An adaptive large neighborhood search heuristic for the pickup and delivery problem with time Windows and scheduled lines

Computers & Operations Research 72 (2016) 12–30

Contents lists available at ScienceDirect

Computers & Operations Research

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

An adaptive large neighborhood search heuristic for the Pickup and Delivery Problem with Time Windows and Scheduled Lines

Veaceslav Ghilas *, Emrah Demir, Tom Van Woensel

School of Industrial Engineering, Operations, Planning, Accounting and Control (OPAC), Eindhoven University of Technology, Eindhoven 5600 MB, The Netherlands

ARTICLE INFO

Available online 4 February 2016

Keywords:
Freight transportation
Pickup and delivery problem
Heuristic algorithm
Scheduled lines

ABSTRACT

The Pickup and Delivery Problem with Time Windows and Scheduled Lines (PDPTW-SL) concerns scheduling a set of vehicles to serve freight requests such that a part of the journey can be carried out on a scheduled public transportation line. Due to the complexity of the problem, which is NP-hard, we propose an Adaptive Large Neighborhood Search (ALNS) heuristic algorithm to solve the PDPTW-SL. Complex aspects such as fixed lines' schedules, synchronization and time-windows constraints are efficiently considered in the proposed algorithm. Results of extensive computational experiments show that the ALNS is highly effective in finding good-quality solutions on the generated PDPTW-SL instances with up to 100 freight requests that reasonably represent real life situations.

1. Introduction

A successful integration of passenger and freight transportation creates a seamless movement of people and freight. This integration achieves socially desirable transport options economically viable in urban areas as it reduces the impact of congestion and air pollution [21]. Actual integration is already being observed in long-haul freight transportation (e.g., passenger aircraft and ferries). Norwegian Hurtigruten carries freight and people efficiently and seamlessly in the region of Northern Europe [18,16]. However, short-haul passenger and freight (i.e., small packages) transportation is rarely integrated, although these services largely use the same infrastructure.

This paper investigates opportunities and the feasibility of using available public transportation vehicles, which operate according to predetermined routes and schedules, for transporting freight. A successful synchronization of pickup and delivery (PD) vehicles with scheduled lines (SLs) is directly related to coordination and consolidation of transport. *Coordination* assures the precision and timing of each leg's movement. On the other hand, *consolidation* implies that the amount of freight transferred to/from SLs fits with capacities of PD and SL vehicles. Considering both coordination and consolidation of such integrated system, we introduce the *Pickup and Delivery Problem with Time Windows and Scheduled Lines* (PDPTW-SL).

There exists a rich literature on PDP-related problems. The interested readers are referred to Berbeglia et al. [3] for a literature survey on the solution methodologies for PDPs. A special case of PDPs where passengers need to be transported from their origin to their destination by considering service level constraints are called dial-a-ride problems (DARP). An overview on the mathematical models and the solution methodologies for single as well as multiple-vehicle DARPs can be found in Cordeau and Laporte [6].

In another extension of the PDP, it is possible to consider transfer opportunity between PD vehicles at predefined transfer nodes (PDP-T). Shang and Cuff [29] proposed an insertion heuristic algorithm to solve instances with up to 167 requests for the PDP-T. Cortes et al. [7] proposed an exact decomposition method, namely Branch-and-Cut (B&C), and solved PDP-T instances with up to six requests. Moreover, Masson et al. [22,23] proposed an ALNS to solve PDP-T and DARP-T, respectively. In both cases, the authors solved instances with up to 100 requests. The PDP with cross-docking opportunities has been studied by Petersen and Røpke [25], who proposed a large neighborhood search heuristic to solve instances with up to 982 requests.

To the best our knowledge, Nash [24] is the first author who conceptually introduced the idea of transporting freight using public transportation. One of the first attempts to combine scheduled line services with DARP was done by Liaw et al. [20]. The authors

* Corresponding author. Tel.: +31 40 247 4984.
E-mail addresses: V.Ghilas@tue.nl (V. Ghilas), E.Demir@tue.nl (E. Demir), T.v.Woensel@tue.nl (T. Van Woensel).

http://dx.doi.org/10.1016/j.cor.2016.01.018
0305-0548/

formulated the problem where transfers to public scheduled transportation are allowed for passengers and wheelchaired persons. The authors proposed two types of heuristics (i.e., online and offline) and solved instances with up to 120 requests. Later, Aldaihani and Dessouky [1] considered an integrated DARP with public transport and proposed a two-stage heuristic algorithm to solve instances with up to 155 requests. Hall et al. [15] introduced a mixed-integer program to solve an integrated dial-a-ride problem (IDARP). The authors considered transfers to fixed lines without modeling schedules of the public transportation. The authors solved small-sized instances with up to 4 requests. Trentini and Malhene [31] presented a review of solutions for combining freight and passenger transportation used in practice. In a related study, Trentini et al. [32] investigated a two-echelon VRP with transshipment in the context of passenger and freight integrated system. The authors proposed an ALNS to solve instances with 50 customers.

As a real-life application of the integrated transport system, a project, called *City Cargo Amsterdam* [5], was held as a pilot experiment in the Netherlands in 2007. Two cargo trams were used to transport freight in the city center of Amsterdam. The goods were transferred from cargo trucks onto trams to be delivered to the inner part of the city. In 2009 the project was quit due to lack of funds.

In this paper, we consider two transport scenarios: (i) an integrated transportation system and (ii) the pickup and delivery problem where transfers to scheduled lines are not allowed. These two scenarios are intended to provide plausible and sufficient reasons to implement such system by comparing them in terms of the total operating costs of the PD vehicles. The PDPTW-SL is NP-hard, which means that its computational complexity exponentially increase when the problem size increases. It is shown that only instances with a limited number of requests can be solved to optimality within a reasonable time limit. Therefore, we have designed a metaheuristic algorithm to generate good-quality routing solutions. In particular, we present an ALNS algorithm which is based on the destroy and re-create principle.

The contribution of the paper is three-fold: (i) to adapt the classical ALNS heuristic algorithm to solve PDPTW-SL, (ii) to efficiently handle synchronization constraints within the proposed ALNS framework and finally (iii) to analyze the cost savings due to the use of scheduled lines in freight transportation. The rest of this paper is structured as follows. In Section 2, we present an arc-based mathematical formulation for the PDPTW-SL. Section 3 describes the proposed ALNS heuristic algorithm for the PDPTW-SL. The experimental results are given in Section 4. Finally, conclusions are provided in Section 5.

2. The Pickup and Delivery Problem with Time Windows and Scheduled Lines

The PDPTW-SL is an extension of the standard PDPTW that additionally considers the flexibility of using available scheduled lines, such as bus, train, and metro. Due to the fact that public transportation systems have a certain coverage (i.e., rural or urban), some delivery trips of the PD vehicles may overlap with SL services. Thus, using public transportation as a part of freight transportation may lead to cost and environmental benefits for the whole transportation system. For instance, due to less driving time of the PD vehicles, logistics service providers may experience substantial operating cost savings. As a consequence, less traveling time of the vehicles also leads to fewer carbon dioxide-equivalent (CO_2e) emissions at the global level for the whole society [9,11]. Furthermore, using scheduled lines for carrying freight gives extra cost benefits for public transport service providers as the utilization of SL services increases. An example application which is based on the current metro system of Amsterdam can be illustrated as in Fig. 1.

The system depicted in Fig. 1 contains four scheduled lines and five transfer nodes (end-of-lines stations). Note that T_1^{51}, T_1^{53}, T_1^{54} as well as T_2^{50} and T_2^{54} can be merged into two transfer nodes as these denote the same locations. Each metro line has one single route with one destination in each direction. To use this metro system as a part of the transportation plan, package(s) can be delivered to one of its end-of-line stations and transported to other side of the line considering available capacity and schedules of the metro line.

2.1. A formal description of the PDPTW-SL

In this section we give a formal description of the PDPTW-SL. All information (e.g., requests, demands, travel times, time windows) is considered known beforehand, and a possible transport plan for the whole planning horizon (e.g., one day) should be generated before the execution of the transport activities. A solution to the problem is a routing plan and schedules for both requests and PD vehicles. We now define the PDPTW-SL on a digraph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ represents the set of nodes (i.e., depots, pickup, delivery and replicated transfer nodes[1]) and $\mathcal{A}$ represents the set of arcs. We note that i, j, r, v, w, and t are used as indices in this paper.

- *Request*: Each request $r \in \mathcal{P}$ is associated with an origin node r and a destination node $r+n$, where n is the number of requests to be satisfied. Each request is associated with two desired time windows: one for the origin ($[l_r, u_r]$), and one for the destination point ($[l_{r+n}, u_{r+n}]$). Furthermore, demand d_r is known for each request.
- *PD vehicle*: A set of PD vehicles is given by $\mathcal{V}$. In addition, each vehicle $v \in \mathcal{V}$ is associated with the carrying capacity Q_v, the depot (origin and destination) o_v, and time window $[l_{o_v}, u_{o_v}]$.
- *Travel and service time*: Travel and service times are known beforehand and remain unchanged during the planning horizon. Each arc $(i, j) \in \mathcal{A}$ is associated with travel time Υ_{ij}. The service time at node $i \in \mathcal{N}$ is represented as s_i.
- *Scheduled line*: A set of all physical transfer nodes is given as $\mathcal{S}$ and the set of all physical scheduled lines is given as $\mathcal{E}$, which is defined by the directed arc between the start and the end of the line (i, j), such that $i, j \in \mathcal{S}$. For example, between two transfer nodes i and j, there are two scheduled lines considered that are one arc for each direction, (i, j) and (j, i). Each scheduled line has a set of indices $\mathcal{K}^{ij}$ for the departure times from i (the start of fixed line), such that the departure time is given as $p_{ij}^w, \forall (i,j) \in \mathcal{E}, w \in \mathcal{K}^{ij}$ (e.g., $p_{T_1,T_2}^0 = 30$). We note that each scheduled line may have different frequencies than other lines, thus the size of the $\mathcal{K}^{ij}$ may differ. Furthermore, it is assumed that SL vehicles are designed to carry a limited number of packages, thus implying a finite carrying capacity $k_{ij}, \forall (i,j) \in \mathcal{E}$.

We state the following additional assumptions related to SL services.

- The package carrying capacity is not influenced by the passenger demand on SL services.

[1] See more detailed explanation in Section 2.2.

