**0.01**	0.17	0.34
OS49	0.74	**0.42**	0.53	0.58	0.56
H6	**0.64**	0.66	0.73	0.75	0.73
H10	1.10	1.03	0.97	**0.68**	0.73
H26	3.14	**2.36**	2.68	2.55	2.72
H29	0.99	0.97	1.10	1.34	**0.92**
H34	1.38	1.15	1.15	**0.90**	1.22
average	1.12	**0.89**	1.00	0.97	0.96

Table 8 shows that the contribution of the noise function is not clear in our case. The average deviations are equal with the noise or without the noise for this set of experiments. In the following, the noise function is used. It can be noted that a similar result has been observed in Bortfeldt et al. (2015).

7.2.3. Number of removed requests

The influence of the range for the number q of removed requests has been studied by Demir et al. (2012) as well as Masson et al. (2014). We propose five different configurations, selecting the percentage of requests to remove within an interval $[\underline{s}, \bar{s}]$, with $\underline{s}$ the lower bound and $\bar{s}$ the upper bound. Table 9 presents the deviation of the average solutions from the best known solutions for the five configurations on the test instances.

Table 9 shows that ALNS is not very sensitive to this parameter, although [5, 10] shows a greater average gap than the other intervals. One configuration outperforms on these results: [10, 20]. Although this study is not exhaustive, we choose the quantity of

Table 10
Influence of the removal operators on the average deviation from the best known solution. Each column indicates the performance of the ALNS without the destroy operator indicated above with $t_{limit} = 600$ s.

Instance	RDO	WDO	DRDO	TRDO	CDO	RODO	REDO	ALL
OS22	**0.08**	0.74	0.82	0.82	0.42	0.90	1.35	0.51
OS32	0.14	0.04	**0.02**	0.40	0.24	0.06	0.27	0.06
OS49	0.44	0.73	0.40	0.47	0.38	**0.33**	0.35	0.42
H6	0.83	0.75	0.55	0.73	0.80	**0.51**	0.52	0.66
H10	0.86	0.81	0.91	0.92	0.94	**0.59**	0.82	1.03
H26	3.51	3.29	3.76	3.45	2.71	3.14	2.82	**2.36**
H29	**0.96**	1.19	1.07	1.44	1.19	1.24	1.22	0.97
H34	1.13	1.26	1.11	**0.86**	1.08	1.29	1.10	1.15
average	0.99	1.10	1.08	1.14	0.97	1.01	1.06	**0.89**

requests to remove between 10% and 20% of the total number of requests. This configuration permits more iterations of the ALNS since neighborhoods are smaller on average.

7.2.4. Performance of destroy operators

Finally, we study the contribution of destroy operators to the ALNS. For each test, we run the ALNS with all destroy operators except one. Table 10 shows the average deviation from the best known solution for each instance and each configuration.

The results do not show significant positive or negative contributions of an operator (maximum deviation is 0.1% to the best configuration). We propose to keep all operators in the following experiments since this combination has the best results.

7.3. Final parameters of the ALNS

Despite all experiments conducted in the calibration procedure, we admit that ALNS is not very sensitive to all parameters. The chosen parameters globally offer the best potential of the ALNS. To summarize and complete the conclusions of our experiments on test instances, we set the roulette wheel parameters as follows $(\sigma_1, \sigma_2, \sigma_3, r) = (33, 20, 13, 0.1)$ where r is the reaction factor which controls weight adjustments (Ropke and Pisinger, 2005). The roulette wheel mechanism is reset every 100 iterations. For the stopping criteria, the total number of iterations of the algorithm is $\Phi = 25000$, the total number of iterations without improvement is $\tau = 5000$ and the total time limit for the algorithm is $t_{limit} = 3600s$. The simulated annealing starts with a temperature which is a function of the cost of the initial solution. At each iteration, the percentage of removed requests is taken with $(\xi_{min}, \xi_{max}) = (0.1, 0.2)$.

Table 12
Results obtained by the ALNS method compared with the results obtained by Hirsch (2011) for the set 2 instances.

