Computers and Operations Research 126 (2021) 105110

Contents lists available at ScienceDirect

Computers and Operations Research

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

A Large Neighborhood Search for the Pickup and Delivery Problem with Time Windows, Split Loads and Transshipments

David Wolfinger [*]

Christian Doppler Laboratory for Efficient Intermodal Transport Operations, Department of Business Decisions and Analytics, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

A R T I C L E I N F O

Article history:
Received 3 May 2020
Revised 13 August 2020
Accepted 24 September 2020
Available online 09 October 2020

Keywords:
Pickup and Delivery Problem
Split Load
Transshipment
Large Neighbourhood Search

A B S T R A C T

In the pickup and delivery problem (PDP) a fleet of capacitated vehicles must be routed in order to satisfy a set of customer requests. In this article, we address a variant of the one-to-one PDP where each customer location can be visited several times and load may be transshipped from one vehicle to another at specific locations. The goal is to minimize the sum of travel costs and transshipment costs. The corresponding problem is called the PDP with time windows, split loads and transshipments. We present an arc-based mixed-integer formulation for the problem and propose a large neighbourhood search metaheuristic for solving it. Extensive computational experiments using both benchmark instances from the literature and instances based on real-world data illustrate the performance of the solution method and give insights regarding the benefits of combining split loads and transshipments.

1. Introduction

In the pickup and delivery problem (PDP) the aim is to define a set of vehicle routes such that a set of customer requests is satisfied and the transportation costs are minimized. A request consists of picking up a load at its origin location and delivering that load to its destination location. Several requests may share the same vehicle as long as their combined load does not exceed the vehicle capacity. Each origin and destination must be visited exactly once. The vehicle routes must start and finish at the vehicle depots, and they must satisfy precedence and coupling constraints: for each request, the origin must be visited before the destination, and both locations must be visited by the same vehicle.

The PDP is an extension of the well-known capacitated vehicle routing problem (CVRP) and has a wide range of practical applications in both people and freight transportation (Toth and Vigo, 2014). The PDP has been extensively studied over the past decades. For detailed reviews we refer the interested reader to Parragh et al. (2008a,b).

In this article, we address a generalized version of the one-to-one PDP, called the PDP with time windows, split loads and transshipments (PDPTWSLT). We present an arc-based mixed-integer formulation for the PDPTWSLT and propose a large neighbourhood search (LNS) metaheuristic for solving it.

In the PDPTWSLT the constraint that each customer location must be visited exactly once is relaxed. This allows for the total load of a request to be delivered in several trips rather than only one. Clearly, this requires additional visits to the request's origin and destination. Nevertheless, performing more visits to a location may indeed result in lower costs due to the vehicle capacity limitations. That is, splitting a load can potentially eliminate a dedicated trip for delivery by allocating it to other vehicles which have excess capacity. Additionally, in the PDPTWSLT, load can be transshipped from one vehicle to another at intermediary locations, called *transshipment locations*. Depending on the application, transshipment may take place either at customer locations (other than the request's origin and destination) or at dedicated transshipment locations, or both. We consider the case where transshipment is only allowed at dedicated transshipment locations. Allowing transshipment requires the relaxation of the coupling constraint, because the vehicle which picks up the load from its origin may not be the same vehicle which delivers it to its destination. Furthermore, allowing transshipment implies a new precedence constraint: if a load uses a transshipment location, it must first be delivered at the transshipment location by the first vehicle before the second vehicle can pick it up. Similar to splitting loads, transshipping loads can lead to lower costs through better utilization of vehicle capacities.

The PDPTWSLT arises naturally in the context of multimodal freight transportation, where transshipments need to take place per definition. In this form of transportation, whenever goods are

* Corresponding author.
E-mail address: david.wolfinger@univie.ac.at

https://doi.org/10.1016/j.cor.2020.105110
0305-0548/© 2020 The Author(s). Published by Elsevier Ltd.

not transported in containers, requests are often splitable. This is the case, for example, for steel products or raw materials (either unpacked or packed in smaller transportation units such as pallets, bags, barrels, boxes or crates). According to the statistical office of the European Union, the share of containerized transport by inland waterway in the EU was only 9.2% in 2017 (Eurostat, 2020). For rail transport the share of containerized transport was 17.9%. Given that most companies in the EU are not located at a port and do not have direct access to a rail network, it is safe to assume that the majority of inland waterway transport, respectively rail transport, is part of a multimodal transport chain. Thus, based on the above numbers, there is a large potential for applying the PDPTWSLT in multimodal freight transportation. The PDPTWSLT also lends itself to various other practical applications such as door-to-door transportation services for people and less-than-truckload transportation. In door-to-door people transportation, split loads and transshipments are already studied in the literature and used in real-world cases, albeit separately (see for example Parragh et al. (2015) for split loads or Masson et al. (2013) for transshipments). Combining both in order to further decrease costs is the logically next step. Similarly, in less-than-truckload transportation load splitting is already considered in some real-world cases and addressed in the literature (see for example Nowak et al., 2008). Depending on the available infrastructure and the type of transported goods, transshipping requests can in many cases additionally be considered.

The contribution of our article is twofold. First, we propose a LNS metaheuristic for solving the PDPTWSLT. This is the first heuristic solution method which combines split loads and transshipments in the context of the PDP. We propose a novel solution representation, a new and efficient strategy of handling split loads, a new destruction heuristic specifically designed for split loads, and a new way of evaluating and inserting requests with transshipments. Second, we present an extensive computational study which illustrates the performance of the LNS and gives insights regarding the benefits of combining split loads and transshipments. We use both benchmark instances from the literature and new random instances based on real-world data for the computational experiments.

The remainder of this article is structured as follows. In the next Section, we discuss the related literature. In Section 3, we give a formal description of the PDPTWSLT and present an arc-based mixed-integer linear programming formulation for it. Section 4 is devoted to the LNS. The computational study is presented in Section 5. Conclusions are reported in Section 6.

2. Literature Review

Despite its wide-ranging practical applicability, the PDPTWSLT has received little attention in the scientific literature. Indeed, the only two closely-related works we are aware of are by Kerivin et al. (2008) and Wolfinger and Salazar-González (2020). Kerivin et al. (2008) assume that all vehicles are identical, that there are no time windows nor transshipment costs, and that there is a completion time limit for each vehicle. Furthermore, they do not consider dedicated transshipment locations, but allow transshipments at every customer location. They propose a branch-and-cut algorithm for solving an arc-based formulation on a time-space graph. The problem addressed by Wolfinger and Salazar-González (2020) is almost identical to ours. Like us, they allow a heterogeneous fleet of vehicles and consider transshipment cost. Contrary to us, they do not consider time windows (and consequently no load-dependent service times), they allow transshipments at every customer location (with the possibility to restrict transshipments to dedicated transshipment locations, though), they consider reloading of requests which were previously unloaded at a transshipment location by the same vehicle, and they allow a location to simultaneously be the origin and/or destination of more than one request. Wolfinger and Salazar-González (2020) present a mixed-integer linear programming formulation for the problem, which eliminates the need to discretize the time component, and propose a branch-and-cut algorithm for solving it. We are not aware of any heuristic solution approach for the PDPTWSLT.

Several other different problems which share characteristics with the PDPTWSLT have been studied in the literature. The option of serving a request in more than one trip was first addressed by Dror and Trudeau (1989, 1990) in the context of the classical CVRP. The problem they proposed relaxes the CVRP and is typically called the split delivery vehicle routing problem (SDVRP). Since then, the SDVRP has been studied – and extended (e.g., with time windows) – by several authors (e.g., Dror et al., 1994; Gendreau et al., 2006; Desaulniers, 2010; Archetti et al., 2011; Archetti et al., 2011). For a detailed review on the SDVRP (and its variants) we refer the interested reader to the surveys of Archetti and Speranza (2008, 2012).

Nowak et al. (2008) introduce the pickup and delivery problem with split loads (PDPSL). They show that it is NP-hard and present a local search based heuristic for the single vehicle version. Additionally, they show that the largest benefit from load splits can be obtained when the size of the loads is just above half of the vehicle capacities. In Nowak et al. (2009) the authors present an empirical study on the benefit of split loads. Şahin et al. (2013) address the multi vehicle variant of the problem. They also propose a local search based heuristic to solve it. In a recent work, Haddad et al. (2018) propose an iterated local search (ILS) metaheuristic for the same problem; which is currently the state-of-the-art heuristic solution approach for the PDPSL. With respect to exact methods, Haddad et al. (2018) additionally propose a branch-and-price algorithm for the PDPSL. Stålhane et al. (2012) address the maritime version of the PDPSL. They additionally consider time windows and solve the problem with a branch-and-price-and-cut method. In the context of people transportation, Parragh et al. (2015) introduce the dial-a-ride problem with split requests and profits. They present a branch-and-price algorithm as well as a variable neighbourhood search algorithm for solving the problem.

Mitrovic-Minic and Laporte (2006) introduce the pickup and delivery problem with transshipment (PDPT). They show the usefulness of performing transshipments and present a two-phase heuristic to solve the problem. Qu and Bard (2012) use a GRASP metaheuristic approach which applies an adaptive large neighbourhood search (ALNS) in the improvement phase to solve the PDPT. Masson et al. (2013) address the PDPT in the context of passenger transportation and use an ALNS to solve it. They extend their solution method and apply it to the dial-a-ride problem with transfers in Masson et al. (2014). Neves-Moreira et al. (2016) present a fix-and-optimize matheuristic algorithm for the full truckload version of the PDPT. Bouros et al. (2011) tackle the dynamic version of the PDPT with a novel heuristic solution method which is based on finding the shortest path from the origin to the destination for each new request. The most recent work addressing the PDPT is by Danloup et al. (2018). They propose both a LNS and a genetic algorithm (GA) for solving it. Both their methods outperform the solution approaches of Qu and Bard (2012) and Masson et al. (2013). With respect to exact methods, Cortés et al. (2010) propose an arc-based mixed-integer linear programming formulation for the PDPT, which they solve with a branch-and-cut method using Benders decomposition. Rais et al. (2014) present a new mixed-integer formulation and solve it with a commercial MIP solver.

Both the PDPT and the PDPSL are known to be much more difficult to solve in practice than the PDP. This is due to the additional aspects and constraints which need to be taken into account (such as dealing with the synchronization of vehicles in the PDPT, or dealing with multiple flows for the same request and guaranteeing that every request is fully satisfied in the PDPSL), as well as the larger solution space (PDP solutions are valid for the PDPT and PDPSL, but not vice versa).