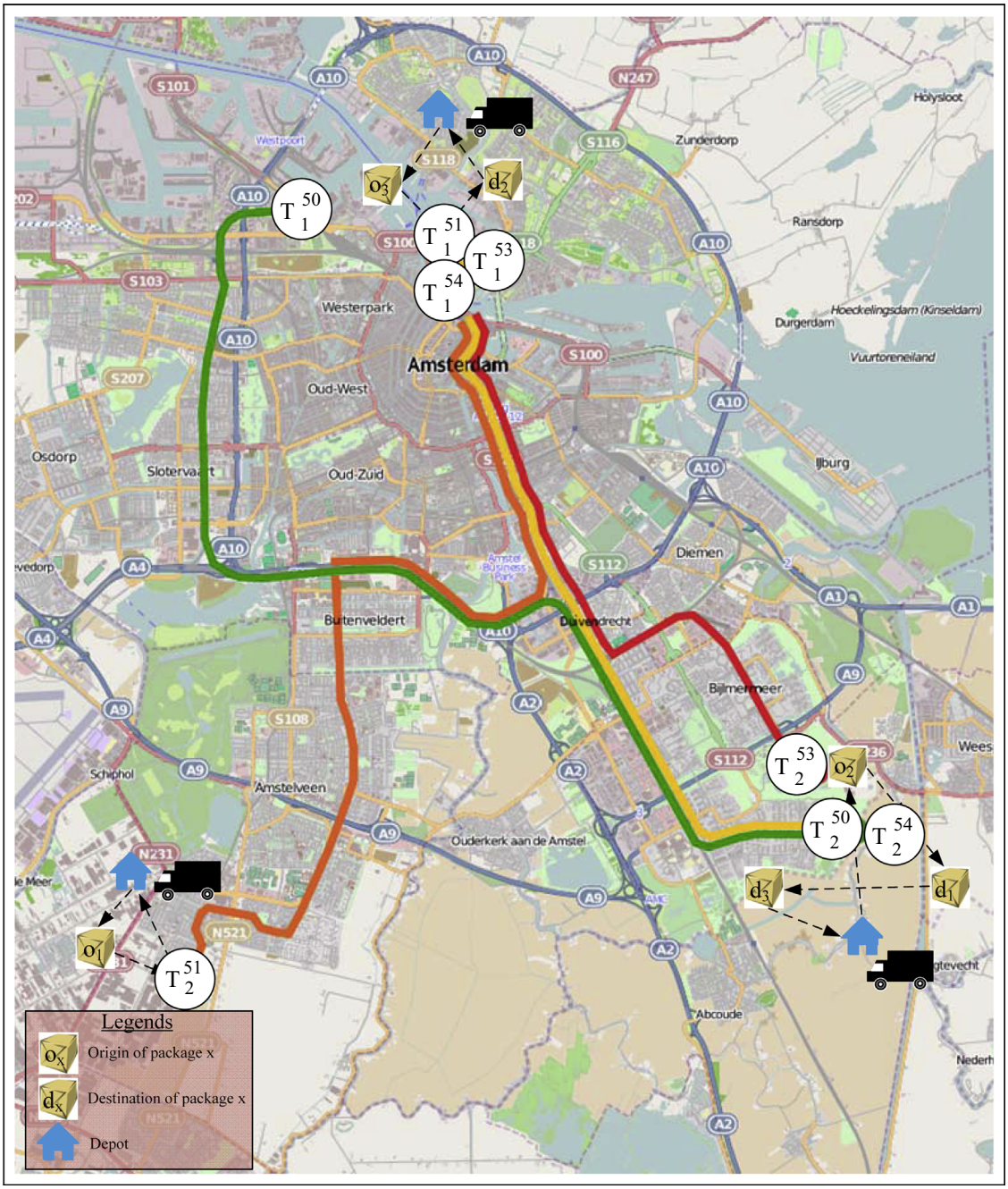

Fig. 1. An illustration of the metro system in Amsterdam (2014).

- Each transfer node is considered as a coordination and a consolidation point for requests (e.g., [12]). In other words, a storage space for to-be-shipped packages is available at the transfer nodes. The packages can be stored until the vehicle's departure time (i.e., scheduled line or PD). Furthermore, it is considered that multiple PD vehicles can be serviced simultaneously at a transfer node. In other words, we assume that the loading/unloading infrastructure is always available for multiple vehicles at a time.
- In case of multiple freight carriers using SL services, it is assumed that each carrier has a limited storage space at the transfer node and on the SL vehicle (e.g., contract-based agreement). Hence, it is assured that the considered capacity is not affected by the demand of other actors involved.
- It is assumed that cost η_{ij} per each unit of parcels shipped on the SL (i, j) includes transportation, handling (transshipment) and storage costs. Handling of the packages during transshipment is assumed to be under the responsibility of the SL service provider (e.g., the driver of the SL vehicle).
- Since the focus of the present problem is related to the operational-level aspects, we disregard all investments needed, such as the re-design of the SL vehicle and the physical storage space required at the transfer nodes. Obviously this assumption may affect the outcome of the system (i.e., might lead to less benefits).
- It is assumed that all data is known beforehand. In practice, this might not be the case because of the uncertainty in travel times and demands. Disregarding these aspects in the planning process might eventually lead to infeasible plans and expensive repair (recourse) actions to tackle such violations.

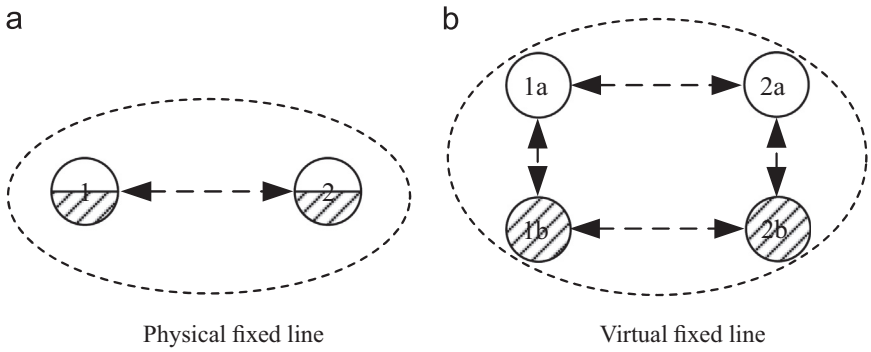

Fig. 2. An illustration of a replicated fixed line.

In order to model waiting times of the PD vehicles and multiple visits at the transfer nodes, each physical scheduled line (i.e., $(i,j) \in \mathcal{E}$, e.g., arcs (1, 2) and (2, 1) in Fig. 2a) is replicated n times as in Hall et al. [15]. In addition, each replication is assigned to one request, and only that specific request can travel on the assigned scheduled line (see Fig. 2b). This is done to reduce the number of decision variables. Therefore, the set of all replicated scheduled lines is given as $\mathcal{F}$ (e.g., $(1a, 2a)$, $(1b, 2b)$, $(2a, 1a)$ and $(2b, 1b)$ in Fig. 2b).

Furthermore, the replication process means that each physical transfer node in $\mathcal{S}$ (e.g., nodes 1 and 2 in Fig. 2a) is replicated n times (e.g., $\mathcal{T} \equiv \{1a,\ 1b,\ 2a,\ 2b\}$ in the considered example). For the sake of modeling let $\psi^t, \forall t \in \mathcal{T}$ be the physical transfer node represented by the replicated transfer node t (e.g., $\psi^{1a}=\psi^{1b}=1$). Consequently, set $\mathcal{T}^t$ can be defined as $\{i \in \mathcal{T} \mid \psi^i = \psi^t, i \neq t\}, \forall t \in \mathcal{T}$ (e.g., in Fig. 2b, $\mathcal{T}^{2a} = \{2b\}$, as $2a$ and $2b$ represent physical transfer node 2).

The proposed model is not limited to only one fixed line (e.g., (1, 2) and (2, 1)). Different topologies may be considered, such as square, triangle, star or any other complex networks. Furthermore, the requests are allowed to be shipped from one scheduled line to the other. The additional parameters used in our model are given in Table 1.

Table 1
Parameters.

Notation	Definition
δ	Number of depots
d_r	Demand of request r
Υ_{ij}	Traveling time from node i to node j
s_i	Service time at node i
k_{ij}	Freight carrying capacity on the fixed line (i, j)
Q_v	Carrying capacity of the vehicle v
$[l_i, u_i]$	Time window of node i
p_{ij}^w	Departure time from transfer node i on the scheduled line (i,j), indexed by w
ψ^t	Physical transfer node that is represented by replicated transfer node t
τ	Number of replicated transfer nodes (i.e., $\lvert\mathcal{T}\rvert$)
f_i^r	$\begin{cases} 1 & \text{if node } i \text{ is } r, \\ 0 & \text{if node } i \in \mathcal{N}_2 \backslash \{r; r+n\} \text{(see Table 2)}, \\ -1 & \text{if node } i \text{ is } r+n. \end{cases}$
θ_v	The routing cost per one time unit of vehicle v
η_{ij}	The cost of shipping one unit of package on the SL (i, j)

Using the parameters above, the notation of sets is given in Table 2.

Table 2
Sets.

Notation	Definition
$\mathcal{V}$	Set of PD vehicles
$\mathcal{O}$	Set of depots, $\mathcal{O} \equiv [1, \ldots, \delta]$(i.e. $o_v \in \mathcal{O}, \forall v \in \mathcal{V}$)
$\mathcal{P}$	Set of requests or pickup nodes, $\mathcal{P} \equiv [\delta+1, \ldots, \delta+n]$
$\mathcal{D}$	Set of delivery nodes, $\mathcal{D} \equiv [\delta+n+1, \ldots, \delta+2n]$
$\mathcal{T}$	Set of replicated transfer nodes, $\mathcal{T} \equiv [\delta+2n+1, \ldots, \delta+2n+\tau]$(see nodes 1a, 1b, 2a and 2b in Fig. 2b)
$\mathcal{T}^t$	Set of replicated transfer nodes associated the same physical transfer node as t (e.g. in Fig. 2b, $\mathcal{T}^{2a} = \{2b\}$, $\mathcal{T}^{1a} = \{1b\}$, etc.)
$\mathcal{N}$	Set of nodes in the graph $\mathcal{G}$; $\mathcal{P} \cup \mathcal{D} \cup \mathcal{O} \cup \mathcal{T} \equiv \mathcal{N}$
$\mathcal{N}_1$	Set of nodes that are related to requests $(\mathcal{P} \cup \mathcal{D})$
$\mathcal{N}_2$	Set of nodes that represent requests and replicated transfer nodes $(\mathcal{P} \cup \mathcal{D} \cup \mathcal{T})$
$\mathcal{E}$	Set of physical SLs which is defined as (i, j), with associated $\mathcal{K}^{ij}$ and k_{ij}
$\mathcal{K}^{ij}$	Set of indices for the departure times from the physical transfer node i on SL $(i,j) \in \mathcal{E}$
$\mathcal{F}$	Set of replicated SLs which is defined as (i, j) with associated $\mathcal{K}^{\psi^i\psi^j}$
$\mathcal{F}^r$	Set of replicated SLs associated with request r (e.g. in Fig. 2, $\mathcal{F}^a = \{(1a,\ 2a),\ (2a,\ 1a)\}$
$\mathcal{F}^t$	Set of replicated SLs connected to the replicated transfer node t (e.g. in Fig. 2, $\mathcal{F}^{1a} = \{(1a, 2a),\ (2a,\ 1a)\}$
$\mathcal{F}^{ij}$	Set of replicated SLs associated with a physical SL $(i,\ j) \in \mathcal{E}$ (e.g. in Fig. 2, $\mathcal{F}^{1,2} = \{(1a,\ 2a),\ (1b,\ 2b)\}$, $\mathcal{F}^{2,1} = \{(2a,\ 1a),\ (2b,\ 1b)\}$
$\mathcal{A}$	Set of arcs in $\mathcal{G}$ defined by $\mathcal{N} \times \mathcal{N}$, (note that $\mathcal{A} \backslash \mathcal{A}^1 \equiv \mathcal{F} \cup \{(i,j) \mid i \in \mathcal{O}, j \in \mathcal{N}_2\} \cup \{(i,j) \mid i \in \mathcal{N}_2, j \in \mathcal{O}\}$
$\mathcal{A}^1$	$= \mathcal{N}_2 \times \mathcal{N}_2 \backslash \mathcal{F}$

The decision variables used to handle routing and scheduling of the PD vehicles, along with the flow and the timing of the requests are given in Table 3.

Table 3
Decision variables.

