Central European Journal of Operations Research (2021) 29:1115–1162
https://doi.org/10.1007/s10100-020-00690-w

ORIGINAL PAPER

On the one-to-one pickup-and-delivery problem with time windows and trailers

Michael Drexl[1]

Published online: 20 June 2020

Abstract

This paper studies an extension of the well-known one-to-one pickup-and-delivery problem with time windows. In the latter problem, requests to transport goods from pickup to delivery locations must be fulfilled by a set of vehicles with limited capacity subject to time window constraints. Goods are not interchangeable: what is picked up at one particular location must be delivered to one particular other location. The discussed extension consists in the consideration of a heterogeneous vehicle fleet comprising lorries with detachable trailers. Trailers are advantageous as they increase the overall vehicle capacity. However, some locations may be accessible only by lorries. Therefore, special locations are available where trailers can be parked while lorries visit accessibility-constrained locations. This induces a nontrivial tradeoff between an enlarged vehicle capacity and the necessity of scheduling detours for parking and reattaching trailers. The contribution of the paper is threefold: (i) it studies a practically relevant generalization of the one-to-one pickup-and-delivery problem with time windows. (ii) It develops an exact amortized constant-time procedure for testing the feasibility of an insertion of a transport task into a given route with regard to time windows and lorry and trailer capacities. (iii) It provides a comprehensive set of new benchmark instances on which the runtime of the constant-time test is compared with a naïve one that requires linear time by embedding both tests in an adaptive large neighbourhood search algorithm. Computational experiments show that the constant-time test outperforms its linear-time counterpart by one order of magnitude on average.

Keywords Vehicle routing · Pickup-and-delivery · Trailers · Insertion heuristic · Constant-time feasibility test

✉ Michael Drexl
michael.drexl@th-deg.de

[1] Faculty of Applied Natural Sciences and Industrial Engineering, Deggendorf Institute of Technology, Deggendorf, Germany

1 Introduction

The one-to-one pickup-and-delivery problem with time windows and trailers (PDPTWT) can be described as follows. There is a set of requests or tasks to transport specified amounts of goods between paired pickup and delivery locations. To fulfil the tasks, a set of capacitated vehicles consisting of *single lorries* and *lorry–trailer combinations (LTCs)* is available. Each vehicle has a given start and a given end location. The start location of a vehicle may differ from the vehicle's end location. A trailer has the same start and the same end location as its associated lorry. Each single lorry and each LTC has a fixed cost, incurred only if it fulfils at least one task, and a travel cost for moving from one location to another. Fixed and travel costs may differ between vehicles; for LTCs, travelling between two locations with the trailer attached may be more expensive than without. Capacities may also differ between vehicles. LTCs have a lorry capacity and a trailer capacity. After picking up and before delivering the goods of a certain task, vehicles may visit other pickup and/or delivery locations. All pickup and all delivery locations can be visited by a single lorry and by an LTC lorry without its trailer. However, some pickup and some delivery locations may have accessibility constraints in the sense that they cannot be visited by an LTC lorry when the trailer is coupled. Because of these accessibility restrictions, there are also *parking and transshipment locations (PTLs)*. At PTLs, trailers can be decoupled, parked, and re-coupled, and load can be transshipped between an LTC lorry and its trailer. In this paper, a fixed lorry–trailer assignment is assumed. This means that each trailer can be pulled only by one lorry, and only this lorry can transfer load to or from the trailer. All task locations, i.e., all pickup or delivery locations, can be visited by any lorry, all locations designated as reachable by trailer can be visited by any trailer, and PTLs can be visited by all LTC lorries and trailers. Each task location is visited exactly once, whereas PTLs can be visited more than once by the same or different LTCs. The load to be picked up at a task location can be split arbitrarily between a lorry and its trailer if the location is visited by an LTC.

Each location has a single, hard time window that may be equal to the length of the planning horizon and thus nonrestrictive. In practice, most PTL time windows are equal to the planning horizon, but there may be some PTLs with a restricted time window. Hence, time windows are also assigned to PTLs. Arrival at a location before the start of its time window is allowed and incurs waiting time but no cost. Waiting time is not limited. There are fixed service times at all task locations and all PTLs. At PTLs, there are two service times, one for the decoupling and one for the re-coupling operation. Travel times between locations and service times are independent of the current vehicle, of its current load and, for LTCs, of whether or not the trailer is attached. Travel and service times as well as fixed and travel costs are time-independent. All vehicles are available throughout the complete planning horizon.