Instance	TS-HIRSCH	$ALNS_0$	
	Best value	Best value	GAP_H %
H21	4749.97	**4747.32**	**−0.06**
H22	**5425.99**	**5425.99**	**0**
H23	**4866.23**	**4866.23**	**0**
H24	**5255.53**	**5255.53**	**0**
H25	6103.5	**6101.86**	**−0.03**
H26	4686.41	**4652.96**	**−0.71**
H27	**6058.37**	**6058.14**	**0**
H28	**5151.54**	**5151.54**	**0**
H29	**5845.09**	**5845.09**	**0**
H30	5347.46	**5340.69**	**−0.13**
H31	5059.85	**5058.21**	**−0.03**
H32	**4691.05**	**4691.05**	**0**
H33	4919.39	**4911.39**	**-0.16**
H34	**5529.69**	**5529.69**	**0**
H35	6469.63	**6466.6**	**−0.05**
H36	4878.02	**4872.66**	**−0.11**
H37	5595.49	**5566.61**	**−0.52**
H38	**5648.81**	**5649.04**	**0**
H39	**5111.85**	**5111.85**	**0**
H40	4677.23	**4626.24**	**−1.09**
average	–	–	−0.14

7.4. Comparison with instances from the literature

To perform a first assessment of the performance of our ALNS algorithm we compare it to the Tabu Search method of Hirsch (2011) on Hirsch instances for the LTSP (described in Section 6.1). We present the results of this comparison in Tables 11 and 12. The Tabu Search is denoted by TS-HIRSCH. The ALNS without the synchronization module is denoted by $ALNS_0$.

Table 11
Results obtained by the ALNS method compared with the results obtained by Hirsch (2011) for the set 1 instances.

Instance	LB	TS-HIRSCH		$ALNS_0$		
		Best value	GAP_{LB} %	Best value	GAP_{LB} %	GAP_H %
H1	2593.37	2596.56	0.12	**2596.13**	0.11	**−0.02**
H2	2371.83	2380.41	0.36	**2378.64**	0.29	**−0.07**
H3	1969.26	**1998.42**	1.48	**1998.42**	1.48	**0**
H4	2399.06	2405.79	0.28	**2405.43**	0.27	**−0.01**
H5	2191.48	**2217.85**	1.2	**2217.85**	1.2	**0**
H6	2496.35	**2610.76**	4.58	**2610.76**	4.58	**0**
H7	2430.39	**2529.03**	4.06	**2529.03**	4.06	**0**
H8	2225.52	**2251.39**	1.16	**2251.39**	1.16	**0**
H9	2298.11	**2324.46**	1.15	**2324.46**	1.15	**0**
H10	2394.29	**2489.35**	3.97	2492.82	4.12	0.14
H11	2264.16	2290.32	1.16	**2290.03**	1.14	**−0.01**
H12	2529.86	**2542.42**	0.5	**2542.42**	0.5	**0**
H13	2305.7	**2309.81**	0.18	**2309.81**	0.18	**0**
H14	2264.43	**2320.79**	2.49	**2320.79**	2.49	**0**
H15	2225.52	**2325.69**	4.5	**2325.69**	4.5	**0**
H16	2274.25	**2367.86**	4.12	**2367.86**	4.12	**0**
H17	2382.47	**2481.33**	4.15	**2481.33**	4.15	**0**
H18	2280.96	**2325.52**	1.95	**2325.52**	1.95	**0**
H19	2656.41	**2736.1**	3	**2736.1**	3	**0**
H20	2227.7	**2233.71**	0.27	2234.69	0.31	0.04
average	–	–	2.03	–	2.04	0.00

The comparisons are made based on the best value out of 10 runs of the ALNS on each instance. Column "LB" provides a *lower bound* for each instance, computed by Hirsch (2011) with a relaxation of the problem solved by the `Xpress` solver. Columns named "Best value" indicate the value obtained by the associated method. GAP_{LB} denotes the percentage deviation of the best value of the proposed method from the *lower bound*. GAP_H shows the percentage deviation of $ALNS_0$ from TS-HIRSCH and is defined by $GAP_H = \frac{Best_{ALNS_0} - Best_{TS-HIRSCH}}{Best_{TS-HIRSCH}}$. A positive value indicates a worse "Best value" of our method and a negative value indicates a better one.

As shown in Tables 11 and 12, our ALNS algorithm is competitive with the TS-HIRSCH. The ALNS is able to find four new best solutions from instances set 1, and 10 new best solutions from instances set 2. Although the algorithm does not improve all solutions, the average deviation is minimal with 0.00% for set 1 (ALNS is equivalent to TS-HIRSCH for 14 instances) and −0.14% for set 2. These tables confirm that of our ALNS algorithm in able to find good solutions to a relaxed version of the considered problem on academic instances. It does not seem to remain trapped in local optima when solving the LTSP.