Notation	Definition
x_{ij}^{v}	A binary variable equal to 1 if arc (i, j) is used by PD vehicle v, 0 otherwise, $\forall\ (i,\ j) \in \mathcal{A}$, $v \in \mathcal{V}$
α_v	A continuous variable which shows the time at which vehicle v returns to its depot, $\forall\ v \in \mathcal{V}$
β_i	A continuous variable which shows the departure time of a vehicle from node i, $\forall\ i \in \mathcal{N}$
y_{ij}^{r}	A binary variable equal to 1 if arc (i, j) is used by request r, 0 otherwise, $\forall i,\ j \in \mathcal{N}_2$, $r \in \mathcal{P}$
γ_i^r	A continuous variable which shows the departure time of request r from node i, $\forall i \in \mathcal{N}_2$, $r \in \mathcal{P}$
q_{ij}^{rw}	A binary variable equal to 1 if replicated fixed line (i, j) is used by request r that departs from i at time p_{ij}^{w}, 0 otherwise, $\forall r \in \mathcal{P}$, $(i,\ j) \in \mathcal{F}^r$, $w \in \mathcal{K}^{\psi^i\psi^j}$

2.2. Mathematical formulation of the PDPTW-SL

The PDPTW-SL can be formalized as the following mixed-integer program:

$$\text{Minimize} \quad \sum_{(i,j)\in\mathcal{A}}\sum_{v\in\mathcal{V}}\theta_v \Upsilon_{ij} x_{ij}^{v} + \sum_{r\in\mathcal{P}}\sum_{(i,j)\in\mathcal{F}^r}\sum_{w\in\mathcal{K}^{\psi^i\psi^j}} \eta_{ij} d_r q_{ij}^{rw} \tag{1}$$

Term (1) minimizes the total travel cost of the PD vehicles and the cost of using SLs for transferred requests:

subject to

Routing and flow constraints

$$\sum_{i\in\mathcal{N}}\sum_{v\in\mathcal{V}} x_{ij}^{v} = 1 \quad \forall j \in \mathcal{N}_1 \tag{2}$$

$$\sum_{i\in\mathcal{N}_2} x_{o_v,i}^{v} \le 1 \quad \forall v \in \mathcal{V} \tag{3}$$

$$\sum_{i\in\mathcal{N}}\sum_{v\in\mathcal{V}} x_{it}^{v} \le 1 \quad \forall t \in \mathcal{T} \tag{4}$$

$$\sum_{j\in\mathcal{N}} x_{ij}^{v} - \sum_{j\in\mathcal{N}} x_{ji}^{v} = 0 \quad \forall i \in \mathcal{N},\ v \in \mathcal{V} \tag{5}$$

$$\sum_{j\in\mathcal{N}_2} y_{ij}^{r} - \sum_{j\in\mathcal{N}_2} y_{ji}^{r} = f_i^r \quad \forall r \in \mathcal{P},\ i \in \mathcal{N}_2 \tag{6}$$

$$\sum_{i\in\mathcal{N}}\sum_{v\in\mathcal{V}} x_{it}^{v} \le \sum_{r\in\mathcal{P}}\sum_{(i,j)\in\mathcal{F}^t} y_{ij}^{r} \quad \forall t \in \mathcal{T} \tag{7}$$

Constraints (2) assure that all pickup and delivery nodes are visited exactly once. Constraints (3) ensure that each vehicle leaves its depot at most once and Constraints (4) assure that each replicated transfer node is visited at most once. Flow conservation for PD vehicles is considered in Constraints (5). Constraints (6) assure flow conservation for the paths of each request. Constraints (7) ensure that if a request uses a scheduled line, a PD vehicle should pick it up/drop it off at a transfer node related to that specific SL:

Scheduling constraints

$$y_{ij}^{r} = 1 \Rightarrow \gamma_j^r \ge \gamma_i^r + \Upsilon_{ij} + s_j \quad \forall r \in \mathcal{P},\ i,j \in \mathcal{N}_2 \tag{8}$$

$$\sum_{v\in\mathcal{V}} x_{ij}^{v} = 1 \Rightarrow \beta_j \ge \beta_i + \Upsilon_{ij} + s_j \quad \forall i \in \mathcal{N},\ j \in \mathcal{N}_2 \tag{9}$$

$$x_{i,o_v}^{v} = 1 \Rightarrow \alpha_v \ge \beta_i + \Upsilon_{i,o_v} + s_{o_v} \quad \forall i \in \mathcal{N}_2,\ v \in \mathcal{V} \tag{10}$$

$$\beta_{r+n} \ge \beta_r + \Upsilon_{r,r+n} + s_{r+n} \quad \forall r \in \mathcal{P} \tag{11}$$

$$l_i \le \beta_i - s_i \le u_i \quad \forall i \in \mathcal{N}_1 \tag{12}$$

$$l_{g_v} \le \alpha_v \le u_{o_v} \quad \forall v \in \mathcal{V} \tag{13}$$

$$\sum_{w\in\mathcal{K}^{\psi^i\psi^j}} q_{ij}^{rw} = y_{ij}^{r} \quad \forall r \in \mathcal{P},\ (i,j) \in \mathcal{F}^r \tag{14}$$

$$q_{ij}^{rw} = 1 \quad \text{and} \quad y_{ij}^{r} = 1 \Rightarrow \gamma_i^r = p_{ij}^{w} \quad \forall r \in \mathcal{P}, (i,j) \in \mathcal{F}^r,\ w \in \mathcal{K}^{\psi^i\psi^j} \tag{15}$$

$$\sum_{r \in \mathcal{P}} \sum_{(a,b) \in \mathcal{F}^{ij}} d_r q_{ab}^{rw} \leq k_{ij} \quad \forall (i,j) \in \mathcal{E},\ w \in \mathcal{K}^{ij} \tag{16}$$

$$\sum_{r \in \mathcal{P}} d_r y_{ij}^r \leq \sum_{v \in \mathcal{V}} Q_v x_{ij}^v \quad \forall (i,j) \in \mathcal{A}^1 \tag{17}$$

Timing for each request is considered in Constraints (8). Similarly for PD vehicles, scheduling is updated in Constraints (9) and (10). Constraints (11) assure precedence relationship of each request. Constraints (12) and (13) force the time windows to be respected. Constraints (14) and (15) assure that if a request uses a scheduled line, it departs at a scheduled departure time. Constraints (16) ensure that package carrying capacity on the scheduled line is not exceeded. Constraints (17) consider the capacity of each PD vehicle:

Synchronization constraints

$$\sum_{j \in \mathcal{N}_1} y_{ij}^r = 1 \Rightarrow \gamma_i^r = \beta_i \quad \forall r \in \mathcal{P},\ i \in \mathcal{T} \tag{18}$$

$$\sum_{j \in \mathcal{N}_2} y_{ij}^r = 1 \Rightarrow \gamma_i^r = \beta_i \quad \forall r \in \mathcal{P},\ i \in \mathcal{N}_1 \tag{19}$$

$$\sum_{i \in \mathcal{N}_2} y_{i,r+n}^r = 1 \Rightarrow \gamma_{r+n}^r = \beta_{r+n} \quad \forall r \in \mathcal{P} \tag{20}$$

$$y_{ij}^r = 1 \Rightarrow \gamma_t^r = \beta_t \quad \forall r \in \mathcal{P},\ t \in \mathcal{T},\ j \in \mathcal{T}^t \tag{21}$$

The set of constraints (18)–(21) ensure the synchronization between requests' and vehicles' schedules. Constraints (18) force departure times of requests and vehicles from a replicated transfer node to be equal if there is a request flow from that node towards a pickup/delivery node. Constraints (19) force departure times of requests and vehicles from a pickup/delivery node to be equal if there is a request flow from that node. Constraints (20) force arrival time at the destination node of a request be equal to departure time of a vehicle from that node. Constraints (21) assure time synchronization between vehicles and requests at each transfer node, with regards to flow between replications of the same original transfer node:

Decision variable domains

$$x_{ij}^v \in \{0,1\} \quad \forall (i,j) \in \mathcal{A},\ v \in \mathcal{V} \tag{22}$$

$$y_{ij}^r \in \{0,1\} \quad \forall i,j \in \mathcal{N}_2,\ r \in \mathcal{P} \tag{23}$$

$$\alpha_v \in R^+ \quad \forall v \in \mathcal{V} \tag{24}$$

$$\gamma_i^r \in R^+ \quad \forall i \in \mathcal{N}_2,\ r \in \mathcal{P} \tag{25}$$

$$\beta_i \in R^+ \quad \forall i \in \mathcal{N} \tag{26}$$

$$q_{ij}^{rw} \in \{0,1\} \quad \forall r \in \mathcal{P}, (i,j) \in \mathcal{F}^r,\ w \in \mathcal{K}^{\psi^i \psi^j} \tag{27}$$

Note that Constraints (8)–(10), (15) and (18)–(21) are formulated as implications, and standard linearization techniques can be used to express them using one or two linear inequalities.

3. An adaptive large neighborhood search heuristic algorithm for the PDPTW-SL

The proposed metaheuristic (ALNS) is an extension of the Large Neighborhood Search (LNS) heuristic, which is first introduced by Shaw [30]. The ALNS metaheuristic was first used to solve several vehicle routing problems by Røpke and Pisinger [28], Pisinger and Røpke [26]. Unlike LNS, where one large neighborhood is usually used, the ALNS applies several neighborhood operators. The neighborhood of a given set of feasible routes is investigated by removing some requests and reinserting them back to the solution. The operators that performed well in previous iterations, are given higher chances to being chosen than the ones that proved poor performance. Thus, each operator is assigned a *probability* of being selected. The newly generated solution is accepted if the criteria defined by the master search framework is satisfied.

3.1. Initialization stage

A greedy insertion heuristic is used to obtain a feasible initial solution to the PDPTW-SL. All requests are initially stored in a list $\mathcal{L}$ and inserted one by one in a random order in their best available positions (for more detail on greedy insertion (GI) – see Section 3.4). The feasibility with regard to the capacity of PD and SL vehicles and time windows is always maintained.

3.2. Probability adjustment procedure

A roulette-wheel mechanism is used to control the selection of the neighborhood operators. At the beginning, the probability of removal and insertion operators are uniformly assigned. As in the implementation of Røpke and Pisinger [28], every N_w iterations, each operator is given a score π_i. The *probability* is updated as $P_d^{t+1} = P_d^t(1-r_p) + r_p \pi_i / \omega_i$, where r_p is the adjustment parameter, π_i is the accumulated score of operator i and ω_i

shows the amount of times the operator was used. The score of the operators is increased by σ_1 if a new best solution is obtained. The score is increased by σ_2 if the newly generated solution outperforms the current solution. Finally, if a deteriorated solution is accepted, the score is increased by σ_3.

3.3. Removal stage

Ten removal operators are used in our ALNS heuristic algorithm. All operators are adapted from or inspired by Shaw [30], Røpke and Pisinger [28] and Demir et al. [8]. The removal stage mainly consists of removing ϕ requests from the current solution and adding them to the so-called *removal list* $\mathcal{L}$ as shown in Algorithm 1. The algorithm starts with a given feasible routing solution X and returns a solution where some requests are disregarded. A chosen operator is used to remove a set of requests (i.e., pickup, delivery and transfer nodes if used) from the solution. The parameter ϕ defines the number of requests to be removed. Removed pickup and delivery nodes are then inserted into list $\mathcal{L}$.

Algorithm 1. The overall structure of the removal operators.

input: A feasible solution X and the number of requests to be removed ϕ
output: A partially destroyed solution X_p
1 *Initialize removal list* ($\mathcal{L} \leftarrow \varnothing$)
2 **for** ϕ *iterations* **do**
3 | *Apply removal operator to remove a request r* (*includes two nodes*; *pickup and delivery*)
4 | $\mathcal{L} \leftarrow \mathcal{L} \cup r$

The removal operators used in our implementation are introduced below:

- *Random removal* (*RR*): This operator randomly removes ϕ requests from the solution, and runs for $\phi \in [\underline{\phi}, \overline{\phi}]$ iterations, where $\underline{\phi}$ and $\overline{\phi}$ are the lower and the upper limits on the number of requests to be removed and ϕ is a random integer number within the specified range.
- *Route removal* (*ROR*): This operator removes a complete route from a given solution. The route that is to be removed is randomly selected. The ROR operator then iteratively selects a node j from the selected route until it is emptied. The corresponding node of j, i.e., its pickup or delivery node, is removed irrespective of which route it is positioned in.
- *Late-arrival removal* (*LAR*): For every request r, this operator calculates the difference between the service start time and l_r, l_{r+n} respectively, and then removes the request with the largest total difference (i.e., $r^* = \arg\max_{r \in \mathcal{P}}\{|[\beta_r - s_r - l_r] + [\beta_{r+n} - s_{r+n} - l_{r+n}]|\}$, where β_x is the departure time from node x). Similarly to RR, this operator runs for $\phi \in [\underline{\phi}, \overline{\phi}]$ iterations.
- *Worst-distance removal* (*WDR*): This operator repeatedly removes costly requests. The cost is considered to be the total distance from the pickup and delivery nodes of a request to the prior and succeeding nodes within given routes, i.e., it removes node $r^* = \arg\max_{r \in \mathcal{P}}\{|\Upsilon_{i-1,r} + \Upsilon_{r,i+1} + \Upsilon_{j-1,r+n} + \Upsilon_{r+n,j+1}|\}$, where $i-1$ and $j-1$, are predecessors and $i+1$ and $j+1$ successors of the pickup and delivery nodes, respectively.
- *Shaw removal* (*SR*): The objective of the SR operator is to remove requests that are similar in terms of certain aspects. The algorithm randomly selects a request r_1 and adds it to $\mathcal{L}$. Let $l_{r_1,r_2} = -1$ if two nodes, one related to r_1 (i.e., r_1 or r_1+n) and another to r_2 (i.e., r_2 or r_2+n) are in the same vehicle route, and 1 otherwise. This operator picks the request $r^* = \arg\min_{r_2 \in \mathcal{P}}\{\Pi_1[\Upsilon_{r_1,r_2} + \Upsilon_{r_1+\ n,\, r_2+n}] + \Pi_2[|\beta_{r_1} - \beta_{r_2}| + |\beta_{r_1+n} - \beta_{r_2+n}|] + \Pi_3 l_{r_1,r_2} + \Pi_4 |d_{r_1} - d_{r_2}|\}$, where Π_1–Π_4 are normalized weights. The operator is applied $\phi \in [\underline{\phi}, \overline{\phi}]$ times by selecting a request, which is not yet added in $\mathcal{L}$ and which is the most similar to the last added request.
- *Proximity-based removal* (*PR*): This operator removes a set of requests that are similar in terms of distance. In other words, it is a special case of SR operator with $\Pi_1 = 1$, and $\Pi_2 = \Pi_3 = \Pi_4 = 0$.
- *Demand-based removal* (*DR*): This operator is a special case of SR with $\Pi_1 = \Pi_2 = \Pi_3 = 0$, and $\Pi_4 = 1$.
- *Time-based removal* (*TR*): This operator is a special case of SR with $\Pi_1 = \Pi_3 = \Pi_4 = 0$, and $\Pi_2 = 1$.
- *Historical knowledge removal* (*HR*): This operator stores the position cost of every request r. The cost is assumed to be the total distance from the origin and destination nodes of request r to the prior and succeeding nodes within the routes, and determined as $c_r = \Upsilon_{i-1,r} + \Upsilon_{r,i+1} + \Upsilon_{j-1,r+n} + \Upsilon_{r+n,j+1}$ at every iteration of the algorithm. Note that $i-1$ and $j-1$ are the preceding nodes of the origin and, respectively, destination nodes of r and $i+1$ and $j+1$ are their successors in the corresponding PD-vehicle routes. The best position cost c_r^* of request r is set to be the minimum of all values found so far. The HR operator then selects the request r^* with the largest cost difference from its best position cost, i.e., $r^* = \arg\max_{r \in \mathcal{P}}\{c_r - c_r^*\}$. Request r^* is added to $\mathcal{L}$. This operator iterates $\phi \in [\underline{\phi}, \overline{\phi}]$.
- *Worst removal* (*WR*): This operator removes $\phi \in [\underline{\phi}, \overline{\phi}]$ requests with the highest cost. The cost in this case is computed as follows: given a solution, the cost of a request r is the difference in the objective function between the current solution (with r) and the same solution without serving r. Note that the difference between WR and WDR is that WR uses the total cost (including transfer cost) of the requests as objective, whereas WDR focuses only on the distance. In addition, WR also takes into account the length of the newly introduced arcs $(i-1, i+1)$ and $(j-1, j+1)$ when removing nodes i and j from a route.

3.4. Insertion stage

We have used 5 insertion operators in the proposed ALNS heuristic algorithm. With the help of insertion operators, the removed requests in $\mathcal{L}$ can be inserted into the existing routes, if possible. The general schematic overview of an insertion procedure is shown in Algorithm 2. It is noteworthy that in Line 5 of the insertion algorithm, we do not consider every possible insertion. In order to avoid redundant computations in *repairTransfers*(X, r, T^r) and *feasibleSchedule*(X) due to time windows, a preliminary time window check is applied as described in Braekers et al. [4].

Due to the fact that PD-vehicle capacity constraints can be fully verified only after applying *repairTransfers*(X, r, T^r) procedure, capacity violations for the routes with no transferable requests can be easily detected. Note that *feasibleSchedule*(X) and *feasibleCapacity*(X, $c(X)$, η_{ij}) (see Line 11 in Algorithm 2) methods are described in Section 3.5.

Algorithm 2. The generic structure of an *insertion* i^* (X^*_{new}, $\mathcal{L}$, $X_{current}$) procedure.

input: A partially destroyed current solution X^*_{new}, a list of removed requests $\mathcal{L}$, and current solution $X_{current}$
output: A feasible solution obtained after the insertion procedure
1 **for** (*each request r in* $\mathcal{L}$) **do**
2 $X_{new} \leftarrow NULL$
3 $c(X^*_{new}) \leftarrow +\infty$
4 **for** (*each route Q and route M*) **do**
5 **for** (*each position q within a route Q and each position m within a route M*) **do**
6 *Insert r and r+n in positions q and m, respectively, in solution* X^*_{new}
7 **if** ($Q \neq M$) **then**
8 $(X_t, t^*_r, t^*_{r+n}) \leftarrow repairTransfers(X^*_{new}, r, T^r)$
9 **else**
10 $X_t \leftarrow X^*_{new}$
11 **if** (*feasibleCapacity*(X_t, $c(X_t)$, η_{ij}) *and feasibleSchedule*(X_t)) **then**
12 **if** (acceptance criteria of i^* is satisfied) **then**
13 $X_{new} \leftarrow X_t$
14
15 **if** ($X_{new} \neq NULL$) **then**
16 $X^*_{new} \leftarrow X_{new}$
17 **else**
return $X_{current}$
18 *return* X^*_{new}

The feasibility with respect to the capacity, SL schedules and time windows is always kept. Note that checking the feasibility is not trivial, since pickup and delivery nodes of the same request can be located in different PD routes, and hence, a tour must include at least one scheduled line. Such partial solutions need to be repaired by adding transfer nodes. Therefore, *repairTransfers*(X, r, T^r) procedure is run to insert corresponding replicated transfer nodes in a greedy fashion. As proposed for the MIP formulation, each original transfer node is replicated n (number of requests) times, hence each request gets assigned a list of replicated transfer nodes. The overview of the repair mechanism is shown in Algorithm 3.

Algorithm 3. The generic structure of the *repairTransfers*(X, r, T^r) procedure.

input: A partial solution X, a request r and T^r replicated transfer nodes related to r
output: A feasible solution X_t and a origin and a destination transfer nodes (t^*_r, t^*_{r+n}) of r in the current solution X_t
1 $Q \leftarrow$ *the route related to r*
2 $M \leftarrow$ *the route related to r+n*
3 $t^*_r \leftarrow$ *transfer node* $\in T^r$ *with the cheapest insertion in route Q, sequenced after r*
4 $T^r = T^r \setminus t^*_r$
5 $t^*_{r+n} \leftarrow$ *transfer node* $\in T^r$ *with the cheapest insertion in route M, sequenced earlier than r+n*
6 $X_t \leftarrow$ *updated X given minimal cost insertions of* t^*_r *and* t^*_{r+n}
7 *return* (X_t, t^*_r, t^*_{r+n})

A transferable request r is assigned two related replicated transfer nodes: t^*_r (i.e., origin transfer node), and t^*_{r+n} (i.e., destination transfer node).

We now describe the insertion operators. We note that the requests are randomly chosen from the removal list $\mathcal{L}$.

- *Greedy insertion* (*GI*): This operator iteratively inserts a request (both pickup and delivery nodes) in the best possible position of the routes. $\Delta^r_{I_iJ_j}$ is the objective function value when the pickup node of r is inserted in route I (position i) and its corresponding delivery node is inserted in route J (position j). Thus, $\Delta^r_{I_iJ_j}* = \arg\min\{\Delta^r_{I_iJ_j}\}$.
- *Second-best insertion* (*SI*): This new operator chooses the second best insertion for randomly selected unassigned request. The main idea of using this operator is to diversify the search.
- *Greedy insertion and noise function* (*GIN*): This operator is similar to the GI and involves certain flexibility in selecting the best insertion of a request. This flexibility is obtained by changing the cost for request r: *Updated Cost* = *Actual Cost* $+ \overline{d}\mu\epsilon$, where $\overline{d}$ is the largest distance between nodes, μ is a noise parameter used for diversification and is set equal to 0.1, and ϵ is a random number between $[-1,1]$. *Updated Cost* is computed for each request in $\mathcal{L}$.
- *Second-best insertion and noise function* (*SIN*): This operator is similar to the SI and involves the same noise function that is used in GIN operator.

- *Best out of λ feasible insertions* (*λFI*): This new operator is similar to GI but chooses the best insertion out of the first λ feasible insertions. The parameter λ is a randomly generated integer number between 1 and ψ. For each unassigned request we generate a set of possible insertions associated with corresponding distance-based costs (disregarding distance to the corresponding transfer nodes), while considering precedence constraints. These positions are not necessary feasible in terms of time windows. Additionally, in order to filter out some more of the infeasible insertions, the time windows of the subsequent nodes visited within the considered routes are verified. The generated set is sorted in increasing order based on insertion costs. Hence the operator investigates the cheapest insertions overall routes first.

Moreover, additional five variants of these operators are also used, where the requests are sorted in $\mathcal{L}$ in terms of their time flexibility (i.e., the least flexible first). The flexibility of request r can be computed as $|u_{r+n}-l_r|$.

3.5. Feasibility of the routes and schedules

Scheduling (time windows) constraints bring complexity to the PDPTW-SL since the requests may be picked up by a PD vehicle and delivered by another one. This implies that PD vehicles and SLs must be synchronized. This problem can be handled as shown in Algorithm 4.

Algorithm 4. The generic structure of the *feasibleSchedule*(X) procedure

input: A solution X
output: Boolean value: *TRUE* if feasible, *FALSE* if infeasible
1 $list \leftarrow \varnothing$
2 $\beta_{t_r^*} \leftarrow 0,\ \beta_{t_{r+n}^*} \leftarrow 0,\ \forall\ r \in \mathcal{P}$
3 **for** (*each route I in X*) **do**
4 **for** (*each node i in I*) **do**
5 $\beta_i \leftarrow \max\{\beta_{p_i} + \Upsilon_{p_i,i} + s_i;\ l_i + s_i\}$
6 **if** ($\beta_i > u_i + s_i$) **then**
7 *return FALSE*
8 **if** ($i = t_{r+n}^*,\ \forall r \in \mathcal{P}$) **then**
9 $list \leftarrow list \cup i$
10 $i \leftarrow$ *end route*
11 **while** ($list \neq \varnothing$) **do**
12 **for** (t_{r+n}^* *in list*) **do**
13 **if** ($\beta_{t_r^*} > 0$) **then**
14 $path_r \leftarrow generatePath(t_r^*, t_{r+n}^*)$
15 $\beta_{t_{r+n}^*} \leftarrow determineTime(\beta_{t_r^*}, path_r)$
16 **for** (*each successor node i of* t_{r+n}^*) **do**
17 $\beta_i \leftarrow \max\{\beta_{p_i} + \Upsilon_{p_i,i} + s_i;\ l_i + s_i\}$
18 **if** ($\beta_i > u_i + s_i$)**then**
19 *return FALSE*
20 **if** ($i = t_{r_1+n}^*,\ \forall r_1 \in \mathcal{P}$)**then**
21 $list \leftarrow list \cup i$
22 $i \leftarrow$ *end route*
23
24 $list \leftarrow list \backslash t_{r+n}^*$
25 **if** ($\beta_{t_r^*} = 0,\ \forall\ \beta_{t_{r+n}^*} \in$ *list*) **then**
 return FALSE
26 *return TRUE*

All β values for the transfer nodes are reset to 0 in Line 2. The first block of the algorithm (Lines 3–10) sets the time for all nodes at the earliest possible value (depending on the start of the time window and the departure time from the preceding node p_i). The algorithm switches to the next route whenever a destination transfer node is reached. This node is added to a *list* that contains all destination transfer nodes which do not have an updated synchronized time. The second part of the algorithm (Lines 11–25) runs until the *list* of destination transfer nodes is emptied or a cycle (see Fig. 4) is found.