3. Problem Description

In this section, we formally describe the PDPTWSLT and present an arc-based mixed-integer linear programming formulation for it.

Let R be the set of requests. Each request $r \in R$ is associated with an origin location r^+, a destination location r^-, a positive load d_r and two time windows; one for the origin location ($[a_{r^+}, b_{r^+}]$) and one for the destination location ($[a_{r^-}, b_{r^-}]$). Let $P = \{r^+ | r \in R\}$ and $D = \{r^- | r \in R\}$ denote the set of pickup and delivery nodes, respectively.

Let K be a heterogeneous fleet of vehicles. Each vehicle $k \in K$ has a start depot k^+, an end depot k^-, and a non-negative capacity q_k. The locations of the start depot and end depot can coincide for a vehicle. The set of depots is denoted by $N = \bigcup_{k \in K}\{k^+, k^-\}$.

Load may be transshipped from one vehicle to another at specific locations. Let T be the set of transshipment locations. Transshipment is allowed only at locations in T. We do not limit the amount of times a request (or a part of it) can be transshipped.

Furthermore, the load of a request may be split. This means that when a vehicle arrives at an origin it can pick up any portion or the complete load of the request. When the vehicle arrives at the request's destination, all load from this vehicle associated with this request is delivered. As a consequence a customer might be visited by more than one vehicle or more than once by the same vehicle. Each part of a request may thus be transported on a different path from its origin to its destination. We adopt the customary definition of a split: a split occurs when the load of a request is serviced by a larger number of trips than the minimum necessary (Haddad et al., 2018).

In the PDPTWSLT each location (except the depots) may be visited more than once by the same vehicle. Multiple visits by the same vehicle can occur when a request is split or at a transshipment location. In order to work on a graph where each vertex is visited at most once, we need to use location-copies; one copy for each potential visit by the same vehicle. We assume to be given an explicit number of copies m_i for each location i. m_i is an upper bound representing the maximum number of allowed visits by the same vehicle at location i. However, the value of m_i can be set arbitrarily large such that it does not impose any limitation on the construction of vehicle routes. Let V_i be the set of m_i copies of location $i \in P \cup D \cup T$. Then, the problem is defined on a directed graph $G = (V, A)$ where the vertex set is $V = N \cup \bigcup_{i \in P \cup D \cup T} V_i$ and the arc set is $A = \bigcup_{k \in K} A^k$, with $A^k \subset V \times V$ representing the feasible movements of each vehicle $k \in K$.

Each arc $a \in A^k$ and each vehicle $k \in K$ is associated with a travel cost c_a^k and a travel time t_a^k. We assume that the triangle inequality holds for travel costs and travel times. Furthermore, each location $i \in T$ is associated with a cost ψ_i per unit of transshipped load.

At each customer location, the service must start within its time window. Arrival of a vehicle before the beginning of the time window is allowed. In this case, the vehicle must await the start of the time window before performing the service. Each depot is also associated with a time window. The service time at a location depends on the quantity to (un)load. h denotes the time needed to (un)load one unit.

A route for a vehicle corresponds to a path from its start depot to its end depot in the graph G. The path is feasible if the vehicle capacity is never exceeded and the time windows at all visited customer locations are respected. In case a vehicle visits both the origin and the destination of a request on its path, the pickup visit must precede the delivery visit. Furthermore, in case of a transshipment, the vehicle which receives the load must start its service at the transshipment location after the vehicle which gives the load.

The objective of the PDPTWSLT is to find a set of feasible vehicle routes which serves all requests, such that the sum of travel costs and transshipment costs is minimized.

The following mathematical variables are used to formulate the PDPTWSLT:

s_v^k	start time of service of vehicle $k \in K$at node $v \in V$
f_{ar}^k	amount of request $r \in R$ transported by vehicle $k \in K$ over arc $a \in A$
$g_{vr}^{kk'}$	amount of request $r \in R$ transshipped from vehicle $k \in K$ to vehicle $k' \in K$ at node $v \in \{V_i \vert i \in T\}$
σ_{vr}^k	amount of request $r \in R$ (un)loaded from vehicle $k \in K$ at node $v \in V$
x_a^k	$\begin{cases} 1 \text{ iff vehicle } k \in K \text{ traverses arc } a \in A \\ 0 \text{ otherwise} \end{cases}$
y_v^k	$\begin{cases} 1 \text{ iff vehicle } k \in K \text{ visits vertex } v \in V \\ 0 \text{ otherwise} \end{cases}$
$z_v^{kk'}$	$\begin{cases} 1 \text{ iff load is transshipped from vehicle } k \in K \\ \text{to vehicle } k' \in K \text{ at node } v \in \{V_i \vert i \in T\} \\ 0 \text{ otherwise.} \end{cases}$

The location-copies make it possible to represent the route of a vehicle k by an elementary path in the graph $G^k = (V, A^k)$, even when the vehicle visits the same location more than once. An elementary path can be formed with the binary variables x_a^k and y_v^k.

To abbreviate notation, for any subset $S \subseteq V$, we write $\delta_k^+(S) = \left\{(u, v) \in A^k | u \in S, v \notin S\right\}$ and $\delta_k^-(S) = \left\{(u, v) \in A^k | u \notin S, v \in S\right\}$. Also, for any subset $F \subseteq A^k$, we write $x^k(F) = \sum_{a \in F} x_a^k$ and $f_r^k(F) = \sum_{a \in F} f_{ar}^k$.

Then, the PDPTWSLT can be formulated as the following arc-based mixed-integer linear program. Note that this formulation is based on the one presented in Wolfinger and Salazar-González (2020). It extends the formulation from Wolfinger and Salazar-González (2020) with the variable σ and constraints (10) and (11) in order to account for load-dependent service times, as well as constraints (15) in order to account for time windows. Furthermore, the variables g and z – and consequently all constraints in which these two variables appear – are slightly modified because we do not allow transshipment at customer locations nor reloading of requests (both variables have one index less in the current formulation).

$$\min \sum_{k \in K} \sum_{a \in A^k} c_a^k x_a^k + \sum_{r \in R} \sum_{i \in T} \sum_{v \in V_i} \sum_{k, k' \in K} \psi_i g_{vr}^{kk'} \tag{1}$$

subject to

$$x^k(\delta_k^+(v)) = y_v^k \qquad \forall k \in K, \forall v \in V \setminus \{k^-\} \tag{2}$$
$$x^k(\delta_k^-(v)) = y_v^k \qquad \forall k \in K, \forall v \in V \setminus \{k^+\} \tag{3}$$
$$y_v^k = 1 \qquad \forall k \in K, \forall v \in \{k^+, k^-\} \tag{4}$$
$$\sum_{k \in K} f_r^k(\delta_k^+(V_{r^+})) = \sum_{k \in K} f_r^k(\delta_k^-(V_{r^-})) = d_r \qquad \forall r \in R \tag{5}$$
$$\sum_{k \in K} f_r^k(\delta_k^-(V_{r^+})) = \sum_{k \in K} f_r^k(\delta_k^+(V_{r^-})) = 0 \qquad \forall r \in R \tag{6}$$
$$f_r^k(\delta_k^-(v)) + \sum_{k' \in K} (g_{vr}^{k'k} - g_{vr}^{kk'}) = f_r^k(\delta_k^+(v)) \qquad \forall r \in R, \forall i \in T, \forall v \in V_i, \ \forall k \in K \tag{7}$$
$$f_r^k(\delta_k^+(v)) = f_r^k(\delta_k^-(v)) \qquad \forall r \in R, \forall i \in P \cup D \setminus (\{r^+, r^-\}), \forall v \in V_i, \forall k \in K \tag{8}$$
$$s_u^k + \sum_{r \in R} h\sigma_{ur}^k + t_{(u,v)}^k - M(1 - x_{(u,v)}^k) \leqslant s_v^k \qquad \forall k \in K, \forall (u,v) \in A^k \tag{9}$$
$$f_r^k(\delta^+(v)) - f_r^k(\delta^-(v)) \leqslant \sigma_{vr}^k \qquad \forall v \in V, \forall r \in R, \forall k \in K \tag{10}$$
$$f_r^k(\delta^-(v)) - f_r^k(\delta^+(v)) \leqslant \sigma_{vr}^k \qquad \forall v \in V, \forall r \in R, \ \forall k \in K \tag{11}$$
$$\sum_{r \in R} f_{ar}^k \leqslant q_k x_a^k \qquad \forall k \in K, \forall a \in A^k \tag{12}$$
$$s_v^k + \sum_{r \in R} h\sigma_{vr}^k - M(1 - z_v^{kk'}) \leqslant s_v^{k'} \qquad \forall i \in T, \forall v \in V_i, \ \forall k, k' \in K \tag{13}$$
$$g_{vr}^{kk'} \leqslant d_r z_v^{kk'} \qquad \forall r \in R, \forall i \in T, \forall v \in V_i, \forall k, k' \in K \tag{14}$$
$$a_v \leqslant s_v^k \leqslant b_v \qquad \forall i \in N \cup P \cup D, \forall v \in V_i, \forall k \in K \tag{15}$$
$$z_v^{kk'} \leqslant y_v^k; \quad z_v^{kk'} \leqslant y_v^{k'} \qquad \forall i \in T, \forall v \in V_i, \ \forall k, k' \in K \tag{16}$$
$$x_a^k, y_v^k, z_v^{kk'} \in \{0,1\}; \quad f_{ar}^k, \, g_{vr}^{kk'}, \, s_v^k, \, \sigma_{vr}^k \geqslant 0 \quad \forall r \in R, \forall v \in V, \ \forall k, k' \in K, \forall a \in A^k \tag{17}$$

The objective function (1) minimizes the total cost. Constraints (2) and (3) regulate the flow conservation of vehicles. Constraints (4) guarantee that every vehicle starts and ends its path at the depot. Constraints (5) make sure that every request is fully serviced. Constraints (6) forbid that load of a request enters (leaves) its own origin (destination). Constraints (7) regulate the flow conservation of load at every transshipment location, while constraints (8) forbid transshipment at customer locations. Time consistency is ensured by constraints (9), with M being a sufficiently large value. Constraints (10) and (11) make sure that σ_{vr}^k is set correctly. Constraints (12) guarantee that the capacity of a vehicle is never exceeded. Constraints (13) ensure time synchronization at transshipment. Constraints (14) guarantee that the load of a request transshipped from one vehicle to another does not exceed the total demand of the request. Constraints (15) ensure that the time windows are respected. The linking constraints (16) allow a transshipment at a vertex from or to a vehicle only if the vehicle visits that vertex. Constraints (17) set the domain of the decision variables.