An LTC route may visit any location and is partitioned into the *main route*, which is the part of the route where the lorry pulls its trailer, and zero or more *subroutes* that start and end at a PTL where the lorry parks its trailer while visiting one or more task locations. An LTC lorry may perform several consecutive subroutes starting and ending at the same PTL before finally pulling away its trailer. If a delivery location is visited on a subroute and the corresponding pickup location has been visited before this subroute, it must be ensured that the entire amount of goods bound for this delivery location is on the lorry at the start of the subroute. This may require a load transfer from a trailer to its lorry at a PTL.

There is no congestion at PTLs: arbitrarily many trailers can be parked at a PTL at the same time. Without loss of generality, it is assumed that a load transfer, if any, between an LTC lorry and its trailer takes place only directly before a decoupling operation, not when re-coupling. The duration (service time) of a decoupling operation includes time for a potential load transfer.

The problem is static and deterministic, i.e., all data are known in advance.

The objective of the PDPTWT is to find a feasible solution with a minimal (or, at least, low) sum of fixed and travel costs. A feasible solution consists of a set of feasible routes, one for each single lorry and one for each LTC, so that each task is covered by exactly one vehicle (single lorry or LTC). A route is feasible if and only if it starts at the start depot of the vehicle that performs the route, fulfils zero or more tasks, and ends at the vehicle's end depot, while maintaining all time windows, accessibility constraints, and lorry and trailer capacities. In a feasible solution, the following nine cases are possible with regard to accessibility constraints:

		Pickup can be visited with a trailer					
		yes	yes	no	no		
Delivery can be visited with a trailer	yes	1	2			yes	Pickup is visited on main route
	yes	3	4	5	6	no	
	no		7			yes	
	no		8		9	no	
		yes	no	yes	no		
		Delivery is visited on main route					

Figure 1 shows an example LTC route that fulfils the nine tasks $t_1, \ldots, t_9$. For $i = 1, \ldots, 9$, p_i and d_i respectively denote the pickup and the delivery location of task t_i. The route starts and ends at the depot bottom left and performs four subroutes, two each at the parking and transshipment locations ptl_1 and ptl_2. In the figure, task t_i corresponds to case i of the above table for $i = 1, \ldots, 9$. Blue triangles represent locations that can be visited with a trailer; green ones can only be visited without. Triangles pointing upwards represent pickups, those pointing downwards represent deliveries.

There is no lack of practical applications of the PDPTWT. This author has seen use cases in the supply of supermarkets, beverage stores, and apparel stores, in the transport of ready-mixed concrete garages and commercial waste bins, and, most notably, in the less-than-truckload business. As for supermarket and store supply, in many cases loaded pallets, bins, or roll cages picked up at (different) warehouses are delivered to stores, and empty transport equipment is picked up at stores and delivered to warehouses. The transport of ready-mixed concrete garages is often performed in two steps. LTCs are loaded at factories and bring the garages to appropriate parking locations. Later on, other LTCs pick up the garages, possibly from different parking locations, and install them at their final destinations. The situation is similar for commercial waste bins. Empty bins are picked up at various depots and delivered to factories, construction sites etc., from where full bins are picked up and delivered to waste dumps or recycling stations. In the less-than-truckload business, ISO standard

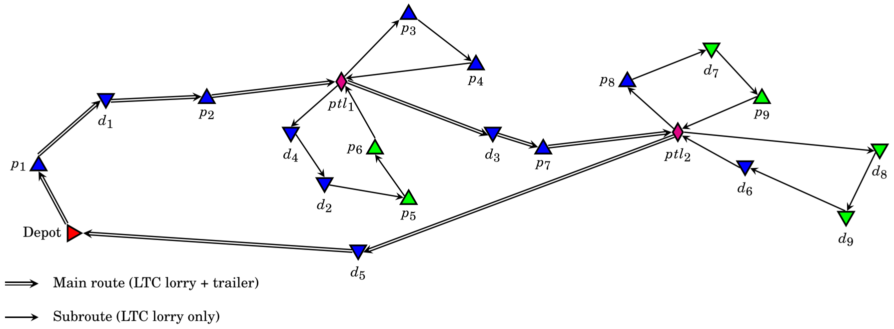

Fig. 1 Example LTC route

containers, swap-body platforms, or smaller collective consignments are picked up at different locations (customer sites or freight forwarding terminals and hubs) and are delivered to other terminals or directly to customers.