- For each $t_{r+n}^* \in list$ with the corresponding origin transfer node (i.e., t_r^*) that has already received a timing, generate a shortest path [13] from t_r^* to t_{r+n}^* (Line 14). Given the shortest path and the departure time from t_r^*, the algorithm computes the time at t_{r+n}^* by considering earliest scheduled departure time later than $\beta_{t_r^*}$ (i.e., $\lceil \beta_{t_r^*} \rceil$) for every intermediate line used (Line 15). Considered t_{r+n}^* with a synchronized timing is removed from the *list*. Finally, all succeeding nodes of t_{r+n}^* get assigned a value until a destination transfer node is reached and added to the *list*.

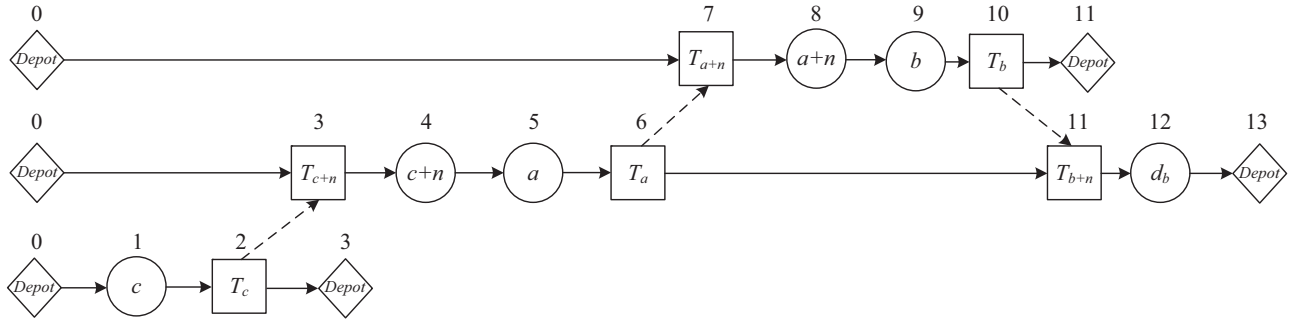

Fig. 3. An interdependency relation of the routes.

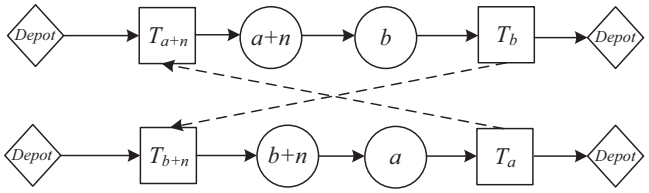

Fig. 4. An example of a two-request cycle.

- Whenever all $\beta_{t_r^*}$ is zero $\forall t_{r+n}^* \in list$, the algorithm finds a cycle (see Fig. 4) and stops. Hence, no feasible schedule is possible for a given solution X.

For a better understanding, we refer to the example shown in Fig. 3. Three requests, namely a, b, and c, are to be delivered to destination nodes $a+n$, $b+n$, and $c+n$, respectively. All the requests are transferable, thus each request is shipped on a scheduled line. All these transfer nodes are replications of the original transfer nodes and each replication is assigned to only one request. In the present example, T_a represents the origin transfer node of a, and T_{a+n} is its destination transfer node. For the sake of simplicity, each arc has one time unit and each node does not require any service time. The numbers on top of the nodes indicate the departure time from that specific node. In this example, three PD vehicles are needed for the transportation of these three requests.

By referring to Fig. 3, the feasibility of the schedule is checked as follows: the process starts in the first route. The algorithm reaches the transfer node T_{a+n} that is a destination transfer node of request a. The algorithm moves to the second route and reaches T_{c+n}, which is a destination transfer node of request c. Finally, it moves to the last route and updates the timing for the whole route as no destination transfer node is encountered. The *list* contains two destination transfer nodes, i.e., T_{a+n} and T_{c+n}. A shortest path is generated from T_c to T_{c+n}, and the departure time $\beta_{T_{c+n}}$ is updated (i.e., 3). Afterwards, the subsequent nodes of T_{c+n} for a given synchronized time $\beta_{T_{c+n}}$ get assigned time values until T_{b+n} is reached. Similarly, the timing of T_{a+n} is updated and followed by T_{b+n}.

Note that a cycle implies that the precedence constraints are violated for at least two transferable requests, since such requests need to be picked up and dropped off twice (see Fig. 4). In addition, cycles may be composed of multiple routes and requests.

The procedure *determineTime*($\beta_{t_r^*}, path_r$) (see Algorithm 4, Line 15) computes departure time from t_{r+n}^* for a given departure time from t_r^*.

The capacity constraints of both PD and SL vehicles are verified in Algorithm 5. First, the algorithm checks SL capacity by first generating scheduled line paths along with every scheduled departure time from the considered transfer nodes (Line 3). The capacity variables are updated for each SL and each possible scheduled departure. The cost of the solution is updated in Line 6. Whenever the capacity is violated (Line 7), the algorithm stops and returns *FALSE* value. The PD-vehicle capacity constraints can be checked in a standard way with some preprocessing (Line 9). In particular, the destination transfer node of a transferable request r (i.e., t_{r+n}^*) becomes an origin node on a different route, hence it will have a positive demand (i.e., d_r) whereas the origin transfer node of r (i.e., t_r^*) becomes a destination node and it is assigned a negative demand (i.e., $-d_r$).

Algorithm 5. The generic structure of *feasibleCapacity*(X, c(X), η_{ij}) procedure.

input: A solution X, routing cost of X, and cost of using SL η_{ij} per unit shipped on (i,j)
output: A boolean value θ
1 *Initialize* $m_{ij}{}^w$ *array for the capacity used on the scheduled line* (i,j) *at time* $p_{ij}{}^w$ *and* $\overline{\mathcal{P}}^t \leftarrow$ *set of transferable requests in solution* X
2 **for** (r *in* $\overline{\mathcal{P}}^t$) **do**
3 *set* $(\overline{\mathcal{K}}^r, \gamma) \leftarrow$ *scheduled lines* (i,j) *used by* r *and corresponding scheduled departure times* γ_{ij}^r
4 **for** $((i,j) \in \overline{\mathcal{K}}^r)$ **do**
5 *set* $m_{ij}^w \leftarrow m_{ij}^w + d_r$
6 *set* $c(X) \leftarrow c(X) + d_r \eta_{ij}$
7 **if** $(m_{ij}^w > k_{ij})$ **then**
8 *return FALSE*
9 **if** (*PDvehicleCapacityViolated*(X)) **then**
10 *return FALSE*
11 *return TRUE*

3.6. Master search framework

In the proposed ALNS, we used simulated annealing (SA). The overall pseudo-code of the proposed algorithm is provided in Algorithm 6.

Algorithm 6. The pseudo-code of the ALNS with SA.

input Removal operators D, insertion operators I, initial temperature T, cooling rate κ
output A feasible solution X_{best}
Generate an initial solution by using the Greedy insertion algorithm
Initialize probability $P_d{}^t$ for each destroy operator $d \in D$ and probability P_i^t for each insertion operator $i \in I$
Let j be the iteration counter with initial value of $j \leftarrow 1$
$X_{current} \leftarrow X_{best} \leftarrow X_{init}$
repeat
  Choose a removal operator $d^* \in D$ with probability P_d^t
  Let X_{new}^* be the solution generated after using operator d^* to $X_{current}$
  Choose an insertion operator $i^* \in I$ with probability P_i^t
  Let X_{new} be the newly−generated solution after applying operator i^* to X_{new}^*
  if $c(X_{new}) < c(X_{current})$ **then**
    $X_{current} \leftarrow X_{new}$
    if $c(X_{current}) < c(X_{best})$ **then**
      $X_{best} \leftarrow X_{current}$
  Let $\nu \leftarrow e^{-(c(X_{new}) - c(X_{current}))/T}$
  Generate a random number $\epsilon \in [0, 1]$
  if $\epsilon < \nu$ **then**
    $X_{current} \leftarrow X_{new}$
  $T \leftarrow \kappa T$
  Update probabilities using the adaptive probability adjustment procedure
  $j \leftarrow j + 1$
until *the maximum number of iterations is reached*

The notation X_{best} shows the best solution obtained, $X_{current}$ presents the current solution, and X_{new} shows a newly generated solution that can be accepted or rejected at each iteration. Each solution X is associated with a cost $c(X)$. Any solution X_{new} can have three outcomes: (i) if $c(X_{new}) < c(X_{current})$, then it is accepted, (ii) if $c(X_{new}) > c(X_{current})$ and $e^{-(c(X_{new}) - c(X_{current}))/T}$ is greater than a random number within [0,1], with T – *temperature*, then X_{new} is accepted as well, and finally, (iii) it is rejected. The initial temperature is selected as T. The temperature T diminishes at each iteration by a rate κ, where $0 < \kappa < 1$. The algorithm runs for a given number of iterations (i.e., 10,000 iterations).

4. Computational experiments

To analyze the performance of the proposed ALNS, we run a series of experimental tests and present the obtained results in this section.

4.1. Data generation

Three sets of instances, namely *R*, *C*, and *RC*, with three scheduled lines in a triangular topology and a frequency of one departure of every 30 time units are considered. Each instance contains 100 requests (i.e., 100 pick-up and 100 delivery nodes) over 200×200 time units on an Euclidean space. Instances follow a naming convention of *G_n_sl_v*, where *G* is the geographic distribution of the customers, *n* is the number of requests that needs to be served, *sl* is the number of SLs, and *v* is the number of available PD vehicles. Instances are classified with regard to the location of the requests. For example, *C* involves clustered nodes around transfer nodes, *R* considers uniform-randomly distributed request nodes, and finally *RC* involve randomly clustered nodes. More specifically, *C* and *RC* have the nodes positioned within at most 30, respectively 80 time units to one of its three available transfer nodes. In all cases, two depots with 30 heterogeneous PD vehicles each are considered.

The planning horizon is set to 10 working hours (i.e., 600 time units). The widths of the time windows are randomly generated between 26 and 91 time units. Service times are considered to be up to three time units. Each demand is assigned between one and three units. The capacity of PD vehicle is generated between six and 20 units. The carrying capacity on the considered SLs is assumed to be 15 demand units. Additional instances were also generated such that subsets of requests, PD vehicles, SLs out of *R*, *C*, and *RC* datasets are used. The datasets (i.e., *R*, *C*, and *RC*) can be found on the web page www.smartlogisticslab.nl.

4.2. Parameter tuning

The proposed algorithm is implemented in NetBeans IDE 7.1.1 using Java. All experiments were conducted on an Intel Core i5 with 2.6 GHz speed and 4 GB RAM. A more detailed tuning was done on the five most sensitive parameters as shown in Table 4. These were identified during some preliminary tests. Three instances of small sizes (i.e., 8, 16 and 25 requests) were solved 10 times each for the given combination of parameters. The rest of the parameters were chosen based on some other experiments and our intuition.

In total, the ALNS algorithm involves 16 parameters which are listed in Table 5.

Table 4
Results of parameter tuning.

Iterations	Destroy rate %	σ_1	σ_2	σ_3	Average value €	Iterations	Destroy rate %	σ_1	σ_2	σ_3	Average value €
10,000	0.15	1	0	1	954.09	15,000	0.15	1	0	1	953.72
10,000	0.15	1	0	5	953.08	15,000	0.15	1	0	5	953.61
10,000	0.15	1	3	1	954.18	15,000	0.15	1	3	1	950.98
10,000	0.15	1	3	5	951.86	15,000	0.15	1	3	5	954.02
10,000	0.15	5	0	1	958.49	15,000	0.15	5	0	1	954.03
10,000	0.15	5	0	5	958.52	15,000	0.15	5	0	5	952.59
10,000	0.15	5	3	1	954.66	15,000	0.15	5	3	1	952.94
10,000	0.15	5	3	5	953.00	15,000	0.15	5	3	5	952.57
10,000	0.25	1	0	1	950.04	15,000	0.25	1	0	1	949.80
10,000	0.25	1	0	5	953.80	15,000	0.25	1	0	5	953.82
10,000	0.25	1	3	1	954.87	15,000	0.25	1	3	1	951.89
10,000	**0.25**	**1**	**3**	**5**	**948.25**	15,000	**0.25**	**1**	**3**	**5**	950.64
10,000	0.25	5	0	1	953.16	15,000	0.25	5	0	1	951.42
10,000	0.25	5	0	5	954.14	15,000	0.25	5	0	5	954.47
10,000	0.25	5	3	1	954.10	15,000	0.25	5	3	1	952.34
10,000	0.25	5	3	5	951.46	15,000	0.25	5	3	5	953.02
10,000	0.35	1	0	1	950.63	15,000	0.35	1	0	1	949.67
10,000	0.35	1	0	5	951.08	15,000	0.35	1	0	5	950.36
10,000	0.35	1	3	1	950.53	15,000	0.35	1	3	1	954.97
10,000	0.35	1	3	5	951.65	15,000	0.35	1	3	5	954.01
10,000	0.35	5	0	1	951.77	15,000	0.35	5	0	1	952.77
10,000	0.35	5	0	5	949.89	15,000	0.35	5	0	5	950.59
10,000	0.35	5	3	1	949.72	15,000	0.35	5	3	1	951.11
10,000	0.35	5	3	5	951.88	15,000	0.35	5	3	5	952.34

Table 5
Used parameters in the computational experiments.