Regarding run-times, the average execution time of our ALNS algorithm is about 200 s over all instances and does not exceed 630s (on a Xeon X5650 at 2.67 GHz with 64 GB of RAM). The detailed runtime for the best solutions found in Hirsch (2011) is not clearly provided, but to our understanding, it ranges between 1560s and 80000s on a Pentium IV processor with 2.52 GHz and 512 MB RAM.

7.5. Impact of the resource synchronization constraints

To evaluate the influence of resource synchronization constraints, we modify the LTSP, considering all sites as a resource of unitary capacity: only one truck can unload or load at a given time on a given resource. These corresponding instances are suffixed with "−RS". To handle the synchronization constraints, we add the specific operators concerning resources in our ALNS, namely the destroy operator REDO and the repair operators **CR+CI+EF** / **CR+ETA+LF**. The modified ALNS with these synchronization operators is denoted $ALNS_1$. We compare the best known results on instances H (set 1 and set 2) with the results obtained with $ALNS_1$ for instances "−RS". Tables 13 and 14 present the comparison between the best known value for the instance without resource synchronization constraints, column "BKS", and our algorithm, column "$ALNS_1$". For this benchmark, we consider the best solution obtained out of five runs of our algorithm for each instance.

The results show that resource synchronization has a minor impact on the cost for these instances. In particular, the average deviation from the best known solutions without synchronization is 0.27% for instances H1-RS - H20-RS and 0.42% for instances H21-RS - H40-RS. A second remark is that ALNS still performs well despite the complexity induced by synchronization constraints.

To further evaluate the impact of resource synchronization on the Hirsch instances, we multiply the service time on delivery points by a so called resource occupancy factor $f \in \{0.5, 1, 1.5, 2, 2.5, 3\}$. We modify the service time on delivery since, in this problem, the considered delivery locations are shared by many requests. They correspond to factories where log trucks from many harvesting sites unload. Five runs of the algorithm were conducted for each instance.

A synthesis of traveling costs of the produced solutions is presented on Fig. 5. We observe that on average, the gap to the best known solution without synchronization is correlated to the increase in the occupation of resources. However, this gap is not increased by more than 2%, even when the occupancy rate is multiplied by 3.

This conclusion has to be put in perspective with the number of vehicles used in each solution. In Hirsch's instances, a maximum of 10 trucks is allowed per instance and each truck is given a maximum operating time. Nevertheless, no fixed cost is associated with using a truck. Fig. 6 shows the average number of vehicles in solutions for each instance with respect to the factor f. We observe here that the impact of increasing resource occupancy is not negligible. In addition, due to the fleet size limitations and time windows, ALNS is not always able to find complete solutions: from $f = 2.5$, one request remains in the request bank at the end of

Table 13
Impact of the synchronization constraints on resources on set 1 instances. Results obtained by $ALNS_1$ compared with the best known results (without resource synchronization constraints, column "BKS").

Instance	BKS	$ALNS_1$	
	Best value	Best value	GAP_{BKS} %
H1-RS	2596.13	2599.48	0.13
H2-RS	2378.64	2381.27	0.11
H3-RS	1998.42	2010.33	0.60
H4-RS	2405.43	2408.18	0.11
H5-RS	2217.85	2226.63	0.40
H6-RS	2610.76	2625.3	0.56
H7-RS	2529.03	2535.11	0.24
H8-RS	2251.39	2258.5	0.32
H9-RS	2324.46	2325.45	0.04
H10-RS	2489.35	2503.56	0.57
H11-RS	2290.03	2296.09	0.26
H12-RS	2542.42	2547.34	0.19
H13-RS	2309.81	2312.44	0.11
H14-RS	2320.79	2324.37	0.15
H15-RS	2325.69	2329.19	0.15
H16-RS	2367.86	2371.86	0.17
H17-RS	2481.33	2486.66	0.21
H18-RS	2325.52	2335.41	0.43
H19-RS	2736.1	2747.42	0.41
H20-RS	2233.71	2238.5	0.21
average	–	–	0.27

Table 14
Impact of the synchronization constraints on resources on set 2 instances. Results obtained by $ALNS_1$ compared with the best known results (without resource synchronization constraints, column "BKS").