4. LNS for the PDPTWSLT

In this section, we describe the LNS metaheuristic developed for the PDPTWSLT. LNS was introduced by Shaw (1997) for solving the CVRP with time windows. The principle mechanism of LNS is to iteratively destroy and repair a solution in order to improve it. We refer to Pisinger and Ropke (2019) for a thorough description of the method.

Algorithm 1 depicts the pseudocode for a minimizing LNS metaheuristic. It takes an initial solution s as input parameter (which has been found, for example, by a simple construction heuristic and does not need to be feasible). In each iteration a number of requests are first removed from the current solution s' (line 5) and subsequently reinserted again (line 6); until all removed requests have been planned or no feasible insertion can be performed. Partial solutions can be used as intermediate infeasible solutions (unplanned requests may be penalized in the objective function). Several different heuristics are used for destroying and repairing a solution. The remaining part of the algorithm updates the so far best solution (line 7) and determines if the newly found solution should be accepted (line 9). Deteriorating solutions may be accepted in order to increase diversification of the search. Before each iteration it is checked if a stopping criterion is met (line 2). In our implementation we stop after a certain amount of time has passed. Finally, at the end, the best solution found is returned (line 11).

Algorithm 1: LNS

```
   Input: initial solution $s$
1  $s_{best} \leftarrow s$;
2  while stopping criterion not met do
3      randomly select one destroy and one repair heuristic;
4      $s' \leftarrow s$;
5      $s' \leftarrow \text{Destroy}(s')$;
6      $s' \leftarrow \text{Repair}(s')$;
7      if $s' < s_{best}$ then
8          $s_{best} \leftarrow s'$;
9      if accept($s'$,$s_{best}$) then
10         $s \leftarrow s'$;
11 return $s_{best}$
```

The proposed LNS for the PDPTWSLT is based on a novel solution representation which allows an efficient handling of both split loads and transshipments. The algorithm additionally adopts a new strategy of handling split loads which does not require to limit the number of split loads (Nowak et al., 2008; Şahin et al., 2013) nor solving computationally difficult and expensive problems (such as a resource-constrained shortest path problem or the knapsack problem) in order to determine which requests to split and in what manner (Haddad et al., 2018). We also propose a new destruction heuristic specifically designed to address split loads. Furthermore, the LNS employs a new way of evaluating and inserting requests with transshipments which does not limit the number of transshipments per request to one (Masson et al., 2013; Danloup et al., 2018). The main components of the LNS are reviewed in more detail in the following subsections.

4.1. Solution representation

In the PDPTWSLT, a request can be serviced by more than one trip. A trip is a path of a (partial) request from its origin to its destination. In case of transshipments, a trip involves two or more vehicles (respectively routes). A trip can be represented by the sequence of nodes which are visited on account of the request. A solution, then, is a set of trips performed by the vehicle fleet.

Fig. 1 shows how a solution is represented in the LNS. It illustrates a solution with three routes, three requests (A, B, C), and two transshipment locations (T_1, T_2). Each trip is marked with a different color. The solution contains both split loads and transshipments. The dashed arrows indicate the direction of flow of a (partial) request in the case of a transshipment. Request A is serviced by two trips (involving all three routes): $(A^+, T_1^{A^-}, T_1^{A^+}, A^-)$ and $(A^+, T_2^{A^-}, T_2^{A^+}, A^-)$. Request B is serviced with one trip (involving all three routes): $(B^+, T_1^{B^-}, T_1^{B^+}, T_2^{B^-}, T_2^{B^+}, B^-)$. Request C is serviced by two trips (one performed by route 2 and the other performed by route 3): (C^+, C^-) and (C^+, C^-).

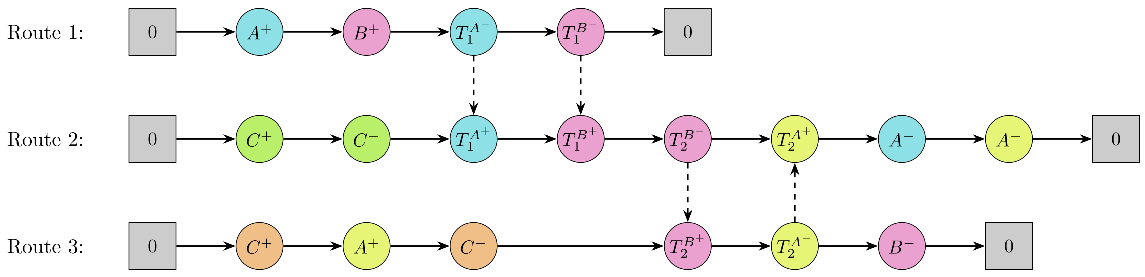


Fig. 1. Representation of a solution (including split loads and transshipments) in the LNS.

4.2. Destruction Heuristics

The destruction heuristics do not remove complete requests, but rather only single trips. Each destruction heuristic selects n trips to remove from the solution. In each iteration n is randomly drawn form the interval $[\alpha \cdot l, \beta \cdot l] \cap \mathbb{N}$, where α and β are two parameters representing the minimum and maximum percentage of trips to be removed in each iteration, and l is the current number of trips in the solution.We implemented five destruction heuristics. The first three are well-known general purpose heuristics from the literature, which can be applied to a variety of different problem types. The fourth heuristic (*cluster removal*) specifically addresses the transshipment aspect, while the last heuristic (*many-split removal*) addresses the split load aspect in our problem. We adapted all previously proposed heuristics from the literature to the concept of trips.

1. *Random removal.* This heuristic randomly selects n trips to be removed from the current solution.
2. *Worst removal.* First, the cost savings produced by the removal of each trip are computed. Then, n trips are randomly selected to be removed. Larger savings correspond to a larger probability of being selected. A complete description of this heuristic can be found in Ropke and Pisinger (2006).
3. *Related removal.* This heuristic was first introduced by Shaw (1998) and aims at removing related requests. The relatedness measure of two trips depends on the distance between the origins and the destinations of the corresponding requests, the difference between the time of service at these nodes and the difference in demand of the corresponding requests. Similar as in the worst removal, n trips are randomly selected to be removed, whereby the probability of being selected increases with the relatedness. Ropke and Pisinger (2006) give a detailed description of this heuristic. We use the same values for the weights as them when calculating the relatedness measure.
4. *Cluster removal.* This heuristic was first introduced by Masson et al. (2013) for the PDPT. It aims at removing requests which can be efficiently routed through a common transshipment location. Two or more requests may benefit from using a common transshipment location if their origin or destination locations form a cluster. If their origins are close to each other, they can be transported together to a transshipment node in order to be assigned to distinct vehicles. Likewise, if their destinations are close, they can be serviced by several vehicles in the first part of their journey and then consolidated into a single vehicle at a transshipment location for the final delivery. The root node of the cluster is selected randomly. Then, all trips are sorted from closest to farthest from the root node with regard to either the origin or the destination of the corresponding requests. Finally, n trips are randomly selected to be removed, whereby the probability of being selected decreases with the distance from the root node. We refer to Masson et al. (2013) for a complete description of the heuristic.
5. *Many-split removal.* This heuristic aims at removing requests that are split many times in the current solution. The general idea is that such requests might be reinserted with possibly fewer trips, thereby creating new, perhaps better, solutions. First, all requests are listed in descending order with respect to the number of times they are split in the current solution (i.e. the number of trips belonging to the same request). Then, we iteratively select a random request from this list and remove all of its trips from the current solution. We do this until the number of removed trips is greater than or equal to n. The probability for a request to be selected increases with the number of splits. Algorithm 2 shows the *many-split removal* heuristic in pseudocode. A determinism parameter $p \geqslant 1$ is used to control the degree of randomization of the request selection (a low value of p corresponds to a high degree of randomness).

Algorithm 2: Many-split removal

```
Input: current solution $s$, $n \in \mathbb{N}$, $p >= 1$
1 $m \leftarrow 0$;
2 List $L \leftarrow$ All planned requests $r$, sorted by descending $nbrTrips(r, s)$;
3 while $m < n$ do
4     select a random number $y$ in the interval $[0, 1)$;
5     request $i \leftarrow L[y^p|L|]$;
6     $m \leftarrow m + nbrTrips(i, s)$;
7     remove all trips of request $i$ from solution $s$;
```

4.3. Repair Heuristics

All repair heuristics iteratively reinsert the previously removed demand into the solution. They stop when all requests are fully serviced or none can be inserted anymore. In all heuristics, in order to facilitate split loads, the amount of load of a request considered for reinsertion d^{insert} is calculated as the minimum of the maximum time-feasible amount, the maximum capacity-feasible amount and the currently still uncovered demand. It can thus take several iterations to fully reinsert a destroyed request. Whenever insertion cost are calculated they are normalized with the considered amount of load, i.e. insertion cost $C = (c^{travel} + c^{transship})/d^{insert}$.

Because all but one destruction heuristics are not guaranteed to remove all trips of a request from the solution, a destroyed solution can have both fully uncovered requests (i.e., no trip in the destroyed solution services the request) and partially uncovered requests (i.e., one or more trips in the destroyed solution still service the request). For all partially uncovered requests, it may be feasible to increase the load of a still planned trip. Increasing the load of a trip does not increase the travel cost, only the transshipment cost. Therefore, before calling a repair heuristic, we first (try to) increase the load of the still scheduled trips for all partially uncovered requests.

We implemented seven repair heuristics. The first three are again general purpose heuristics from the literature, which can be applied to a variety of different problem types and are not capable of handling transshipments. The remaining four heuristics are specifically designed for handling transshipments. Again, we adapted all previously proposed heuristics from the literature to the concept of trips.