Group	Parameter	Value
I	The total number of iterations (N_i)	10,000
	The number of route-wheel iterations (N_w)	200
	The roulette-wheel constant (r_p)	0.1
	New best solution (σ_1)	1
	Improving solution (σ_2)	3
	Deteriorating solution (σ_3)	5
II	Initial temperature (T)	200
	Cooling degree (κ)	0.9995
	Lower limit of removable requests ($\underline{\phi}$)	2.5% of $\lvert\mathcal{P}\rvert$
	Upper limit of removable requests ($\overline{\phi}$)	25% of $\lvert\mathcal{P}\rvert$
	First Shaw parameter (Π_1)	0.5
	Second Shaw parameter (Π_2)	0.2
	Third Shaw parameter (Π_3)	0.1
	Fourth Shaw parameter (Π_4)	0.2
	Noise parameter (μ)	0.1
	Number of feasible insertions (ψ)	30

It is noted that a driving cost of the PD vehicles is assumed to be 0.5 €per minute. It seems reasonable considering all operational costs, such as fuel consumption, driver wage, insurance, and tax. The cost of each demand unit of package request shipped on a fixed line is set to 1 €, which includes handling, storage and transportation costs. The parameters used in the ALNS algorithm are grouped into two categories below:

- Group I contains the parameters related to the selection procedure of the operators. The values of σ_1, σ_2 and σ_3 do not satisfy $\sigma_1 \geq \sigma_2 \geq \sigma_3$. In our implementation and similar to Demir et al. [8,10], we have chosen (based on parameter tuning) a parameter setting which rewards more acceptance of a worse solutions than discovering of better ones. It turns out to be an effective way for diversifying the search.
- Group II parameters are used to fine-tune the SA master search and the operators.

Tables 6 and 7 indicate the number of times each operator was used within the ALNS. In addition, these tables show in the parenthesis the total time spent to run each operator. We note that the results are obtained using only one instance of each different size in terms of the number of requests.

The results shown in Table 6 show that operators RR, SR, PR and HR are used almost equally often. In many cases, the most widely used operators are LAR, TR and WR.

As seen in Table 7, most used insertion operators are the ones that have ordered $\mathcal{L}$. Since least flexible requests are inserted first, these operators have higher chances to insert all requests within the existing routes. More specifically, oGI, oSI and their variants with noise

Table 6
Number of iterations and the computational times needed by each removal operator.

Instance	RR	ROR	LAR	WDR	SR	PR	DR	TR	HR	WR
R_25_1_8	986(0.0)	800(0.0)	**1358(0.1)**	766(0.0)	979(0.0)	938(0.0)	900(0.0)	**1121(0.0)**	904(0.0)	**1248(0.0)**
R_50_1_14	946(0.0)	947(0.0)	900(0.0)	951(0.0)	**1051(0.0)**	963(0.0)	826(0.0)	**1241(0.0)**	946(0.0)	**1229(0.0)**
R_75_1_20	995(0.0)	893(0.0)	**1104(0.0)**	943(0.1)	958(0.1)	933(0.1)	769(0.0)	**1196(0.1)**	936(0.1)	**1273(0.0)**
R_100_1_24	1024(0.0)	640(0.0)	**1323(0.1)**	**1294(0.1)**	907(0.1)	926(0.1)	743(0.1)	**1205(0.1)**	1096(0.1)	842(0.1)

Table 7
Number of iterations and the computational times needed by each insertion operator.

Instance	GI	SI	GIN	SIN	λ FI	oGI	oSI	oGIN	oSIN	oλ FI
R_25_1_8	860(0.2)	855(0.2)	537(0.1)	510(0.2)	451(0.0)	**1843(0.9)**	**1945(1.2)**	1024(0.6)	**1045(0.6)**	930(0.3)
R_50_1_14	655(0.7)	631(0.5)	312(0.1)	414(0.4)	453(0.1)	**2359(11.2)**	**2332(10.8)**	**971(4.5)**	941(4.5)	932(1.1)
R_75_1_20	636(1.4)	532(0.9)	404(0.6)	263(0.5)	286(0.0)	**2697(44.9)**	**2643(42.5)**	**1040(13.5)**	832(13.0)	667(1.5)
R_100_1_24	531(4.2)	485(2.1)	212(1.3)	252(1.1)	273(0.2)	**3315(285.0)**	**3095(263.8)**	**875(56.7)**	510(42.5)	452(4.7)

functions are widely selected. It is noted that randomly ordered insertion operators do not perform well, as many times the algorithm cannot insert all unassigned requests back to the solution due to their order of insertion.

Tables 8 and 9 indicate the number of times an operator has found the best and a better solution compared to the current one, respectively. It is noted that the number in parenthesis indicates the number of times a current solution is improved, but not to become a best known.

Table 8
Number of global best solutions found and number of improving solutions achieved by the removal operators.

Instance	RR	ROR	LAR	WDR	SR	PR	DR	TR	HR	WR
R_25_1_8	1(61)	0(47)	1(29)	0(4)	0(77)	0(43)	**2(48)**	**4(97)**	1(60)	**2(152)**
R_50_1_14	**24(179)**	2(111)	**6(205)**	3(68)	4(188)	**5(137)**	1(72)	5(257)	1(56)	1(154)
R_75_1_20	**14(195)**	6(94)	3(208)	1(154)	**7(235)**	6(178)	4(42)	6(290)	**9(189)**	4(168)
R_100_1_24	**17(226)**	**9(73)**	**9(292)**	4(107)	7(228)	7(196)	2(32)	4(306)	8(163)	7(256)

Table 9
Number of global best solutions found and number of improving solutions achieved by the insertion operators.

Instance	GI	SI	GIN	SIN	λ FI	oGI	oSI	oGIN	oSIN	oλ FI
R_25_1_8	1(7)	1(7)	0(3)	0(2)	0(0)	**3(253)**	**4(210)**	**2(73)**	0(63)	0(0)
R_50_1_14	**18(36)**	0(10)	0(3)	0(1)	0(0)	**19(755)**	**15(575)**	0(25)	0(22)	0(0)
R_75_1_20	**8(21)**	0(5)	0(0)	0(2)	0(0)	**28(1069)**	**23(628)**	1(15)	0(13)	0(0)
R_100_1_24	**12(27)**	0(6)	0(1)	0(0)	0(0)	**32(1211)**	**30(630)**	0(2)	0(2)	0(0)

The results indicate that all removal operators, to some extent, contribute to achieving improved solutions. On the other hand, some insertion operators (i.e., λFI and oλFI) do not improve a current solution at any time. However, as it will be shown below, these operators are needed to diversify the search and achieve better overall performance of the algorithm. Furthermore, as expected, greedy operators (i.e., best and second-best) help obtaining improved solutions.

In order to identify the usefulness of the new insertion operators proposed in this paper, we tested four configurations of the ALNS. These are shown in Table 10, along with the average objective function values over 10 runs of the algorithm. We used the same instances as in Table 4.

Table 10
A tuning of insertion operators.

Configuration	Average value €
Without SI and SIN	949.61
Without λFI and oλFI	952.15
Without SI, SIN, λFI and oλFI	956.13
With all operators	**948.25**

According to the obtained results, using SI, SIN and λFI operators leads to the best performance of the algorithm. These operators are mainly destined to diversify the search. As seen in Table 10, it seems imperative to use such operators along with the unconventional scoring setting.

4.3. Results of the proposed algorithm on the PDPTWs

In this section, we provide in Tables 11–13 computational results on the PDPTW benchmark instances (i.e., [19]), which come in three sets: *R*, *C* and *RC* classified with respect to the positioning of the customers (i.e., random, clustered and randomly clustered). The reason for choosing these instances is that Røpke and Cordeau [27] and Baldacci et al. [2] provided their results by using the minimization of operating costs as our algorithm does. Tables 11–13 compare published results to the ones obtained by our ALNS heuristic algorithm. The values of the best known (or even optimal) solutions obtained from [27,2,28] are given in under column "Best known value". Note that the values in bold emphasize that the proposed algorithm found the best known solution. The symbol "*" indicates that the values are not necessary optimal and are obtained by Røpke and Pisinger [28]. Moreover, we note that all figures presented in these tables use a single

Table 11
Results of the proposed algorithm on benchmark PDPTW-*C* instances.

Li and Lim [19] instances	# of requests	Best known value	CPU seconds	ALNS best found	ALNS average value	GAP %	CPU seconds
LC1_2_1	106	2704.6	3	**2704.6**	**2704.6**	0.00	47
LC1_2_2	105	2764.6	22	**2764.6**	2764.8	0.01	51
LC1_2_3	103	2772.2	115	**2772.2**	2779.3	0.26	121
LC1_2_4	105	2661.4*	209	**2661.4**	2684.0	0.84	123
LC1_2_5	107	2702.0	5	**2702.0**	**2702.0**	0.00	40
LC1_2_6	107	2701.0	7	**2701.0**	**2701.0**	0.00	42
LC1_2_7	107	2701.0	8	**2701.0**	**2701.0**	0.00	38
LC1_2_8	105	2689.8	16	**2689.8**	2689.9	0.00	43
LC1_2_9	105	2724.2	55	**2724.2**	2758.1	1.23	63
LC1_2_10	104	2741.6	137	2743.9	2763.8	0.80	67
Average			58			0.31	66

Table 12
Results of the ALNS heuristic on benchmark PDPTW-*RC* instances.

Li and Lim [19] instances	# of requests	Best known value	CPU seconds	ALNS best found	ALNS average value	GAP %	CPU seconds
LRC1_2_1	106	3606.1	3	**3606.1**	3608.9	0.08	45
LRC1_2_2	103	3292.4	322	**3292.4**	3304.3	0.36	67
LRC1_2_3	105	3079.5*	183	3108.5	3121.2	1.34	107
LRC1_2_4	106	2525.8*	284	2552.1	2583.2	2.22	190
LRC1_2_5	107	3715.8	42	3766.2	3788.3	1.91	55
LRC1_2_6	105	3360.9	7	3382.4	3401.0	1.18	39
LRC1_2_7	106	3317.7	408	3344.3	3377.1	1.76	52
LRC1_2_8	104	3086.5	1563	3129.5	3143.7	1.82	63
LRC1_2_9	104	3053.8	1757	3093.6	3141.5	2.79	54
LRC1_2_10	105	2837.5*	156	2857.2	2881.3	1.52	51
Average			473			1.49	72

Table 13
Results of the ALNS heuristic on benchmark PDPTW-*R* instances.