Instance	BKS	$ALNS_1$	
	Best value	Best value	GAP_{BKS} %
H21-RS	4747.32	4760.41	0.28
H22-RS	5425.99	5426.54	0.01
H23-RS	4866.23	4866.23	0
H24-RS	5255.53	5280.33	0.47
H25-RS	6101.86	6124.25	0.37
H26-RS	4652.96	4725.18	1.55
H27-RS	6058.14	6094.37	0.60
H28-RS	5151.54	5172.73	0.41
H29-RS	5845.09	5865.07	0.34
H30-RS	5340.69	5352.39	0.22
H31-RS	5058.21	5060.08	0.04
H32-RS	4691.05	4692.46	0.03
H33-RS	4911.39	4936.19	0.50
H34-RS	5529.69	5541.32	0.21
H35-RS	6466.6	6511.03	0.69
H36-RS	4872.66	4890.2	0.36
H37-RS	5566.61	5603.37	0.66
H38-RS	5648.81	5673.83	0.44
H39-RS	5111.85	5111.85	0
H40-RS	4626.24	4684.43	1.26
average	–	–	0.42

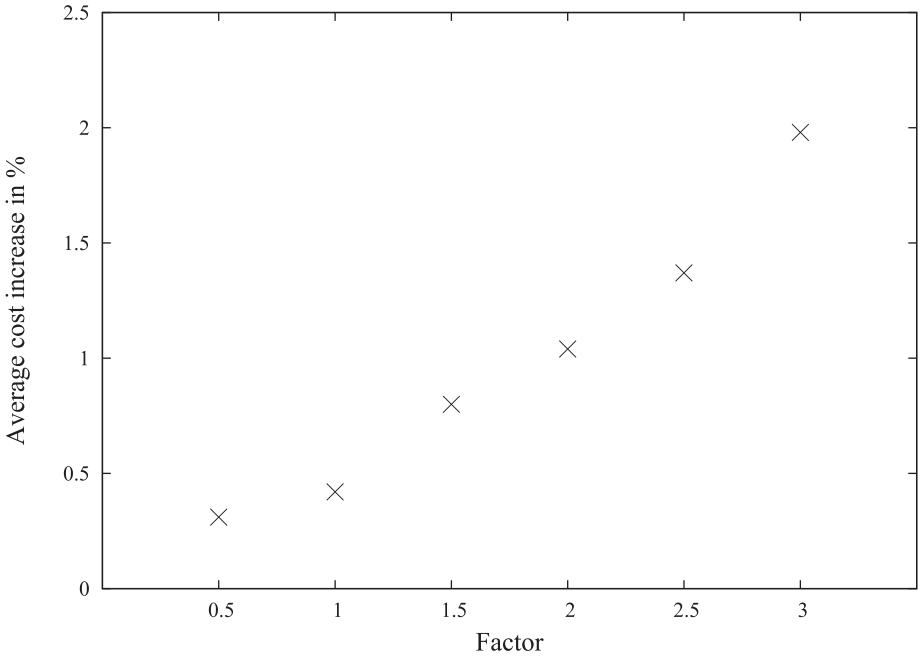

Fig. 5. Influence of the resource occupancy factor on average cost on instances H1-RS - H40-RS.

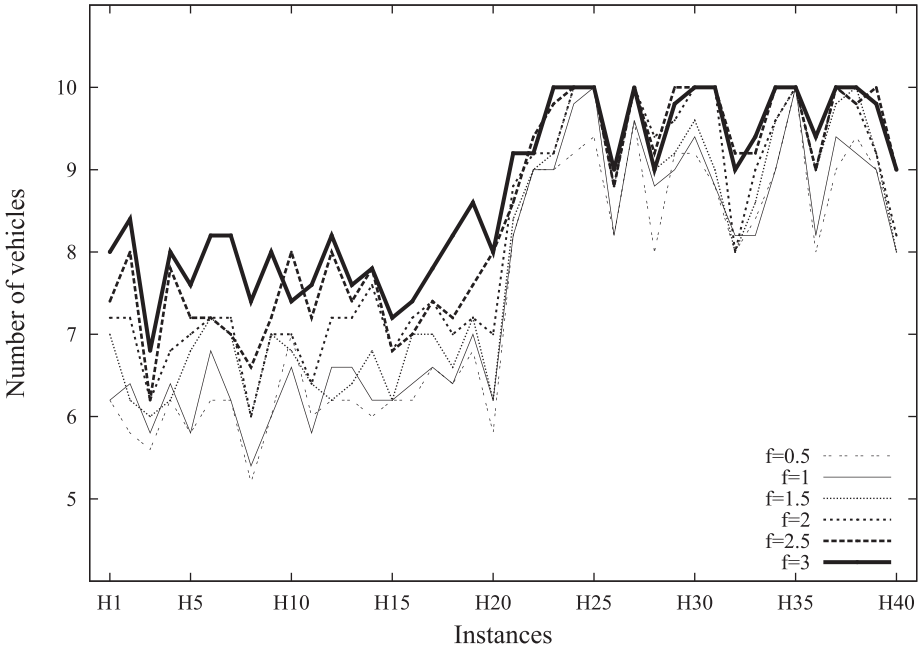

Fig. 6. Influence of the resource occupancy factor on the number of vehicles on instances H1-RS - H40-RS.

the ALNS for instance H27-RS. This is also the case for $f = 3$ for instance H29-RS.