1. *Best insertion*. In each iteration, the best insertion cost is computed for each (partially) uncovered request. The request with the smallest insertion cost is inserted at its best position (Ropke and Pisinger, 2006).
2. *Regret insertion*. This heuristic is based on the concept of regret. Let U be the set of (partially) unplanned requests and let Δf_r^i denote the insertion cost of the request $r \in U$ in the ith best route at its best position. In each iteration, the heuristic selects the request r^* for insertion such that $r^* = argmax_{r \in U}(\sum_{i=2}^{k} \Delta f_r^i - \Delta f_r^1)$. The request is inserted at its best position. Ropke and Pisinger (2006) give a complete description of this heuristic. We consider four regret-k heuristics with $k \in \{2, 3, 4, 5\}$.
3. *Random best insertion*. In each iteration, a randomly selected (partially) uncovered request is inserted at its best position.
4. *Best insertion with transshipment*. This heuristic is inspired by the *best insertion with transfer* heuristic proposed by Masson et al. (2013). First, the best insertion without transshipment is evaluated. Then, we evaluate the insertion with transshipment as described in Algorithm 3 (for simplicity we omit the duplication of the transshipment nodes in the notation of a trip). For each (partially) uncovered request (r^+, r^-), the algorithm considers r^+ as the root node from which it constructs a tree of trips which visit at least one and at most $|T|$ intermediary transshipment locations and end at r^-. Each trip is extended as long as it has not reached r^- and its cost is less than the cost of the current best trip. At the end, the algorithm returns the trip that represents the best insertion with transshipment. Finally, the better insertion among the one with transshipment and the one without transshipment is performed.
5. *Transshipment first insertion*. This heuristic is an adaptation of the *transfer first* heuristic proposed by Masson et al. (2013). The underlying idea is that often a transshipment location becomes beneficial only when two ore more requests are transshipped there. The *best insertion with transshipment* might therefore miss good opportunities to use a transshipment location since it evaluates the insertion of each request separately. *Transshipment first insertion* aims at avoiding this by prioritizing the use of transshipment locations. As long as there are (partially) uncovered requests, *best insertion with transshipment* without the first step is performed (i.e., without evaluation of the best insertion without transshipment). Only if no feasible insertion with transshipments can be performed, insertion without transshipment is considered.
6. *Most constrained insertion*. This heuristic is inspired by the operator *mostConstrainedRepair* proposed by Danloup et al. (2018). It is based on the idea that the ease with which a request can be inserted differs from request to request. For example, if a request has tight time windows, a large distance between its origin and destination and a large demand, it will be difficult to insert. Thus, the aim is to insert difficult requests first. Let us denote with Z_r the insertion ease of request r, and compute it as
$$Z_r = \gamma_1 c_{(r^+, r^-)} - \gamma_2 (b_{r^+} - a_{r^+} + b_{r^-} - a_{r^-}) + \gamma_3 d_r.$$
γ_1, γ_2 and γ_3 are weights for each term of the equation. In each iteration, first, all requests $r \in U$ are listed in descending order of Z_r. Then, a request r^* is randomly selected from the list in the same way as in the *many-split removal* heuristic, i.e. the probability of being selected increases with the value of Z_r. Finally, r^* is inserted with transshipments at its best position. If an insertion with transshipment is not feasible, insertion without transshipment is considered.
7. *Random best insertion with transshipment*. In each iteration, a randomly selected (partially) uncovered request is inserted with transshipment at its best position.

Algorithm 3: Cost evaluation algorithm for insertion with transshipment

```
    Input: set of uncovered requests $U$, set of transshipment locations $T$
1   stack of trips $S \leftarrow$ empty stack, best trip $\mathcal{P}^{\text{best}} \leftarrow$ empty trip, best cost $u \leftarrow \infty$;
2   foreach uncovered request $(r^+, r^-) \in U$ and $t \in T$ do
3       evaluate insertion of pair $(r^+, t)$, and consider it as inserted at its best position;
4       push $(r^+, t)$ onto the stack of trips $S$;
5       while $S$ not empty do
6           trip $\mathcal{P} \leftarrow$ top($S$), and pop($S$);
7           if $C(\mathcal{P}) < u$ then
8               if $\mathcal{P}$ ends at $r^-$ then
9                   $\mathcal{P}^{\text{best}} = \mathcal{P}$ and $u = C(\mathcal{P})$;
10              else
11                  foreach $t' \in \{\bar{t} \in T \mid \bar{t} \notin \mathcal{P}\} \cup \{r^-\}$ do
12                      extend $\mathcal{P}$ with $t'$ to obtain trip $\mathcal{P}' = (\mathcal{P}, t')$, and push it onto $S$;
13  return $\mathcal{P}^{best}$
```

After every insertion heuristic we search in all routes of the new solution for visits to the same origin-destination pair (r^+, r^-) appearing in the order $r^+ \rightarrow r^+ \rightarrow r^- \rightarrow r^-$ (with possible visits to other customers in between). Any of the visits may in fact also be to a transshipment location instead of an origin or a destination (for an example see request A in route 2 in Fig. 1). If this situation occurs, the two trips may be merged as one single trip while maintaining feasibility and improving the travel cost. There are four possible combinations of the two previous pickups and deliveries for the merged trip; the best one in terms of cost is chosen.

4.4. Initial Solution, Selection of Heuristics, Hcceptance and Stopping Criteria

To generate an initial solution from scratch, we make use of the above described insertion heuristics. We start with *best insertion*. If it does not yield a feasible solution, we start again from an empty solution and use the next heuristic in the list (*regret insertion*). We continue down the list until we find a feasible solution or all heuristics have been tried; in which case we use the solution with the least amount of uncovered demand as the initial solution.

In each iteration of the LNS we randomly select one destruction heuristic and one repair heuristic according to a uniform probability distribution. The newly found solution s' is accepted if $s' < (1+\chi) \cdot s_{best}$, where χ is a parameter representing the maximum degree of deterioration such that s' is still accepted (Danloup et al., 2018). A value greater than 0 allows the deterioration of s', which is useful in order to escape local minima. We set $\chi = 0.01$ in our implementation. The algorithm stops when a certain amount of time has passed. We denote with τ the maximum run time in seconds.

4.5. Penalty and Noise

It is possible that not all removed requests can be completely reinserted by the repair heuristic. In this case we add a penalty to the cost of the solution and allow it to be accepted as the new incumbent solution. Accepting intermediate infeasible solutions helps diversify the search. The penalty $\mathscr{P}$ is given by

$$\mathscr{P} = (\zeta / \sum_{r \in R} d_r)\mu + \zeta^{\frac{\pi}{\tau}},$$

where ζ is the cost of the initial solution, μ is the amount of unsatisfied demand in the current solution and π is the current run time in seconds.

Furthermore, in order to counter the myopia of the repair heuristics we randomly add noise to the insertion cost (like in Ropke and Pisinger, 2006). In every iteration of the LNS we randomly decide whether noise is added or not with an equal probability for each case. In the case when noise is added, whenever we calculate the cost C of inserting a trip into a route, we additionally calculate a random number φ in the interval $[-\Phi, \Phi]$ and compute the modified insertion cost $C' = max(0, C + \varphi)$. Like Ropke and Pisinger (2006) we relate the magnitude of noise to the characteristics of the problem instance and compute $\Phi = \eta \cdot max_{u,v \in V}(c_{(u,v)})$, where η is a parameter which controls the amount of noise.

4.6. Limiting the Number of Considered Transshipment Locations

For problems with a large number of transshipment locations, considering each of these locations in the insertion heuristics is time consuming. We therefore limit for each request the number of transshipment locations that will be considered for the first transshipment. For that purpose, we generate for each request $r \in R$ a set of potential transshipment locations $T'_r \subseteq T$. We then replace T with T'_r in line 2 in Algorithm3. The set contains the ρ closest transshipment locations of both the origin and the destination of request r. Note, that $\rho \leq |T'_r| \leq min(2\rho, |T|)$, since the set of the ρ closest transshipment locations of the origin and the set of the ρ closest transshipment locations of the destination may not be disjoint.

4.7. Alternating between Split Loads and Transshipments

Considering split loads and transshipments simultaneously significantly enlarges the solution space compared to considering only one or the other. This translates to longer run times and a worse solution quality of the insertion heuristics. We therefore employ a solution strategy where we alternate between considering only split loads and considering only transshipments. We start with considering split loads and switch after every 1/10 of total run time.

5. Computational experiments

In this section, we summarize the computational experiments that we performed to assess the performance of our solution approach and the benefits of considering split loads and transshipment simultaneously. The LNS is implemented in C++ and each experiment was executed on a single thread and performed on an Intel Xeon E5-2650 v2 2.6 GHz CPU with 4 GB of RAM. Table 1 summarizes the parameter values for the LNS, which were determined in the course of preliminary experiments.

In order to evaluate the performance of the LNS we solve benchmark instances from the literature and compare our results with the results of the current state-of-the-art solution approach. As already mentioned in Section 1, we are not aware of any heuristic solution approach which combines split loads and transshipments for the PDP. We therefore compare the performance of the LNS with that of the state-of-the-art solver for the PDPSL and the state-of-the-art solver for the PDPT separately.

In the following three subsections we first present the instances which we use for the computational experiments (Section 5.1). We then evaluate the performance of the LNS (Section 5.2). Finally, in Section 5.3 we analyze the benefits of combining split loads and transshipments.

5.1. Test Instances

We evaluate the LNS on the PDPSL benchmark instances from Haddad et al. (2018) and on the PDPT benchmark instances from Qu and Bard (2012).

Table 1
Parameter values for the LNS.

Parameter	Value	Description
α	0.1	Min. percentage of trips removed in each iteration of the LNS
β	0.25	Max. percentage of trips removed in each iteration of the LNS
γ_1	0.5	Weight of distance-term for *most constrained insertion*
γ_2	0.2	Weight of time-term for *most constrained insertion*
γ_3	0.3	Weight of demand-term for *most constrained insertion*
ν_1	9	Weight of distance-term for *related removal*
ν_2	3	Weight of time-term for *related removal*
ν_3	2	Weight of demand-term for *related removal*
η	0.025	Noise control parameter
p	7	Determinism parameter
χ	0.01	Max. degree of deterioration for new solution s' to be accepted
ρ	2	Number of closest transshipment locations considered as potential locations for the first transshipment

Table 2
Comparison with Haddad et al. (2018).

Size	Gap-Best (%)	Gap-Avg (%)	# New Best	Time (s)
10	0.00	−0.01	0	36
15	−0.16	−0.47	2	36
20	−0.11	−1.37	2	36
25	−1.51	−2.23	5	36
75	−3.14	−4.11	15	918
100	−1.62	−2.24	15	2023
125	−0.53	−1.18	11	3452
Avg.	−1.01	−1.66		

The instances proposed by Haddad et al. (2018) are divided into two sets. The first set contains 40 small instances: 10 instances with 10, 15, 20 and 25 requests, respectively. For twenty instances from this set the optimal solution value is known. The second set corresponds to the 45 medium-size instances from Nowak et al. (2008). In all instances a distance constraint is considered – 300 for the first set and 1000 for the second set. We use the fact that none of the instances consider time windows to account for the distance constraints. That is, we set the travel times between the nodes equal to the distances between the nodes, and set the time window of the depot to $[0, 300]$ in each instance of the first set and to $[0, 1000]$ in each instance of the second set.