Li and Lim [19] instances	# of requests	Best known value	CPU seconds	ALNS best found	ALNS average value	GAP %	CPU seconds
LR1_2_1	105	4819.1	2	**4819.1**	**4819.1**	0.00	42
LR1_2_2	105	4093.1	21	**4093.1**	4101.4	0.20	96
LR1_2_3	104	3486.8	3691	**3486.8**	3503.6	0.48	162
LR1_2_4	105	2830.7*	228	2839.1	2873.1	1.48	201
LR1_2_5	106	4221.6	3	**4221.6**	4239.2	0.42	42
LR1_2_6	107	3763.0	181	**3763.0**	3769.9	0.18	70
LR1_2_7	103	3112.9*	173	**3112.9**	3124.9	0.38	107
LR1_2_8	103	2645.4*	226	2652.4	2664.7	0.72	159
LR1_2_9	105	3953.5	15	**3953.5**	3961.0	0.19	52
LR1_2_10	104	3386.3	1377	3390.4	3411.2	0.73	62
Average			592			0.48	99

decimal digit. For the ALNS algorithm, we then present the best solution value obtained in column "ALNS best found". In addition, we indicate "ALNS average value" after 10 runs with the corresponding average GAP (%) (i.e., let $v(ALNS)$ be the value of the solution found by the proposed ALNS, then, the GAP (%) $= 100\,(v(ALNS) - v(\text{Best}))/v(ALNS)$, where $v(\text{Best})$ is the value of the best solution known). Finally, we show the average CPU time required to solve the instances using the algorithm.

As shown in Tables 11–13, the ALNS heuristic proved good performance on the PDPTW instances. For 18 out of 30 instances, our heuristic algorithm obtained the best known solutions in at least one out of the 10 runs. For the remaining 12 instances, the GAPs are found to be not greater than 2.79%. The average CPU time needed to solve the instances is found to be 78 s.

4.4. Results of the generated instances

In this section, we provide the results on the four generated sets of PDPTW-SL instances. These sets are generated from the three main datasets described in Section 4.1, by considering subsets of request, transfer node and PD vehicle sets. For the first group (with up to 12 requests and one SL), each instance was solved 10 times with the proposed heuristic and once with the PDPTW-SL MIP model by using CPLEX 12.3 (IBM ILOG [17]) with its default settings and the valid inequalities proposed by Ghilas et al. [14]. A time-limit of ten hours was imposed to CPLEX. The following three groups were solved 10 times using the proposed ALNS in the context of PDPTW-SL and PDPTW. The detailed results of these experiments are presented in Tables 14–17.

Table 14
Computational results for the instances with up to 12 requests.

Instance	CPLEX				ALNS		
	Upper bound	Lower bound	GAP %	CPU seconds	Value	GAP %	CPU seconds
C_6_1_4	369.36	369.36	0.00	5267	369.36	0.00	1
C_7_1_4	390.31	390.31	0.00	3762	390.31	0.00	1
C_8_1_4	446.35	384.93	13.76	36,000	446.35	–	2
C_9_1_4	493.66	327.43	33.67	36,000	472.68	–	2
C_10_1_4	–	335.89	–	36,000	510.01	–	2
C_11_1_4	–	347.68	–	36,000	522.84	–	2
C_12_1_4	–	366.81	–	36,000	541.60	–	2
RC_6_1_4	572.76	572.76	0.00	185	572.76	0.00	1
RC_7_1_4	575.95	575.95	0.00	368	575.95	0.00	1
RC_8_1_4	585.32	585.32	0.00	961	585.32	0.00	1
RC_9_1_4	593.70	593.70	0.00	32,204	593.70	0.00	2
RC_10_1_4	599.95	599.95	0.00	24,925	599.95	0.00	2
RC_11_1_4	624.48	624.48	0.00	18,128	624.48	0.00	2
RC_12_1_6	–	608.89	–	36,000	662.03	–	2
R_6_1_4	416.16	416.16	0.00	2	416.16	0.00	1
R_7_1_4	473.06	473.06	0.00	5	473.06	0.00	1
R_8_1_4	558.17	558.17	0.00	16	558.17	0.00	1
R_9_1_4	632.42	632.42	0.00	8782	632.42	0.00	1
R_10_1_4	636.05	636.05	0.00	3421	636.05	0.00	2
R_11_1_4	748.29	748.29	0.00	17,493	748.29	0.00	2
R_12_1_6	–	771.01	–	36,000	934.73	–	2

Table 15
Results of *C* instances.

Instance	Best known value	Average value	Average GAP %	Cost savings %	Best driving time	Driving time savings %	CPU seconds	# of vehicles used	Units on SLs
C25_0_8	1076.70	1083.40	0.62		2153.40		1	6	
C25_1_8	879.94	882.63	0.30	18.27	1737.88	19.30	7	5	11
C25_2_8	878.86	881.26	0.27	18.37	1723.72	19.95	8	7	17
C25_3_8	753.03	754.33	0.17	30.06	1468.06	31.83	14	5	19
C_50_0_12	1529.37	1541.56	0.79		3058.73		4	8	
C_50_1_12	1415.60	1425.67	0.71	7.44	2793.19	8.68	124	10	14
C_50_2_12	1342.23	1356.01	1.02	12.24	2634.47	13.87	121	10	50
C_50_3_12	1214.16	1229.30	1.23	20.61	2354.31	23.03	144	10	74
C_75_0_16	2040.54	2069.10	1.38		4081.09		12	10	
C_75_1_16	1807.33	1829.14	1.19	11.43	3558.66	12.80	411	12	56
C_75_2_16	1686.85	1702.30	0.91	17.33	3283.70	19.54	528	12	90
C_75_3_16	1621.18	1641.60	1.24	20.55	3112.36	23.74	650	13	130
C_100_0_20	2349.46	2378.79	1.23		4698.91		34	12	
C_100_1_20	2112.73	2126.42	0.64	10.08	4167.47	11.31	2378	13	58
C_100_2_20	2040.36	2069.71	1.42	13.16	3964.72	15.62	2224	14	116
C_100_3_20	1915.40	1939.87	1.26	18.47	3662.80	22.05	2357	14	168

Table 16
Results of *RC* instances.

Instance	Best known value	Average value	Average GAP %	Cost savings %	Best driving time	Driving time savings %	CPU seconds	# of vehicles used	Units on SLs
RC_25_0_8	1572.08	1572.08	0.00		3144.16		1	7	
RC_25_1_8	1530.43	1530.43	0.00	2.65	3048.86	3.03	3	8	6
RC_25_2_8	1413.06	1414.44	0.10	10.12	2808.12	10.69	4	7	9
RC_25_3_8	1377.54	1382.58	0.36	12.37	2727.08	13.27	6	7	14
RC50_0_14	2270.83	2270.83	0.00		4541.66		3	10	
RC50_1_14	2213.06	2213.54	0.02	2.54	4416.12	2.76	11	10	5
RC50_2_14	2213.06	2213.85	0.04	2.54	4416.12	2.76	12	10	5
RC50_3_14	2211.89	2214.13	0.10	2.60	4409.78	2.90	15	10	7
RC_75_0_16	2863.05	2868.34	0.18		5726.11		9	12	
RC_75_1_16	2814.39	2825.63	0.40	1.70	5616.78	1.91	143	12	12
RC_75_2_16	2814.39	2836.39	0.78	1.70	5616.78	1.91	155	12	12
RC_75_3_16	2814.39	2839.05	0.87	1.70	5616.78	1.91	181	12	12
RC_100_0_18	3,114.35	3,119.10	0.15		6228.70		25	12	
RC_100_1_18	3088.07	3093.66	0.18	0.84	6164.13	1.04	794	12	12
RC_100_2_18	3088.07	3100.22	0.39	0.84	6164.13	1.04	677	12	12
RC_100_3_18	3088.07	3134.30	1.48	0.84	6164.13	1.04	985	12	12

Table 17
Results of *R* instances.

Instance	Best known value	Average value	Average GAP %	Cost savings %	Best driving time	Driving time savings %	CPU seconds	# of vehicles used	Units on SLs
R_25_0_8	1717.39	1724.71	0.42		3449.42		1	8	
R_25_1_8	1636.12	1637.72	0.10	4.73	3266.24	5.31	2	8	3
R_25_2_8	1597.75	1601.97	0.26	6.97	3185.50	7.65	2	8	5
R_25_3_8	1540.96	1545.94	0.32	10.27	3069.92	11.00	3	8	6
R_50_0_14	2620.88	2649.70	1.09		5241.76		2	13	
R_50_1_14	2595.14	2604.2	0.35	0.98	5186.28	1.06	8	12	2
R_50_2_14	2412.89	2420.67	0.32	7.94	4803.78	8.36	20	12	11
R_50_3_14	2392.81	2403.52	0.45	8.70	4759.62	9.20	21	12	13
R_75_0_20	3337.85	3337.85	0.00		6675.70		7	14	
R_75_1_20	3337.85	3355.41	0.52	0.00	6675.70	0.00	69	14	0
R_75_2_20	3321.14	3361.77	1.21	0.50	6630.28	0.68	98	15	12
R_75_3_20	3321.14	3349.07	0.83	0.50	6630.28	0.68	110	15	12
R_100_0_24	3643.07	3646.89	0.10		7286.14		26	15	
R_100_1_24	3643.07	3665.31	0.61	0.00	7286.14	0.00	538	15	0
R_100_2_24	3628.87	3646.65	0.49	0.39	7245.74	0.55	690	16	12
R_100_3_24	3628.87	3637.37	0.23	0.39	7245.74	0.55	725	16	12

Table 14 indicates that in most of the cases, the ALNS algorithm generated the same solutions as CPLEX, but in a significantly shorter CPU times. For the instances solved to optimality, the average CPU time required by CPLEX is approximately 8251 s whereas ALNS needed approximately 2 s. In some cases where CPLEX could not find any solution or could obtain a sub-optimal one, the proposed ALNS was able to find solutions that have a tighter GAP relative to the best lower bound found within the imposed time limit.

Tables 15–17 provide the results obtained for larger instances. The column *Instance* indicates the instance identification. The *Best known cost* column indicates the best objective value found after 10 runs of the algorithm. In addition, columns *Average cost* and *Average GAP* show the average objective values over 10 runs and respectively the average GAP from the best solution found. The *Cost savings* column indicates the cost savings of the best PDPTW-SL solution compared to the best corresponding PDPTW solution. The *Best driving time* column shows the total driving time of the best solution found and the *Driving time savings* indicate the savings with regard to the total driving time. *CPU* indicates the average computational time for solving the instances. *Vehicles used* and *Units on SLs* provide the number of PD vehicles used and the number of shipments (demand units) on the available SLs in the best solution found.

The proposed algorithm is relatively fast. For example, instances of up to 100 requests are solved in less than 40 min. As it can be noticed from Tables 15–17, the ALNS algorithm for the PDPTW-SL is substantially slower than the same algorithm in case of the PDPTW. The main reason is the extra complexity that is induced by having the flexibility of using available scheduled lines, thus making multiple PD-vehicle routes depend on each other. In particular synchronization constraints (i.e., time windows and capacity) require extra computation time (e.g., 20–30% of the CPU time).

It is noted that the efficiency of the proposed system may be highly dependent on both the spatial pattern of the requests and the configuration of the scheduled lines. Unless the design of the scheduled line services (routes and schedules) is integrated with vehicle routing, it is likely that the gains from an integrated system operation would be very small. Hence, designing such a system involves tactical decisions related to the pattern of the scheduled lines (positioning of the transfer nodes relative to the demand nodes clusters), the storage areas at the transfer nodes, and the re-design of the SL vehicles (e.g., freight compartment), that are not taken into consideration in this paper as the focus was on operational costs of the proposed system.

Overall, the results indicate the potential operating costs due to available scheduled lines. In particular, the savings range from 0 to 30% with regard to operating costs and from 0 to 31% in terms of driving time. Note that in this study the amount of CO_2e emissions is directly proportional to total driving time as we disregard the extra emissions produced by the SLs due to extra carried weight (i.e., packages).

The most of the savings can be achieved by shipping requests on the available SLs. However, the number of PD vehicles used slightly increases in PDPTW-SL compared to the solutions for PDPTW, especially in *C* instances. It is explained by the fact that the number of vehicles used depends on the time windows, capacities, and demands. Moreover, we note that savings decrease along with the increase in the number of requests for *R* and *RC* instances. This can be explained by the increasing density of the requests over the considered area (200×200 time units). Hence, driving time from one demand node to another becomes shorter. For *C* instances the savings remain significant for larger instances as well. Hence, we can conclude that the more demand points are clustered around transfer nodes, the better performance of the system is. An obvious point that is supported by the results is that the more SLs are available, more savings can be achieved compared to the classical PDPTW.