The contribution of this paper is threefold: (i) it studies a practically relevant extension of the one-to-one pickup-and-delivery problem with time windows. Put differently, it generalizes vehicle routing problems (VRPs, i.e., problems where either all pickups or all deliveries take place at a central depot) with trailers to pickup-and-delivery problems. (ii) It develops an exact amortized constant-time procedure for testing the feasibility of an insertion of a task into a given PDPTWT route concerning time windows and lorry and trailer capacities. 'Exact' means that the testing procedure will declare the insertion as feasible if and only if the route resulting from the insertion is feasible. 'Amortized constant-time' means that the test itself takes constant time and is independent of the number of tasks (or, equivalently, the number of locations visited) on the route, but that the test uses auxiliary data which must be computed in a preprocessing step which does not run in constant time. (iii) The paper provides a comprehensive set of new benchmark instances and empirically compares the runtime of the constant-time test on these instances with a naïve one that requires linear time by embedding both tests in an adaptive large neighbourhood search algorithm for the heuristic solution of the problem. The results of computational experiments show that the constant-time test outperforms its linear-time counterpart by one order of magnitude on average.

The rest of the paper is structured as follows. The next section gives a brief review of related literature. Section 3 presents the adaptive large neighbourhood search procedure used to solve the PDPTWT. In Sect. 4, the insertion feasibility tests regarding time and capacity are described. Section 5 presents the newly created benchmark instances and discusses the computational results obtained on them. Finally, Sect. 6 gives a conclusion and proposes topics for further research.

2 Related work

This section briefly reviews pertinent literature, focussing on works concerned with pickup-and-delivery problems, routing problems with trailers, and efficient feasibility tests in heuristics for routing problems. *Pickup-and-delivery problems* (without trailers) exist in several variants (one-to-one, one-to-many-to-one, many-to-many, simultaneous delivery and pickup) and have been extensively studied in the last decades. Important surveys are presented by Parragh et al. (2008a, b), Doerner and Salazar-González (2014), and Battarra et al. (2014). These works also provide classification schemes for the different variants. The static, deterministic, multi-vehicle, one-to-one variant with time windows is the most widely studied type. Exact algorithms for this problem are presented by Ropke et al. (2007), Ropke and Cordeau (2009), and Baldacci et al. (2011). According to Battarra et al. (2014), the most successful heuristic procedures, by Bent and Van Hentenryck (2006) and Ropke and Pisinger (2006), are both based on large neighbourhood search.

Routing problems with trailers have also attracted a lot of interest from researchers. The surveys by Prodhon and Prins (2014) and Cuda et al. (2015) contain sections on VRPs with trailers, which are commonly referred to as truck-and-trailer routing problems (TTRPs). Most works on TTRPs consider no time windows. Exact algorithms for TTRPs with time windows (TTRPTWs) are presented by Parragh and Cordeau (2017) and Rothenbächer et al. (2018). Heuristics for TTRPTWs are described by Drexl (2011) (heuristic column generation), Lin et al. (2011) (simulated annealing), Derigs et al. (2013) (hybrid local and large neighbourhood search, attribute-based hill climber), and Parragh and Cordeau (2017) (adaptive large neighbourhood search). *Pickup-and-delivery problems with time windows and trailers* are less well studied. Most papers on this topic consider approaches for problems where vehicles consisting of a tractor and a semi-trailer are employed to perform full-load tasks, i.e., where a vehicle can transport only one task at a time. Examples are the problems examined by Cheung et al. (2008) (attribute-decision model), Xue et al. (2014) (tabu search) and Tilk et al. (2018) (branch-and-price-and-cut). Concerning the PDPTWT version studied here, this author is aware of only one paper: Bürckert et al. (2000) describe a holonic multi-agent system heuristic for a generalization of the PDPTWT in the context of long-distance transport. The authors take into account eight types of resource: driver, lorry with loading capacity, lorry without loading capacity, tractor, trailer, semi-trailer, chassis, and swap-body. Adequate combinations of these resources must be created to fulfil tasks.