The instance set proposed by Qu and Bard (2012) is divided into five subsets named “pdpt1” to “pdpt5”. Each subset contains 10 instances, where each instance contains 25 requests, one transshipment location and one depot. The instances were created in such a manner that the optimal solution is known and all instances of a subset have the same optimal solution value.

To evaluate the LNS with respect to combining split loads and transshipments, and to analyze the benefits of combining split loads and transshipments, we generated a new set of random instances. This new set of instances is based on real-world data which we determined in collaboration with an Austrian logistics service provider. The instances are intended to mimic long-distance transportation between and within countries along the river Danube. For that purpose we generated a set of 55 customer locations. Each of these locations corresponds to a random address in the industrial area of a major city from Austria, Bulgaria, Hungary, Romania, Serbia or Slovakia. We selected five locations from Serbia and ten locations from each of the other five countries. The origin and the destination location for each request are randomly selected from this set. The demand of each request is randomly generated with a uniform probability distribution in $[5, 20] \cap \mathbb{N}$. Each instance considers a period of 30 days. The time windows of each request are placed randomly within this interval. Each time window has a width of either 24, 72, or 120 hours; chosen at random with an equal probability for each width-size. We generated 20 instances for each of the following size categories (i.e. number of requests): 25, 50, 75 and 100. Each of these 80 instances contains five transshipment locations; which correspond to five major ports along the river Danube. All instances consider two types of vehicles: trucks and ships. The capacity for trucks is 20. For ships it is 1000. The travel cost is 0.73 €/km for trucks and 43.76 €/h for ships (it is customary to express travel cost for water vehicles in monetary units per time unit). The distance- and travel time-matrices for trucks were generated with the software MS MapPoint 2013; using average velocities of 65 km/h, 60 km/h and 20 km/h for motorways, highways and city, respectively. The travel time matrices for ships were calculated with the online travel time calculator https://www. danube-logistics.info/travel-time-calculator/. In each instance, the transshipment cost are assumed to be 3 €/unit at all transshipment locations. All instances are available at https://dx.doi.org/10.17632/gks43g58mw. In the following, we refer to these instances as PDPTWSLT-instances.

5.2. Evaluating the Performance of the LNS

In order to evaluate the performance of the LNS with respect to split loads, we compare our results on the instances from Haddad et al. (2018) with the results obtained with the ILS proposed in Haddad et al. (2018). We do not allow transshipments when solving these instances. To make a fair comparison, we account for the difference in the computing power of our CPU and the one used in Haddad et al. (2018). As their CPU (Intel Core2 Quad Q6600 2.4 GHz) is only ca. 60% as efficient as ours[1], we set the time limit for the LNS equal to 60% of the CPU time alloted to the ILS. We perform twenty runs for each instance (as in Haddad et al., 2018). For the comparison we calculate for each instance the percentage gap between the average cost found with the LNS and the average cost found with the ILS ($\text{gap} = 100 \cdot (z_{LNS} - z_{ILS})/z_{ILS}$, where z denotes the solution value). Additionally, we calculate the percentage gap between the best cost found by both methods.

Table 2 presents both gaps averaged over all instances belonging to the same size category. (Individual results for each instance are presented in Tables 6 and 7.) The results show that the LNS performs well. It was able to find better average cost for 71 (out of the 85) instances. Considering the gaps regarding the average costs (column *Gap-Avg(%)*), we observe negative values for each instance set. This means that the average solution quality of the LNS is better than the one from the ILS. With regard to the best cost (out of 20 runs), the LNS was able to find better solutions for 50 instances (see column *# New Best*); with improvements of up to 8%. Furthermore, the LNS found all 20 known optimal solutions in at least one run, whereas the ILS found only 18. Additionally, the LNS found 13 of the known optimal solutions in all 20 runs.

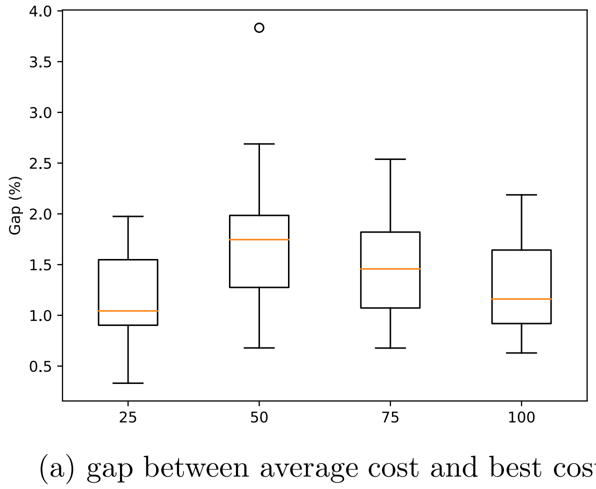
We performed a paired t-test comparing the solution values for each instance in order to validate the statistical significance of the results. The test yielded a value $p < 1.7 \cdot 10^{-10}$ for the best cost and a value $p < 1.4 \cdot 10^{-15}$ for the average cost, which indicates a significant difference in performance. We also conducted a Wilcoxon signed-rank test which supports the existence of significant differences between the two methods. The p-values were $p < 3.3 \cdot 10^{-9}$ for the best cost and $p < 2.4 \cdot 10^{-13}$ for the average cost.

In order to evaluate the performance of the LNS with respect to transshipments, we compare our results on the instances from Qu and Bard (2012) with the results obtained with the two solution methods – LNS and GA – proposed by Danloup et al. (2018). We do not allow split loads when solving these instances. As the CPU used in Danloup et al. (2018) (Intel Core i7-5700HQ 2.70GHz) is similar to ours, we use the same amount of CPU time as reported in the article. Like in Danloup et al. (2018), we perform thirty runs for each instance and use the cost of the best run to calculate the average cost for each instance set (pdpt1 to pdpt5).

[1] https://www.cpubenchmark.net/compare/Intel-Core2-Quad-Q6600-vs-Intel-Xeon-E5-2650-v2-vs-Intel-i7-5700HQ/1038vs2042vs2533. Accessed on March 13, 2020.

Table 3
Comparison with Danloup et al. (2018) on the instances from Qu and Bard (2012).

Set	Cost			Gap (%)			Time (s)
	Opt.	Danloup et al.	LNS	Avg.-Opt.	Max.-Opt.	Danloup et al.	
pdpt1	2510.94	2510.94	2510.94	0.00	0.00	0.00	31
pdpt2	2359.01	2360.60	2360.65	0.07	0.70	0.00	85
pdpt3	2384.84	2386.70	2403.31	0.77	1.24	0.70	35
pdpt4	2405.80	2405.80	2405.80	0.00	0.00	0.00	29
pdpt5	3479.03	3479.03	3479.03	0.00	0.00	0.00	87
Avg.				0.17	0.39	0.14	



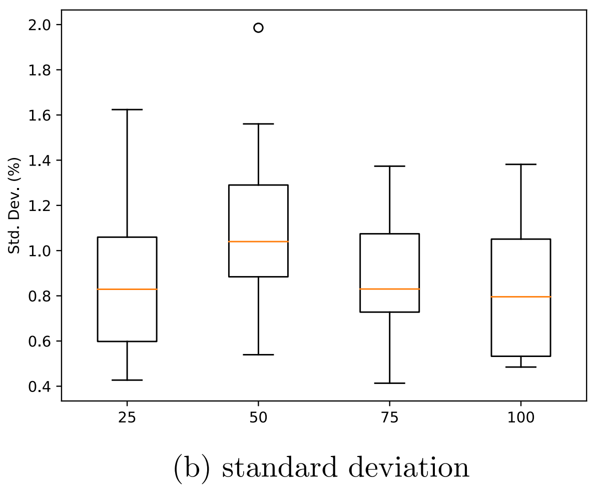


Fig. 2. Percentage gap between the average cost and the best cost (2a) and standard deviation in percent of the average cost (2b) for each instance. One boxplot for all instances of the same size.

In Table 3 we compare the average cost found by our LNS with the best average cost found in Danloup et al. (2018) (either by the GA or by the LNS). For each instance set, we set the run time of our LNS equal to the run time of the better performing algorithm in terms of cost, or, in case both methods perform equally good, to the method which is faster. Additionally, Table 3 presents for each instance set the optimal solution cost (column *Opt.*) as well as the average and maximum percentage gap between the optimal cost and the cost found with the LNS (columns *Avg.-Opt.* and *Max.-Opt.*, respectively). The results show that the LNS also performs well regarding transshipments. Like Danloup et al. (2018), it finds the optimal solution for each instance in the sets pdpt1, pdpt4 and pdpt5. For the sets pdpt2 and pdpt3 the percentage gap between the average cost found with the LNS and the average cost found in Danloup et al. (2018) (column *Danloup et al.*) is $\sim 0.0\%$ and 0.7%, respectively. Furthermore, the gap between the optimal cost and the cost found with the LNS is less than 1% on average and maximally 0.7% respectively 1.24% for an individual instance in those two sets.

Table 4
Average percentage cost increase when the solution strategy from Subsection 4.7 is not used.

Size	Gap-Best (%)	Gap-Avg (%)
25	−0.99	−0.34
50	0.31	1.07
75	1.37	2.71
100	3.07	6.65
Average	0.94	2.52

From the above results we conclude that the solution quality of our LNS is comparable with that of the current state-of-the-art solution approach for split loads and for transshipments, respectively. In the following we present several experiments on the PDPTWSLT-instances in order to evaluate the performance of the LNS with respect to considering split loads and transshipments simultaneously. In all experiments we perform ten runs for each instance.

We first evaluate the stability of the LNS. For that purpose we calculate for each instance the percentage gap between the average cost and the best cost found by the LNS, and the standard deviation expressed in percent of the average cost. Fig. 2 shows boxplots of the percentage gaps (Fig. 2a) and and the standard deviation (Fig. 2b); one boxplot for each set of instances with the same number of requests. The LNS appears to be quite stable. For 90% of the instances the gap between the average solution and the best solution is less than 2%, and the standard deviation is less than 1.4% of the average cost. To provide a base for comparisons with future algorithms, Tables 8–11 show the results for each instance individually.