7.6. Benchmarks on real-life instances

In this section, we propose to solve real instances of a public works company (instances named "O"). In a first part, these instances are solved with a MIP solver using a straightforward implementation of model (1)–(10). Second, ALNS is applied to these instances with the parameters defined in the calibration procedure. Its performances are evaluated with respect to the solver results. Finally, the stability and convergence of the proposed ALNS are analyzed, extending the termination limits determined for the industrial case.

7.6.1. Results of the MIP solver

In this section, the MIP solver `Gurobi 6.5.1` was used to solve the real instances with the model proposed in Section 3. Several experiments are presented in Table 15. Two tests were conducted with different time limits: 600 s and 7200 s. Column G_{600} and G_{7200} reports the best solution found after 600 s and 7200 s of computational time, respectively. The lower bounds returned by the solver in the 7200 s runs are reported in column G_{LB}. The gap to the lower bound (in %) is reported on columns GAP_{LB} for each test.

Over 600 s or 7200 s of computational time, the solver is not able to find any optimal solution on any instance. One interesting observation is that the lower bound provided by the solver is not

Table 15
Results obtained with the Gurobi solver on real instances. The table presents the best lower bound found for each instance (G_{LB}), the best solution found with 600 s (G_{600}) or with 7200 s (G_{7200}) and the associated gap to the lower bound.

Instance	G_{LB}	600s		7200s	
		G_{600}	GAP_{LB}%	G_{7200}	GAP_{LB}%
OS22	2446.00	2776.95	13.53	**2546.44**	4.11
OS32	4527.28	4909.84	8.45	**4820.26**	6.47
OS49	6159.78	6992.72	13.52	**6806.04**	10.49
OU35	2368.73	3996.12	68.70	**3545.65**	49.69
OU51a	3879.12	6507.23	67.75	**5870.73**	51.34
OU51b	3596.37	6013.83	67.22	**5892.94**	63.86
OU85	3938.40	–	–	–	–

Table 16
Results obtained with the ALNS on real instances. The table presents the best found solution (BFS), either by the MIP solver or the ALNS, the best solution found (A^{best}_{25000}) by the ALNS, the average deviation to BFS, the average cpu time in seconds needed by the 2500 iterations of the ALNS and the average solution (A^{avg}_{25000}). The stopping criteria are those from Section 7.3 ($\Phi = 25000$, $\tau = 5000$ and $t_{limit} = 3600$ s).

Instance	G_{7200}	A^{best}_{25000}	GAP_{G-A}%	*time*	A^{avg}_{25000}
OS22	**2546.44**	2779.92	9.17	60	2798.24
OS32	**4820.26**	4840.26	0.41	162	4845.66
OS49	6806.04	**6384.11**	−6.21	655	6403.78
OU35	3545.65	**3276.03**	−7.60	182	33370.98
OU51a	5870.73	**5374.81**	−8.45	615	5386.12
OU51b	5892.94	**4985.63**	−15.40	540	5048.11
OU85	–	**5902.89**	–	1506	6054.47
average		–	−4.68	–	

improved over the search. However, the lower bounds found for OS instances are much better than those found for OU instances. For the biggest instance OU85, the solver is not able to find any solution due to memory overflow. Not surprisingly for an arc base formulation of a VRP, the straightforward implementation of the model in a commercial solver does not give satisfactory results for medium to large instances, in particular when the time is limited to a reasonable amount.

7.6.2. Results of the ALNS with parameters defined by the calibration procedure

These experiments have been performed using the parameters defined in the calibration procedure. Table 16 summarizes the results returned by the proposed ALNS setting on the realistic instances. For each instance, ten runs of the ALNS were performed. For comparison, we report the best solution found by the solver (in column G_{7200}), the best solution found by the ALNS over the ten runs (column A^{best}_{25000}), and the gap between the two results (column GAP_{G-A}). The average computational time in seconds over the ten ALNS runs is reported in column *time*. The average solution cost is reported in A^{avg}_{25000}.