Seminal works on *efficient feasibility tests* for insertion or local search procedures for different types of VRPs and PDPs are the ones by Savelsbergh (1985, 1990, 1992), Kindervater and Savelsbergh (1997), Funke et al. (2005), Irnich et al. (2006), Irnich (2008a, b), Masson et al. (2013b), Vidal et al. (2014), and Grangier et al. (2016). None of these, however, considers routing problems with trailers.

3 Adaptive large neighbourhood search for the PDPTWT

Adaptive large neighbourhood search (ALNS) is a very widely and successfully used metaheuristic, in particular for, but not limited to, many different types of routing problem. Pisinger and Ropke (2010) present a tutorial and a literature survey on (A)LNS. The basic idea of large neighbourhood search (LNS), as introduced by Shaw (1997), is to repeatedly perform the following steps. Given an incumbent solution, some of its elements are removed (destruction step) and reinserted (reconstruction step) to create a new solution that replaces the current incumbent if it either improves the best solution found so far or fulfils some other acceptance criterion. ALNS was first used by Ropke and Pisinger (2006) and extends the LNS principle by adding different removal and reinsertion operators and an adaptive operator selection scheme. In the context of pickup-and-delivery or vehicle routing problems, given a complete route plan, a subset of requests or customers is removed from their respective routes in the destruction step, and they are reinserted into the resulting partial routes in the reconstruction step. A pseudocode of the ALNS procedure in general and of our implementation in particular is presented in Fig. 1. The concrete ALNS implementation used for the computational experiments described in this paper follows the set-up described by Ropke and Pisinger (2006) for the PDPTW without trailers. Details on the method are given below.

Algorithm 1 Basic Adaptive Large Neighbourhood Search

Construct an initial feasible solution x and save x as the current best solution x^*, i.e., set $x^* := x$
repeat
  Select a destruction and a reconstruction operator by roulette-wheel selection based on the current operator weights
  Create a neighbouring solution x' from x using the procedures corresponding to the selected destruction and reconstruction operators `// cf. Algorithm 2`
  Update the operator scores `// cf. Subsection 3.3`
  if Solution x' can be accepted, i.e., if a simulated annealing acceptance criterion is fulfilled `// cf. Subsection 3.4`
    Set $x := x'$
  if x is better, i.e., has a lower objective function value, than x^*
    Set $x^* := x$
until a termination criterion is reached (maximal number of iterations)
return x^*

3.1 Destruction procedures

The destruction/removal operators described by Ropke and Pisinger (2006) (random, worst and Shaw removal) are applied. In addition, three further removal strategies are built into the ALNS. In the *arc frequency history removal* heuristic, proposed by Masson et al. (2013a), the aim is to remove tasks that seem to be at bad positions compared to the best known solutions. The heuristic keeps track of how often each arc (connection between two locations) appears in any one of the solutions contained in a fixed-size set composed of the best solutions found so far. In each ALNS iteration, if a solution enters or leaves this set, the frequencies of the arcs in this solution are incremented or decremented accordingly. When the arc frequency history removal heuristic is selected, a frequency value is computed for each task by summing up the frequencies of the arcs over which the pickup and the delivery locations of the task are reached and left in the current solution. Then, the tasks with the lowest frequency values are removed. The *zero-split removal* heuristic, proposed by Parragh et al. (2010), removes sequences of task locations where the vehicle is empty when reaching the first location and when leaving the last. Longer sequences are preferred, and the removed tasks are reinserted one by one. Finally, the *subroute removal* heuristic, as its name implies, removes entire subroutes, which are selected at random. 'Removing a subroute' means that all tasks with at least one location on the subroute are removed. The removed tasks are reinserted one by one in this heuristic, too.

The worst and Shaw removal heuristics exist in a static and a dynamic version. In the static versions, the removal criteria are computed anew only once in an ALNS iteration, in the dynamic versions, they are updated after each removal of a single task. The removal criterion for a task in the worst removal heuristic is the difference in the costs of the current solution with and without the task. The Shaw removal operator uses, for each pair of tasks, a relatedness measure that takes into account the distances between the pickup locations, the distances between the delivery locations, the overlap of the time windows of the pickup locations, the overlap of the time windows of the delivery locations, and the difference between the capacity requirements of the two tasks.