In the next set of experiments we evaluate the impact of the solution strategy described in SubSection 4.7, i.e. alternating between considering only split loads and considering only transshipments. For that purpose we calculate for each instance the percentage gap between the average cost found with the strategy and the average cost found without the strategy ($\text{gap} = 100 \cdot (z_{w/o} - z_w)/z_w$, where z_w and $z_{w/o}$ denote the solution value with and without the strategy, respectively). We also calculate the percentage gap between the best cost found with and without the strategy. Table 4 presents both gaps averaged over all instances belonging to the same size category. The results show that for small instances

Table 5
Average percentage cost decrease of the PDPTWSLT solutions compared to the PDP, PDPSL, and PDPT solutions

Size	PDP	PDPSL	PDPT
25	8.44	5.11	3.41
50	8.19	4.50	2.57
75	4.39	1.32	2.47
100	3.55	0.93	2.69
Average	6.14	2.96	2.78

not using the strategy actually leads to better average solutions and better best solutions (although only minimally better), on average. This is not surprising given the still rather small solution space induced by these instances. For larger instances, on the other hand, not using the strategy leads, on average, to up to 3% higher best cost and up to more than 6% higher average cost.

We now measure the performance of the destroy and insertion heuristics. For that purpose, we compute the number of times a heuristic improved the incumbent solution and the number of times it improved the best solution. Both indicators are expressed as the percentage over the number of times the heuristic was used. Figs. 3 and 4 show the values of both indicators for the insertion heuristics and the destroy heuristics, respectively. The values are averages over all PDPTWSLT-instances. The best insertion heuristics are the *regret-k insertion* heuristics, with smaller values of *k* correlating with larger values for both indicators. Then, *random best insertion* performs better than *best insertion with transshipment*, followed by *most constrained insertion*, *transshipment first insertion*, and *random best insertion with transshipment*. Interestingly, *best insertion* performs the worst with respect to improving the incumbent solution, but performs second best with respect to improving the best solution. The insertion heuristics which consider transshipments perform considerably worse than the insertion heuristics which do not consider transshipments. This is to

Table 6
Individual results for the instances from Haddad et al. (2018) – 1/2. Optimal solution values are underlined in the column *Best-20* of the LNS.

Instance	ILS		LNS		Gap-Best (%)	Gap-Avg (%)	Time (s)
	Best-20	Avg-20	Best-20	Avg-20			
10-1	953.00	953.00	953.00	953.00	0.00	0.00	36
10-2	1020.00	1021.80	1020.00	**1020.45**	0.00	−0.13	36
10-3	968.00	970.70	968.00	**968.50**	0.00	−0.23	36
10-4	979.00	**979.00**	979.00	981.40	0.00	0.25	36
10-5	1026.00	1026.00	1026.00	1026.00	0.00	0.00	36
10-6	742.00	742.00	742.00	742.00	0.00	0.00	36
10-7	1019.00	1019.00	1019.00	1019.00	0.00	0.00	36
10-8	818.00	818.00	818.00	818.00	0.00	0.00	36
10-9	765.00	765.00	765.00	765.00	0.00	0.00	36
10-10	1058.00	1058.00	1058.00	1058.00	0.00	0.00	36
15-1	1346.00	1379.20	1346.00	**1346.00**	0.00	−2.41	36
15-2	1116.00	**1116.00**	1116.00	1119.25	0.00	0.29	36
15-3	1184.00	1194.60	**1180.00**	**1181.70**	−0.34	−1.08	36
15-4	1297.00	1302.60	1297.00	**1297.00**	0.00	−0.43	36
15-5	1028.00	1028.00	1028.00	1028.00	0.00	0.00	36
15-6	1155.00	1157.80	**1140.00**	**1140.00**	−1.30	−1.54	36
15-7	1102.00	**1116.90**	1102.00	1120.60	0.00	0.33	36
15-8	1145.00	1145.30	1145.00	**1145.00**	0.00	−0.03	36
15-9	1150.00	**1154.40**	1150.00	1156.65	0.00	0.19	36
15-10	1060.00	1060.00	1060.00	1060.00	0.00	0.00	36
20-1	1350.00	1351.10	1350.00	**1350.50**	0.00	−0.04	36
20-2	1454.00	1457.90	1454.00	**1456.10**	0.00	−0.12	36
20-3	861.00	**862.20**	861.00	863.10	0.00	0.10	36
20-4	1348.00	1350.80	1348.00	**1348.50**	0.00	−0.17	36
20-5	1327.00	1337.00	**1314.00**	**1316.85**	−0.98	−1.51	36
20-6	1614.00	1625.30	1614.00	**1617.15**	0.00	−0.50	36
20-7	1399.00	1399.50	1399.00	**1399.00**	0.00	−0.04	36
20-8	1299.00	1329.30	1299.00	**1299.00**	0.00	−2.28	36
20-9	1469.00	1588.00	**1467.00**	**1475.05**	−0.14	−7.11	36
20-10	1429.00	1460.50	1429.00	**1431.10**	0.00	−2.01	36
25-1	1887.00	1908.80	**1787.00**	**1875.80**	−5.30	−1.73	36
25-2	1692.00	1748.40	1692.00	**1692.15**	0.00	−3.22	36
25-3	1549.00	1555.20	**1545.00**	**1545.00**	−0.26	−0.66	36
25-4	1675.00	1685.50	1675.00	**1675.00**	0.00	−0.62	36
25-5	1415.00	1428.90	**1400.00**	**1402.35**	−1.06	−1.86	36
25-6	1882.00	1923.30	**1876.00**	**1876.00**	−0.32	−2.46	36
25-7	1487.00	1604.30	1487.00	**1569.40**	0.00	−2.18	36
25-8	1429.00	1446.60	1429.00	**1429.00**	0.00	−1.22	36
25-9	1613.00	1616.10	1613.00	**1613.75**	0.00	−0.15	36
25-10	1802.00	1802.60	**1654.00**	**1654.00**	−8.21	−8.24	36
10	934.80	935.25	934.80	935.13	0.00	−0.01	36
15	1158.30	1165.48	1156.40	1159.42	−0.16	−0.47	36
20	1355.00	1376.16	1353.50	1355.63	−0.11	−1.37	36
25	1643.10	1671.97	1615.80	1633.25	−1.51	−2.23	36

Table 7
Individual results for the instances from Haddad et al. (2018) – 2/2.

Instance	ILS		LNS		Gap-Best (%)	Gap-Avg (%)	Time (s)
	Best-20	Avg-20	Best-20	Avg-20			
75-1A	3782.93	3865.12	**3686.03**	**3724.47**	−2.56	−3.64	918
75-1B	3747.57	3836.27	**3672.05**	**3706.44**	−2.02	−3.38	918
75-1C	3793.62	3864.07	**3645.50**	**3708.30**	−3.90	−4.03	918
75-1D	3765.05	3835.80	**3657.81**	**3705.95**	−2.85	−3.39	918
75-1E	3757.36	3835.99	**3633.93**	**3685.69**	−3.29	−3.92	918
75-2A	3097.22	3200.18	**3051.87**	**3084.09**	−1.46	−3.63	918
75-2B	3123.89	3181.35	**3008.21**	**3059.79**	−3.70	−3.82	918
75-2C	3110.82	3174.16	**3026.04**	**3063.69**	−2.73	−3.48	918
75-2D	3097.16	3163.17	**3008.85**	**3050.54**	−2.85	−3.56	918
75-2E	3118.38	3192.46	**3014.42**	**3064.96**	−3.33	−3.99	918
75-3A	3866.08	3981.00	**3695.69**	**3783.92**	−4.41	−4.95	918
75-3B	3855.72	3976.12	**3702.67**	**3763.67**	−3.97	−5.34	918
75-3C	3886.66	3959.61	**3713.00**	**3751.04**	−4.47	−5.27	918
75-3D	3870.89	3944.61	**3743.71**	**3772.16**	−3.29	−4.37	918
75-3E	3828.10	3958.88	**3740.23**	**3767.95**	−2.30	−4.82	918
100-1A	4920.25	4993.39	**4807.25**	**4876.65**	−2.30	−2.34	2023
100-1B	4940.53	5029.61	**4899.12**	**4949.67**	−0.84	−1.59	2023
100-1C	4903.04	5010.18	**4808.91**	**4896.09**	−1.92	−2.28	2023
100-1D	4928.88	5012.28	**4865.01**	**4916.60**	−1.30	−1.91	2023
100-1E	4869.26	4977.90	**4846.78**	**4882.61**	−0.46	−1.91	2023
100-2A	4212.57	4270.83	**4134.30**	**4203.96**	−1.86	−1.57	2023
100-2B	4226.56	4304.39	**4189.16**	**4241.77**	−0.88	−1.45	2023
100-2C	4213.68	4281.58	**4107.44**	**4211.96**	−2.52	−1.63	2023
100-2D	4217.27	4305.97	**4150.75**	**4204.43**	−1.58	−2.36	2023
100-2E	4186.25	4275.55	**4152.48**	**4200.73**	−0.81	−1.75	2023
100-3A	4982.29	5131.18	**4879.75**	**4986.57**	−2.06	−2.82	2023
100-3B	5057.84	5189.52	**4960.68**	**5022.65**	−1.92	−3.22	2023
100-3C	5031.36	5137.47	**4924.20**	**4996.53**	−2.13	−2.74	2023
100-3D	5049.84	5152.46	**4954.21**	**4998.15**	−1.89	−2.99	2023
100-3E	5029.86	5144.86	**4940.47**	**4991.81**	−1.78	−2.97	2023
125-1A	5794.79	5888.54	**5748.16**	**5810.43**	−0.80	−1.33	3452
125-1B	5880.96	5966.92	**5760.58**	**5882.34**	−2.05	−1.42	3452
125-1C	**5738.87**	5881.25	5752.58	**5822.43**	0.24	−1.00	3452
125-1D	**5738.32**	5945.99	5783.52	**5857.76**	0.79	−1.48	3452
125-1E	5822.20	5915.08	**5720.27**	**5821.15**	−1.75	−1.59	3452
125-2A	5310.49	5419.77	**5303.54**	**5377.26**	−0.13	−0.78	3452
125-2B	5361.25	5476.77	**5348.33**	**5431.01**	−0.24	−0.84	3452
125-2C	5357.90	5417.37	**5298.69**	**5399.25**	−1.11	−0.33	3452
125-2D	**5331.69**	5458.60	5376.67	**5442.44**	0.84	−0.30	3452
125-2E	**5339.33**	5443.11	5345.52	**5410.80**	0.12	−0.59	3452
125-3A	6177.11	6283.43	**6138.31**	**6197.20**	−0.63	−1.37	3452
125-3B	6205.87	6343.14	**6181.91**	**6257.52**	−0.39	−1.35	3452
125-3C	6230.02	6312.84	**6091.37**	**6181.42**	−2.23	−2.08	3452
125-3D	6181.96	6351.26	**6164.50**	**6255.12**	−0.28	−1.51	3452
125-3E	6128.42	6313.40	**6107.23**	**6206.18**	−0.35	−1.70	3452
75	3580.10	3664.59	3466.67	3512.84	−3.14	−4.11	918
100	4717.97	4814.48	4641.37	4705.34	−1.62	−2.24	2023
125	5773.28	5894.50	5741.41	5823.49	−0.53	−1.18	3452

Table 8
PDPTWSLT results for the 25-requests instances. Time limit: 600 seconds per run.