4.5. The effect of heterogeneous routing costs on the algorithm performance

In this section, we present computational experiments on the instances with heterogeneous vehicle routing costs that are based on the vehicle capacity. The minimum-capacity vehicle is assumed to cost 0.5 €per operating time unit. Larger vehicles are assigned a cost that increases linearly along with the carrying capacity. Each instance is run 10 times and the results are given in Table 18. The columns are self-explanatory, similar to previously presented tables.

Table 18
An analysis of heterogeneous vehicle routing costs.

Instance	Best known value	Average value	Average GAP %	Cost savings %	CPU seconds	# of vehicles used	Units on SLs
25_C_0_8	1676.53	1693.02	0.97		2	6	
25_C_1_8	1333.99	1354.51	1.51	20.43	6	6	13
25_RC_0_8	2513.39	2540.65	1.07		1	7	
25_RC_1_8	2403.87	2421.96	0.75	4.36	4	8	6
25_R_0_8	2145.41	2171.02	1.18		2	8	
25_R_1_8	2019.78	2035.34	0.76	5.86	2	8	1
50_C_0_12	1623.09	1635.45	0.76		3	8	
50_C_1_12	1459.64	1489.81	2.03	10.07	37	9	17
50_RC_0_14	3751.17	3784.39	0.88		5	10	
50_RC_1_14	3570.35	3611.17	1.13	4.82	12	10	5
50_R_0_14	3271.88	3281.55	0.29		4	12	
50_R_1_14	3231.22	3258.57	0.84	1.24	7	13	6
75_C_0_16	2278.57	2309.17	1.33		8	11	
75_C_1_16	1955.53	1972.39	0.85	14.18	698	11	25
75_RC_0_16	3164.14	3183.45	0.61		7	12	
75_RC_1_16	3144.66	3155.30	0.34	0.62	122	12	6
75_R_0_20	3583.39	3603.99	0.57		5	14	
75_R_1_20	3572.31	3586.76	0.40	0.31	138	15	2
100_C_0_20	3985.45	4017.42	0.80		22	12	
100_C_1_20	3557.39	3576.59	0.54	10.74	2178	13	36
100_RC_0_18	3420.74	3461.23	1.17		9	13	
100_RC_1_18	3344.28	3362.95	0.56	2.24	644	12	6
100_R_0_24	3758.27	3792.44	0.90		13	15	
100_R_1_24	3758.27	3825.33	1.75	0.00	668	15	0

According to the obtained results, the proposed ALNS seems to perform stable when considering heterogeneous costs, leading to solutions with an average GAP of 0.91% compared to the best known solutions. In addition, the results indicate that objective function values tend to increase due to larger routing costs (heterogeneous costs) for the considered vehicles, compared to homogeneous case (i.e., 0.5 €for all vehicles). In this context, making use of available SLs leads to average cost savings of 6.24% compared to the corresponding best-known PDPTW solutions.

4.6. An application study

In this section we investigate the performance of the PDPTW-SL environment on a realistic scheduled lined system. In particular, we solve one instance of 100 randomly generated requests on a 60×60 time-units area, three depots and 20 PD vehicles. The scheduled lines graph is shown in Fig. 5 and it is inspired from the current metro system in Amsterdam (see Fig. 1). The distances are considered Euclidean and time windows are randomly generated.

The results shown in Table 19 are obtained after five runs of the algorithm and indicate the best solutions found for both PDPTW-SL and PDPTW. In particular, the PDPTW-SL integrated transportation system led to 5% savings in terms of operating costs and 9% in shorter

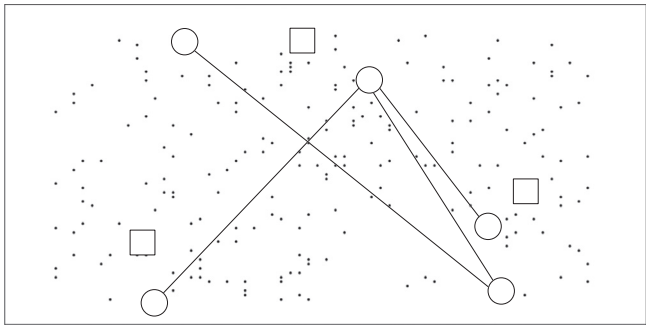

Fig. 5. An illustrative representation of Amsterdam's metro system.

Table 19
Results on a realistic real-life scheduled line system.

Instance	PDPTW-SL					PDPTW			
	Driving time	Units on SLs	Cost €	CPU seconds	# of vehicles	Driving time	Cost €	CPU seconds	# of vehicles
Amsterdam 100 requests	1767.22	36	919.61	1258.52	10	1945.69	972.85	34.62	7

total driving time. Even though PDPTW-SL system can lead to operating costs and CO_2e emissions benefits, the number of vehicles used is increased as compared to PDPTW.

5. Conclusions

We proposed a metaheuristic algorithm to solve the PDPTW-SL. To investigate the performance of the algorithm, we have generated several sets of PDPTW-SL instances. Compared to the existing solutions on a set of PDPTW instances, the proposed algorithm performed well in terms of both, solution quality (with a maximum GAP of 2.79%) and CPU time (78 s on average). Furthermore, we have also shown that small PDPTW-SL instances can be solved optimally by the proposed formulation. The solutions obtained from solving larger instances, up to 100 requests, were compared to their corresponding PDPTW solutions and it is concluded that the flexibility of using scheduled line services leads to significant cost savings and fewer CO_2e emissions. However, note that the performance of the PDPTW-SL system may be highly dependent on the relative positioning of the scheduled lines to the request nodes. In addition, investment costs needed for implementing such system may affect its outcome.

The reliability of such a system may decrease due to extra causes of delays and cargo damages (i.e., transfers to/from SLs, delays in SL schedules). Therefore, shippers may not be willing to use PDPTW-SL system. In order to tackle such issues, more advanced planning tools are needed, which consider stochastic aspects of the problem. The current research state of the considered problem is yet young and further industry collaborations are to be done in order to learn more about practical issues that may or may not prevent the implementation and the execution of such a system.

Overall the numerical experiments indicate that the proposed algorithm leads to high-quality routing solutions for relatively large instances in a reasonable amount of time. Regarding extensions of the investigated problem, additional aspects may be considered such as driver-related constraints, stochastic aspects (demands, travel times) or passenger requests, and the other related constraints (e.g., maximum ride-time). Investigation of exact decomposition algorithms (e.g., Branch and Price) could also be an interesting research direction. In addition, since the results show that the number of PD vehicles used may increase when SLs are introduced, it would be interesting to introduce fixed costs for the PD vehicles as well to make more clear the trade-off between using fewer PD vehicles or using SLs with more PD vehicles.

Acknowledgments

The authors gratefully acknowledge funds provided by Dinalog, the Dutch Institute for Advanced Logistics, under the Grant titled “Cargo Hitching”, # 2011 4 086R. Thanks are due to the Editor and two anonymous reviewers for their useful comments and for raising interesting points for discussion.

References

[1] Aldaihani M, Dessouky M. Hybrid scheduling methods for paratransit operations. Comput Ind Eng 2003;45(1):75–96.
[2] Baldacci R, Bartolini E, Mingozzi A. An exact algorithm for the pickup and delivery problem with time windows. Oper Res 2011;59(2):414–26.
[3] Berbeglia G, Cordeau J-F, Gribkovskaia I, Laporte G. Static pickup and delivery problems: a classification scheme and survey. TOP: Off J Span Soc Stat Oper Res 2007;15(1):1–31.
[4] Braekers K, Caris A, Janssens G. Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. Transp Res Part B 2014;67:166–86.

[5] Cargo Tram. Official webpage. Available at: ⟨www.eltis.org/index.php?id=13&study_id=1547⟩; 2012 [accessed on 21.11.2015].
[6] Cordeau J-F, Laporte G. The dial-a-ride problem: models and algorithms. Ann Oper Res 2007;153(1):29–46.
[7] Cortes E-C, Matamala M, Contardo C. The pickup and delivery problem with transfers: formulation and a branch-and-cut solution method. Eur J Oper Res 2010;200(3):711–24.
[8] Demir E, Bektaş T, Laporte G. An adaptive large neighborhood search heuristic for the pollution-routing problem. Eur J Oper Res 2012;223(2):346–59.
[9] Demir E, Bektaş T, Laporte G. A review of recent research on green road freight transportation. Eur J Oper Res 2014;237(3):775–93.
[10] Demir E, Bektaş T, Laporte G. The bi-objective pollution-routing problem. Eur J Oper Res 2013;232(3):464–78.
[11] Demir E, Huang Y, Scholts S, Van Woensel T. A selected review on the negative externalities of the freight transportation: modeling and pricing. Transp Res Part E: Logist Transp Rev 2015;77:95–114.
[12] DHL-Packstation. DHL official webpage. Available at: ⟨www.dhl.de/en/paket/pakete-empfangen/packstation.html⟩; 2013 [accessed on 21.11.2015].
[13] Dijkstra E-W. A note on two problems in connexion with graphs. Numer Math 1959;1:269–71.
[14] Ghilas V, Demir E, Van Woensel T. Integrating passenger and freight transportation: model formulation and insights. Technical Report, Industrial Engineering & Innovation Sciences, Eindhoven University of Technology, BETA No. 441; 2013. p.23.
[15] Hall C, Andersson H, Lundgren J, Varbrand P. The integrated dial-a-ride problem. Public Transp 2009;1:39–54.
[16] Hurtigruten. Hurtigruten official webpage. Available at: ⟨www.hurtigruten-web.com/index_en.html⟩; 2013 [accessed on 11.11.2015].
[17] IBM ILOG. Copyright international business machines corporation 1987; 2013.
[18] Levin Y, Nediak M, Topaloglu H. Cargo capacity management with allotments and spot market demand. Oper Res 2012;60(2):351–65.
[19] Li H, Lim A. A metaheuristic for the pickup and delivery problem with time windows. Int J Artif Intell Tools 2003;12(2):173–86.
[20] Liaw C-F, White C, Bander J. A decision support system for the bimodal dial-a-ride problem. IEEE Trans Syst Man Cybern Part A 1996;26(5):552–65.
[21] Lindholm M, Behrends S. Challenges in urban freight transport planning—a review in the Baltic Sea Region. J Transp Geogr 2012;22:129–36.
[22] Masson R, Lehuede F, Peton O. An adaptive large neighborhood search for the pickup and delivery problem with transfers. Transp Sci 2012;47:1–12.
[23] Masson R, Lehuede F, Peton O. The dial-a-ride problem with transfers. Comput Oper Res 2014;41:12–23.
[24] Nash C, editor. Economics of public transport. London, UK: Longman; 1982.
[25] Petersen HL, Røpke S. The pickup and delivery problem with cross-docking opportunity. In: Computational Logistics: Second International Conference, ICCL 2011, Hamburg, Germany, 19-22 September 2011, Lecture notes in computer science, vol. 6971. Springer, Berlin, Heidelberg; 2011. p. 101–13.
[26] Pisinger D, Røpke S. A general heuristic for vehicle routing problems. Comput Oper Res 2007;34(8):2403–35.
[27] Røpke S, Cordeau J-F. Branch and cut and price for the pickup and delivery problem with time windows. Transp Sci 2009;43(3):267–86.
[28] Røpke S, Pisinger D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp Sci 2006;40(4):455–72.
[29] Shang J, Cuff C. Multicriteria pickup and delivery problem with transfer opportunity. Comput Ind Eng 1996;30(4):631–45.
[30] Shaw P. Using constraint programming and local search methods to solve vehicle routing problems. In: Proceedings of the 4th international conference on principles and practice of constraint programming. New York: Springer; 1998. p. 417–31.
[31] Trentini A, Malhene N. Toward a shared urban transport system ensuring passengers & goods cohabitation. Trimest del Lab Territ Mobilita e Ambient—TeMALab 2010;4:37–44.
[32] Trentini A, Masson R, Lehuede F, Malhene N, Peton O, Tlahig H. A shared “passenger & goods” city logistics system. In: 4th international conference on information systems, logistics and supply chain, Quebec, Canada; 2012.