For five instances out of seven, the ALNS is able to find a better solution than the solver. We observe a positive correlation between the computational time of the ALNS and the number of requests. These results also show that instances with scheduled requests are more difficult to solve than instances with unscheduled requests. For instance OS22, the solver finds a solution that is significantly better than the best ALNS solution. We ascribe this behavior to the fact that requests are concentrated on few resources on this instance. Resource constraints significantly complicate the search for the ALNS and the returned solution uses an extra vehicle with respect to the best solver solution. The gap between the average solution cost over 10 runs and the best solution cost remains very small for each instance, which denotes a good stability for the algorithm.

7.6.3. Impact of the number of iterations

This section compares the performance of the ALNS with $\Phi = 50000$ iterations as unique stopping criterion (ie. $\Phi = 50000$, $\tau = +\infty$ and $t_{limit} = +\infty$) to the previous results. Ten runs of each real instance were performed and results are reported in Table 17. Column BFS points out the best solution found between the solver and the two ALNS configurations. We compare the obtained results with those of the configuration from Section 7.6.2.

These results show that the solution is improved in all cases except for instance OS22. For configuration A_{50000}, computational times are higher (at least 6 times), but the cost improvement remains relatively small. The A_{25000} configuration is considered satisfying by the company.

8. Conclusion

We have presented in this paper a new problem denoted FT-PDP-RS arising in public works companies. We developed an ALNS to solve the FT-PDP-RS with some specific repair and destroy operators. An efficient method was proposed to verify in constant time the feasibility of the insertion of a request. To calibrate our algorithm, we have performed some tests on relevant parameters of the ALNS. The algorithm was tested on instances from the literature and real case instances from a public works company. The algorithm was first evaluated without synchronization constraints on data sets from the literature. The results demonstrate that our ALNS is competitive compared to the best known solutions of these instances, and can even improve some of the solutions. When vehicles have to be synchronized on resources, we have shown that the impact on traveling costs remains limited on the considered instances, but the number of needed vehicles increases significantly. On instances from a public works company,

Table 17
Results obtained with the ALNS on real instances. The table presents the best solution found by the three compared algorithms (BFS), the best solution and the average deviation to BFS returned by the two configurations of ALNS and the cpu time in seconds. A_{25000} denotes the settings that have been chosen to suit industrial constraints in term of runtime ($\Phi = 25000$, $\tau = 5000$ and $t_{limit} = 3600$ s), whereas A_{50000} correspond to the settings $\Phi = 50000$, $\tau = +\infty$ and $t_{limit} = +\infty$.

Instance	BFS	A_{25000}	GAP_{BFS}%	time	A_{50000}	GAP_{BFS}%	time
OS22	2546.44	2779.92	9.17	60	2779.92	9.17	556
OS32	4820.26	4840.26	0.41	162	4839.84	0.41	1127
OS49	6364.38	6384.11	0.31	655	**6364.38**	0.00	3825
OU35	3264.27	3276.03	0.36	182	**3264.27**	0.00	1566
OU51a	5374.81	**5374.81**	0.00	615	**5374.81**	0.00	3665
OU51b	4950.18	4985.63	0.72	540	**4950.18**	0.00	4208
OU85	5725.13	5902.89	3.10	1506	**5725.13**	0.00	3601
average	–	–	2.01	–	–	1.36	–

ALNS provides good solutions within a satisfying solving time for regular size instances and is highly competitive with respect to a solver. All these results show the performance of the ALNS algorithm to solve the FT-PDP-RS problem. For further research, we will introduce constraints relative to transportation companies such as the limitation on the working hours and driving time of drivers as well as the minimization of route duration.

Acknowledgments

The authors want to thank Patrick Hirsch for kindly sharing his instances and results.

This work is part of project ORLoGES (Optimisation du Réseau Logistique en Génie Civil, avec les aspects Economiques et Sociétaux) funded by “Caisse des Dépôts” and labeled by the competitivity cluster Nov@log in France.