Instance	Best-10	Avg-10	Gap (%)	Std. Dev. (%)
25-0	16495.03	16609.55	0.69	0.43
25-1	19439.89	19618.30	0.92	0.88
25-2	15496.32	15596.26	0.64	0.46
25-3	16677.02	16844.98	1.01	0.82
25-4	14972.80	15259.98	1.92	1.24
25-5	18608.09	18881.61	1.47	1.00
25-6	16306.76	16597.38	1.78	1.41
25-7	16085.89	16403.54	1.97	1.62
25-8	16466.43	16620.70	0.94	0.46
25-9	14791.78	14986.29	1.31	0.57
25-10	14441.86	14489.71	0.33	0.43
25-11	15463.16	15741.56	1.80	1.16
25-12	12752.46	12875.46	0.96	0.67
25-13	14571.89	14725.78	1.06	0.79
25-14	16741.92	16874.14	0.79	0.63
25-15	17115.21	17261.98	0.86	0.61

(*continued on next page*)

Table 8 (*continued*)

Instance	Best-10	Avg-10	Gap (%)	Std. Dev. (%)
25-16	14651.02	14834.12	1.25	1.45
25-17	17031.69	17207.26	1.03	0.84
25-18	14592.62	14755.40	1.12	0.91
25-19	15034.16	15318.12	1.89	1.03
Average			1.19	0.87

Table 9
PDPTWSLT results for the 50-requests instances. Time limit: 1800 seconds per run.

Instance	Best-10	Avg-10	Gap (%)	Std. Dev. (%)
50-0	28855.08	29269.12	1.43	1.02
50-1	28265.54	28715.45	1.59	1.14
50-2	31026.32	31607.04	1.87	1.10
50-3	29987.26	30793.59	2.69	1.56
50-4	26962.07	27291.42	1.22	0.67
50-5	27260.59	27521.19	0.96	0.91
50-6	26294.57	26806.98	1.95	1.02
50-7	31895.75	32182.21	0.90	0.54
50-8	28126.53	28812.17	2.44	1.36
50-9	29729.07	29982.60	0.85	0.57
50-10	31845.36	32061.51	0.68	0.59
50-11	28664.75	29170.09	1.76	0.93
50-12	29658.05	30235.68	1.95	0.83
50-13	30766.67	31164.41	1.29	0.90
50-14	30117.91	30638.74	1.73	1.43
50-15	27922.21	28440.47	1.86	1.06
50-16	28803.18	29405.79	2.09	1.28
50-17	32542.51	33081.56	1.66	1.20
50-18	29980.70	30683.53	2.34	1.33
50-19	27614.77	28673.46	3.83	1.99
Average			1.75	1.07

Table 10
PDPTWSLT results for the 75-requests instances. Time limit: 3600 seconds per run.

Instance	Best-10	Avg-10	Gap (%)	Std. Dev. (%)
75-0	45078.15	45625.94	1.22	0.70
75-1	41514.30	42061.97	1.32	0.82
75-2	43459.14	43787.33	0.76	0.41
75-3	41790.35	42493.11	1.68	0.72
75-4	40341.12	41067.71	1.80	0.95
75-5	42598.41	43330.35	1.72	1.11
75-6	40463.76	40909.85	1.10	0.84
75-7	39924.02	40678.81	1.89	1.31
75-8	44326.03	45180.02	1.93	1.07
75-9	38360.15	38794.67	1.13	0.77
75-10	43355.65	43682.77	0.75	0.63
75-11	38521.56	39075.99	1.44	0.81
75-12	43513.21	43879.72	0.84	0.73
75-13	36853.93	37215.87	0.98	0.89
75-14	39001.83	39577.45	1.48	1.14
75-15	38735.84	39719.09	2.54	1.37
75-16	36017.33	36720.81	1.95	1.08
75-17	44401.02	44701.87	0.68	0.43
75-18	41789.35	42484.56	1.66	0.90
75-19	38332.48	39052.50	1.88	0.78
Average			1.44	0.87

be expected, given the vastly larger solution space which is explored by the heuristics which consider transshipments. The best destruction heuristic is *many-split removal*, with a significantly better performance than the remaining destruction heuristics. *Cluster removal* performs second best, followed by *related removal*, *random removal*, and *worst removal*.

5.3. Evaluating the Benefits of Combining Split Loads and Transshipments

Now we analyze the benefits of combining split loads and transshipments. For that purpose we solve each PDPTWSLT-instance in three different variants: considering only split loads (i.e. PDPSL),

Table 11
PDPTWSLT results for the 100-requests instances. Time limit: 5400 seconds per run.

Instance	Best-10	Avg-10	Gap (%)	Std. Dev. (%)
100-0	51620.27	52109.31	0.95	0.77
100-1	52271.21	53170.23	1.72	1.21
100-2	56779.50	57136.75	0.63	0.48
100-3	47439.75	48099.32	1.39	0.85
100-4	57343.58	57931.50	1.03	0.78
100-5	60798.88	61782.59	1.62	0.81
100-6	53293.52	53925.92	1.19	1.05
100-7	51765.03	52676.23	1.76	1.00
100-8	52259.71	52665.24	0.78	0.51
100-9	47877.55	48201.48	0.68	0.63
100-10	53704.63	54879.20	2.19	1.07
100-11	55843.58	56571.96	1.30	0.54
100-12	58468.16	59011.66	0.93	0.51
100-13	56392.11	56892.61	0.89	0.52
100-14	52378.02	52972.45	1.13	1.06
100-15	56149.60	57330.00	2.10	1.38
100-16	56608.51	57482.87	1.54	1.00
100-17	58396.57	59055.36	1.13	0.66
100-18	52098.31	52498.46	0.77	0.49
100-19	54270.57	55355.47	2.00	1.29
Average			1.29	0.83

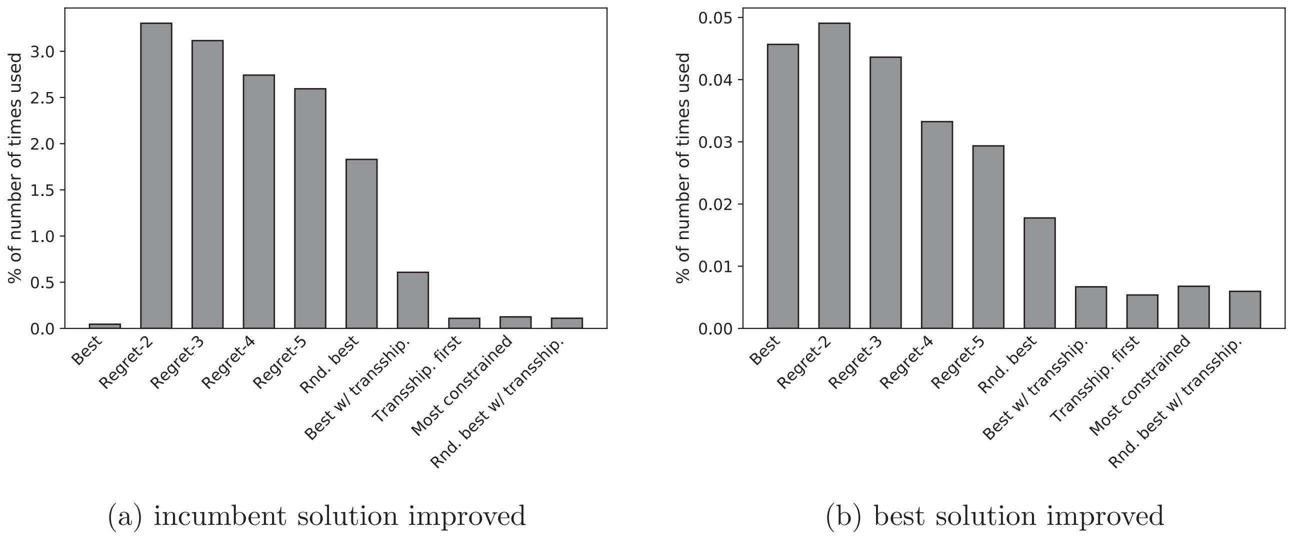


Fig. 3. Efficiency of the insertion heuristics.

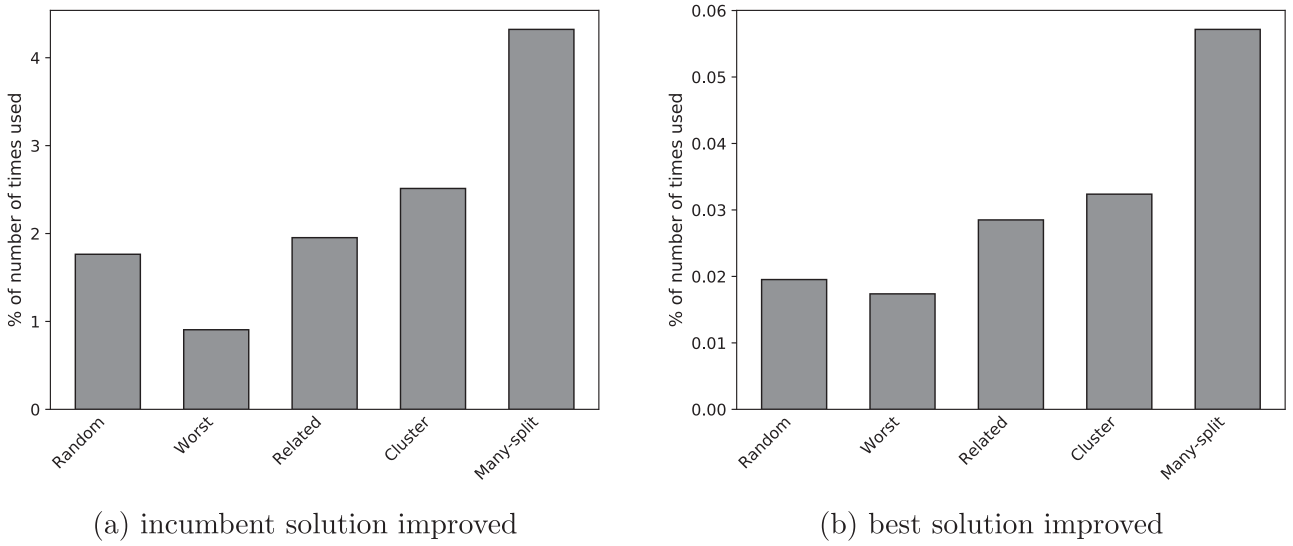


Fig. 4. Efficiency of the destroy heuristics.

considering only transshipments (i.e. PDPT), and considering neither (i.e. PDP). We perform 10 runs for each instance (and problem type) and use only the best run for the comparison. Table 5 shows for each set of instances the average percentage cost decrease of the PDPTWSLT solutions compared to the PDP, PDPSL, and PDPT solutions, respectively. The results show that considering split loads and transshipments simultaneously leads to cost savings of ca. 6%, on average, compared to considering neither. Compared to considering only split loads, or only transshipments, the average cost savings are only half as much; with a cost decrease of roughly 3% compared to both the PDPSL and PDPT solutions. It is worth mentioning here that, in general, the amount of achievable cost savings is very sensitive to the geographical distribution of the customers, their demands, the cost structure and capacity of the involved vehicles, and the magnitude of the transshipment cost. Thus, these results are valid only for PDPs with characteristics similar to the ones in our instances and cannot be generalized for all PDPs. Nevertheless, they still provide an indication of how much can be gained in practice.