References

Afifi, S., Dang, D.-C., Moukrim, A., 2016. Heuristic solutions for the vehicle routing problem with time windows and synchronized visits. Optim. Lett. 10 (3), 511–525.
Arunapuram, S., Mathur, K., Solow, D., 2003. Vehicle routing and scheduling with full truckloads. Transp. Sci. 37 (2), 170–182.
Asbach, L., Dorndorf, U., Pesch, E., 2009. Analysis, modeling and solution of the concrete delivery problem. Eur. J. Oper. Res. 193 (3), 820–835.
Bektas, T., 2006. The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega 34 (3), 209–219.
Bortfeldt, A., Hahn, T., Männel, D., Mönch, L., 2015. Hybrid algorithms for the vehicle routing problem with clustered backhauls and 3D loading constraints. Eur. J. Oper. Res. 243, 82–96.
Brachner, M., 2013. Solution methods for combined scheduling and transportation problems. Høgskolen i Molde - Vitenskapelig høgskole i logistikk.
Bredström, D., Rönnqvist, M., 2008. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. Eur. J. Oper. Res. 191 (1), 19–31.
Brucker, P., 2007. Scheduling algorithms. Springer.
Caris, A., Janssens, G., 2009. A local search heuristic for the pre- and end-haulage of intermodal container terminals. Comput. Oper. Res. 36 (10), 2763–2772.
Cordeau, J.-F., Laporte, G., Mercier, A., 2001. A unified tabu search heuristic for vehicle routing problems with time windows. J. Oper. Res. Soc. 52 (8), 928–936.
Cordeau, J.-F., Laporte, G., Mercier, A., 2004. Improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows. J. Oper. Res. Soc. 55 (5), 542–546.
Currie, R., Salhi, S., 2004. A tabu search heuristic for a full-Load, multi-Terminal, vehicle scheduling problem with backhauling and time windows. J. Math. Modell. Algo. 3 (3), 225–243.
Demir, E., Bektaş, T., Laporte, G., 2012. An adaptive large neighborhood search heuristic for the pollution-Routing problem. Eur. J. Oper. Res. 223, 346–359.
Desaulniers, G., Desrosiers, J., Erdmann, A., Solomon, M., Soumis, F., 2001. VRP with pickup and delivery. In: Toth, P., Vigo, D. (Eds.), The vehicle routing problem. Siam, pp. 225–242.
Desrosiers, J., Laporte, G., Sauve, M., Soumis, F., Taillefer, S., 1988. Vehicle routing with full loads. Comput. Oper. Res. 15, 219–226.
Drexl, M., 2012. Synchronization in vehicle routing - a survey of VRPs with multiple synchronization constraints. Transp. Sci. 46 (3), 297–316.
Drexl, M., 2014. A Generic Heuristic for Vehicle Routing Problems with Multiple Synchronization Constraints. Technical Report. Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany.
Dumas, Y., Desrosiers, J., Soumis, F., 1991. The pickup and delivery problem with time windows. Eur. J. Oper. Res. 54 (1), 7–22.
Ebben, M., van der Heijden, M., van Harten, A., 2005. Dynamic transport scheduling under multiple resource constraints. Eur. J. Oper. Res. 167 (2), 320–335.
El Hachemi, N., El Hallaoui, I., Gendreau, M., Rousseau, L.-M., 2014. Flow-based integer linear programs to solve the weekly log-truck scheduling problem. Ann. Oper. Res. 1–11.
El Hachemi, N., Gendreau, M., Rousseau, L., 2013. A heuristic to solve the synchronized log-truck scheduling problem. Comput. Oper. Res. 40 (3), 666–673.
Flisberg, P., Lidén, B., Rönnqvist, M., 2009. A hybrid method based on linear programming and tabu search for routing of logging trucks. Comput. Oper. Res. 36 (4), 1122–1144.
Goel, A., Meisel, F., 2013. Workforce routing and scheduling for electricity network maintenance with downtime minimization. Eur. J. Oper. Res. 231, 210–228.
Grangier, P., Gendreau, M., Lehuédé, F., Rousseau, L.-M., 2016. An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. Eur. J. Oper. Res. 254 (1), 80–91.
Grimault, A., Lehuédé, F., Bostel, N., 2014. A two-phase heuristic for full truckload routing and scheduling with split delivery and resource synchronization in public works. In: 2014 International Conference on Logistics Operations Management, pp. 57–61.
Gronalt, M., Hartl, R., Reimann, M., 2003. New savings based algorithms for time constrained pickup and delivery of full truckloads. Eur. J. Oper. Res. 151 (3), 520–535.
Gronalt, M., Hirsch, P., 2007. Log-truck scheduling with a tabu search strategy. In: Doerner, K., Gendreau, M., Greistorfer, P., Gutjahr, W., Hartl, R., Reimann, M. (Eds.), Metaheuristics. Springer US, pp. 65–88.
Hemmelmayr, V., Cordeau, J.-F., Crainic, T., 2012. An adaptive large neighborhood search heuristic for two-Echelon vehicle routing problems arising in city logistics. Comput. Oper. Res. 39 (12), 3215–3228.
Hempsch, C., Irnich, S., 2008. Vehicle routing problems with inter-tour resource constraints. In: Golden, B., Raghavan, S., Wasil, E. (Eds.), The Vehicle Routing Problem: Latest Advances and New Challenges SE - 19. In: Operations Research/Computer Science Interfaces, vol. 43. Springer US, pp. 421–444.
Hirsch, P., 2011. Minimizing empty truck loads in round timber transport with tabu search strategies. Int. J. Inf. Syst. Supply Chain Manag. 4 (2), 15–41.
Kirkpatrick, S., Gelatt, C., Vecchi, M., 1983. Optimization by simulated annealing. Science 220 (4598), 671–680.
Kolisch, R., 1996. Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. Eur. J. Oper. Res. 90 (2), 320–333.
Kovacs, A., Parragh, S., Doerner, K., Hartl, R., 2012. Adaptive large neighborhood search for service technician routing and scheduling problems. J. Scheduling 15 (5), 579–600.
Lehuédé, F., Masson, R., Parragh, S., Péton, O., Tricoire, F., 2013. A multi-criteria large neighbourhood search for the transportation of disabled people. J. Oper. Res. Soc. 65 (7), 983–1000.
Liu, R., Jiang, Z., Fung, R., Chen, F., Liu, X., 2010. Two-phase heuristic algorithms for full truckloads multi-depot capacitated vehicle routing problem in carrier collaboration. Comput. Oper. Res. 37 (5), 950–959.
Liu, Z., Zhang, Y., Li, M., 2014. Integrated scheduling of ready-mixed concrete production and delivery. Autom. Constr. 48, 31–43.
Masson, R., Lehuédé, F., Péton, O., 2012. An adaptive large neighborhood search for the pickup and delivery problem with transfers. Transp. Sci. 47 (3), 344–355.
Masson, R., Lehuédé, F., Péton, O., 2013. Efficient feasibility testing for request insertion in the pickup and delivery problem with transfers. Operations Research Letters 41 (3), 211–215.
Masson, R., Lehuédé, F., Péton, O., 2014. The dial-a-ride problem with transfers. Comput. Oper. Res. 41, 12–23.
Neves-Moreira, F., Amorim, P., GuimarÃ£es, L., Almada-Lobo, B., 2016. A long-haul freight transportation problem: synchronizing resources to deliver requests passing through multiple transshipment locations. Eur. J. Oper. Res. 248 (2), 487–506.
Palmgren, M., Rönnqvist, M., Värbrand, P., 2004. A near-exact method for solving the log-truck scheduling problem. International Transactions in Operational Research 11, 447–464.
Pisinger, D., Ropke, S., 2007. A general heuristic for vehicle routing problems. Comput. Oper. Res. 34 (8), 2403–2435.
Ropke, S., Pisinger, D., 2005. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. 40 (August), 1–30.
Ropke, S., Pisinger, D., 2006. A unified heuristic for a large class of vehicle routing problems with backhauls. Eur. J. Oper. Res. 171 (3), 750–775.
Rubasheuskaya, A., 2012. Combined Scheduling-Transportation Model. Molde University College.
Savelsbergh, M., 1985. Local search in routing problems with time windows. Oper. Res. 4, 285–305.
Savelsbergh, M., 1992. The vehicle routing problem with time windows minimizing route duration. OSRA J. Comput. 4, 146–154.
Schmid, V., Doerner, K., Hartl, R., Salazar-González, J., 2010. Hybridization of very large neighborhood search for ready-mixed concrete delivery problems. Comput. Oper. Res. 37 (3), 559–574.
Schmid, V., Doerner, K., Hartl, R., Savelsbergh, M., Stoecher, W., 2009. A hybrid solution approach for ready-Mixed concrete delivery. Transp. Sci. 43 (1), 70–85.
Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G., 2000. Record breaking optimization results using the ruin and recreate principle. J. Comput. Phys. 159 (2), 139–171.
Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In: Principles and Practice of Constraint Programming CP98 SE - 30. Springer Berlin Heidelberg, pp. 417–431.
Solomon, M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper. Res. 35 (2), 254–265.