The considerably smaller cost savings for the large instances are mainly due to the vast increase in solution space combined with a significant decrease in the number of iterations of the LNS. As a consequence, the LNS struggles to find good solutions which combine split loads and transshipments. These results, thus, implicitly also show the limit of the LNS and reflect the difficulty of the problem.

6. Conclusion

In this article we address the PDP with time windows, split loads and transshipments. We first present an arc-based mixed-integer formulation for it and then propose a large neighbourhood search algorithm for solving the problem. The LNS proposed in this article differs in several aspects from the solution approaches in the literature for the PDPSL and the PDPT, respectively. First, it is based on a novel solution representation which allows an efficient simultaneous handling of split loads and transshipments. Then, it is the first LNS addressing split loads in the context of the PDP. Additionally, it adopts a new strategy of handling split loads which does not require to limit the number of split loads nor solving computationally difficult and expensive problems for determining the allocation of (split) loads to vehicles. A new destruction heuristic specifically designed to address split loads is proposed as well. LNS has already been used in the literature to address transshipments in the context of the PDP (Qu and Bard, 2012; Masson et al., 2013; Danloup et al., 2018). However, contrary to those LNS algorithms, our LNS does not limit the number of transshipments per request to one. This allows for covering a larger variety of problems; such as multimodal transport, for example, where requests typically need to be transshipped at least twice.

The performance of the proposed LNS is validated through extensive computational experiments. It outperforms the current state-of-the-art solver for the PDPSL in comparable computational time on the multi-vehicle instances proposed in Haddad et al. (2018); and finds new best known solutions for 50 out of 85 instances. With respect to transshipments, the experiments show that the performance of the LNS is comparable with that of the current state-of-the-art solution approach for the PDPT. We evaluate the LNS with respect to combining split loads and transshipments on problem instances which are based on real-world data and mimic long-distance transportation. The results show that the algorithm is efficient to solve the PDPTWSLT. For future comparisons, we report individual solutions for each instance of this new benchmark collection. Additional experiments on these instances show the benefit of combining split loads and transshipments compared to considering only split loads, only transshipments, or neither. We observe cost reductions of roughly 6% on average compared to considering neither split loads nor transshipments, and cost reductions of roughly 3% on average compared to considering either only split loads or only transshipments, respectively. While those results are not generalizable, they still provide an indication of the achievable cost savings in practice.

This work suggests several improvements and directions for further research. With regard to the methodological aspect, solving the PDPTWSLT requires much more computational effort than solving the PDPSL, the PDPT, or the PDP. The majority of the time is spent on calculating insertion positions and performing tasks related to it (e.g. updating auxiliary data structures which help to speed up the feasibility check). Combining split loads and transshipments significantly increases the number of possible insertion positions. Furthermore, inserting a request into a route which contains a transshipment location not only affects the subsequent visits along this route, but also those along all routes that are (directly or even transitively) connected to this route through transshipment locations. This makes the feasibility check much more difficult, and consequently more time consuming. An interesting direction for further research is therefore the design of more powerful and smarter insertion heuristics and data structures which speed up the computation of (good) insertion positions and the feasibility check. The solution quality improvement achieved by the solution strategy described in Subsection 4.7 suggests another promising line of research, i.e. developing more sophisticated overall guiding strategies. With regard to the conceptual aspect, solving the PDPTWSLT-instances clearly showed the potential benefits of combining split loads and transshipments. A comprehensive analysis regarding favorable circumstances for split loads and/or transshipments would therefore be another research line of interest.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

David Wolfinger: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization, Project administration.

Acknowledgments

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged. We would also like to thank the three anonymous referees for their helpful comments.

References

Archetti, C., Bianchessi, N., Speranza, M.G., 2011. A column generation approach for the split delivery vehicle routing problem. Networks 58, 241–254.
Archetti, C., Bouchard, M., Desaulniers, G., 2011. Enhanced branch and price and cut for vehicle routing with split deliveries and time windows. Transp. Sci. 45, 285–298.
Archetti, C., Speranza, M.G., 2008. The split delivery vehicle routing problem: a survey. In: Golden, B., Raghavan, S., Wasil, E. (Eds.), The Vehicle Routing Problem: Latest Advances and New Challenges. Springer, US, Boston, MA, pp. 103–122.
Archetti, C., Speranza, M.G., 2012. Vehicle routing problems with split deliveries. Int. Trans. Oper. Res. 19, 3–22.

Bouros, P., Sacharidis, D., Dalamagas, T., Sellis, T., 2011. Dynamic pickup and delivery with transfers. In: Pfoser, D., Tao, Y., Mouratidis, K., Nascimento, M.A., Mokbel, M., Shekhar, S., Huang, Y. (Eds.), Advances in Spatial and Temporal Databases. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 112–129.

Cortés, C.E., Matamala, M., Contardo, C., 2010. The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. Eur. J. Oper. Res. 200, 711–724.

Danloup, N., Allaoui, H., Goncalves, G., 2018. A comparison of two meta-heuristics for the pickup and delivery problem with transshipment. Comput. Oper. Res. 100, 155–171.

Desaulniers, G., 2010. Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. Oper. Res. 58, 179–192.

Dror, M., Laporte, G., Trudeau, P., 1994. Vehicle routing with split deliveries. Discr. Appl. Math. 50, 239–254.

Dror, M., Trudeau, P., 1989. Savings by split delivery routing. Transp. Sci. 23, 141–145.

Dror, M., Trudeau, P., 1990. Split delivery routing. Naval Res. Logist. 37, 383–402.

Eurostat, 2020. Freight transported in containers - statistics on unitisation. online. https://bit.ly/2v6L6Bu. accessed February, 4th 2020..

Gendreau, M., Dejax, P., Feillet, D., Gueguen, C., 2006. Vehicle routing with time windows and split deliveries. Technical Report 851. Laboratoire Informatique d'Avignon, Avignon, France..

Haddad, M.N., Martinelli, R., Vidal, T., Martins, S., Ochi, L.S., Souza, M.J.F., Hartl, R., 2018. Large neighborhood-based metaheuristic and branch-and-price for the pickup and delivery problem with split loads. Eur. J. Oper. Res. 270, 1014–1027.

Kerivin, H., Lacroix, M., Mahjoub, A.R., Quilliot, A., 2008. The splittable pickup and delivery problem with reloads. Eur. J. Ind. Eng. 2, 112–133.

Masson, R., Lehuédé, F., Péton, O., 2013. An adaptive large neighborhood search for the pickup and delivery problem with transfers. Transp. Sci. 47, 344–355.

Masson, R., Lehuédé, F., Péton, O., 2014. The dial-a-ride problem with transfers. Comput. Oper. Res. 41, 12–23.

Mitrovic-Minic, S., Laporte, G., 2006. The pickup and delivery problem with time windows and transshipment. Infor 44, 217.

Neves-Moreira, F., Amorim, P., Guimarães, 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, 487–506.

Nowak, M., Ergun, Ö., White III, C.C., 2008. Pickup and delivery with split loads. Transp. Sci. 42, 32–43.

Nowak, M., Ergun, O., White III, C.C., 2009. An empirical study on the benefit of split loads with the pickup and delivery problem. Eur. J. Oper. Res. 198, 734–740.

Parragh, S.N., Doerner, K.F., Hartl, R.F., 2008. A survey on pickup and delivery problems. J. Betriebswirtschaft 58, 21–51.

Parragh, S.N., Doerner, K.F., Hartl, R.F., 2008. A survey on pickup and delivery problems. J. Betriebswirtschaft 58, 81–117.

Parragh, S.N., Pinho de Sousa, J., Almada-Lobo, B., 2015. The dial-a-ride problem with split requests and profits. Transp. Sci. 49, 311–334.

Pisinger, D., Ropke, S., 2019. Large neighborhood search. In: Gendreau, M., Potvin, J. Y. (Eds.), Handbook of Metaheuristics. third ed. Springer, pp. 99–127.

Qu, Y., Bard, J.F., 2012. A grasp with adaptive large neighborhood search for pickup and delivery problems with transshipment. Comput. Oper. Res. 39, 2439–2456.

Rais, A., Alvelos, F., Carvalho, M.S., 2014. New mixed integer-programming model for the pickup-and-delivery problem with transshipment. Eur. J. Oper. Res. 235, 530–539.

Ropke, S., Pisinger, D., 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. 40, 455–472.

Şahin, M., Çavuşlar, G., Öncan, T., Şahin, G., Aksu, D.T., 2013. An efficient heuristic for the multi-vehicle one-to-one pickup and delivery problem with split loads. Transp. Res. C Emerg. Technol. 27, 169–188.

Shaw, P., 1997. A new local search algorithm providing high quality solutions to vehicle routing problems. APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.F. (Eds.), Principles and Practice of Constraint Programming — CP98. Springer, Berlin Heidelberg, pp. 417–431.

Stålhane, M., Andersson, H., Christiansen, M., Cordeau, J.F., Desaulniers, G., 2012. A branch-price-and-cut method for a ship routing and scheduling problem with split loads. Comput. Oper. Res. 39, 3361–3375.

Toth, P., Vigo, D., 2014. Vehicle routing: problems, methods, and applications. SIAM.

Wolfinger, D., Salazar-González, J.J., 2020. The pickup and delivery problem with split loads and transshipments: a branch-and-cut solution approach. Eur. J. Oper. Res.. To appear.