The number m of tasks to be removed in each iteration is selected randomly in the interval $[\min(30, 0.1 \cdot n), \min(60, 0.4 \cdot n)]$, where n is the number of tasks in the instance. All removal operators are randomized in a manner similar to the one proposed by Ropke and Pisinger (2006). Given a list of tasks that contains l elements and is sorted according to one of the criteria of the operators, if m tasks are to be removed, then not necessarily the first m tasks in the list are removed. Instead, the task at position $l \cdot y^p$ is removed. In this formula, y is a uniform random number from the interval [0, 1) and p is the *randomization degree*, which differs between operators as specified in Table 3 in the "Appendix". This is repeated until m tasks have been removed.

3.2 (Re)Construction procedures

The (re)construction procedures are iterative (re)insertion operators that, in each iteration, insert one task into a given empty or incomplete route plan. The heuristics used for this purpose are the parallel greedy and the regret-2, -3, -4 and -M (re)insertion operators. In each iteration, the parallel greedy heuristic inserts the pickup and the delivery locations of a task t at certain positions into a route r if this insertion causes a smallest increase in the total cost of the current route plan. Regret heuristics insert a task t into a route if t has a maximal regret value over all tasks not currently performed. The regret-p value of a task t is the difference in the costs between a cheapest insertion of t and a p-cheapest one. The initial feasible solution is computed with the greedy heuristic.

In lieu of the noise mechanism used by Ropke and Pisinger (2006), *insertion preference strategies* are used. This means that, in each iteration of a reinsertion heuristic, one of the following five strategies is randomly selected with equal probability and applied before deciding which task to insert into which route: (i) make the insertion of tasks where the pickup location can be visited with a trailer more attractive; (ii) similar for tasks where this is not the case; (iii) make the insertion into single lorry routes more attractive; (iv) similar for LTC routes; (v) make it more attractive to insert tasks where the pickup location can be visited with a trailer into LTC routes. This is achieved by appropriately modifying the insertion costs of tasks into particular routes.

All types of insertion heuristic for routing problems, i.e., all procedures for inserting one or more locations into an existing route, test the feasibility of inserting the location(s) at the respective position(s). In other words, they test whether the resource windows of all relevant resources, such as time windows, vehicle capacities, and accessibility constraints for the PDPTWT studied here, are maintained at *each* position in the enlarged route. These tests can be performed in a naïve manner by passing through the enlarged route once, updating all resource consumptions along the way. However, in particular for instances where longer routes containing many locations are possible, this approach is very time-consuming. Constant-time feasibility tests, such as those presented in this paper, are a much more efficient approach. The efficiency gains obtained by constant-time procedures during the actual feasibility tests must, of course, be charged up against the preprocessing efforts needed to update a set of auxiliary data structures which store the information that enables a constant-time test. This update, though, need be performed only after an actual insert of a task into a route has been performed. This means that in each destruction–reconstruction sequence (line 4 in Algorithm 1), when n tasks are currently not on a route after the destruction step, the update is performed n times, and each time, the update is performed for only one route, namely, the one into which the last insert was performed. In each reconstruction step, each currently unperformed task is first tested for insertability into each existing route as well as into a new route to be performed by a vehicle of each vehicle class of which there is still an unused vehicle available. These insertability tests mean testing, for each position on a route, whether the pickup location of a task can be inserted directly behind this position and whether the delivery location of the task can be inserted directly behind the pickup location or at any subsequent position. Moreover, after each actual insert into a certain route, all remaining unperformed tasks must again be tested in this manner for insertability into the changed route. Compared to the computational costs these operations require, the time for the update of the auxiliary data structures is negligible. The experimental results described in Sect. 5 clearly confirm this.

When, in an insertion step, the creation of a new subroute must be tested, which is necessary if a location not reachable by trailer is to be inserted directly after a location that is left with the trailer coupled, a suitable PTL must be selected. The ALNS does not necessarily choose the PTL closest to the task location in question. Instead, a certain degree of randomness is introduced, with closer PTLs being selected with higher probability. This mechanism is similar to what is done in the removal heuristics, as explained in the preceding subsection. Details on how such an insertion is performed are given in Sect. 4.

Algorithm 2 describes the reconstruction process more formally. It presents in detail what happens in line 4 of Algorithm 1.

Algorithm 2 Destruction-Reconstruction Loop

Select the number n of tasks to be removed as described in Section 3.1
Remove n tasks from their routes using the selected destruction procedure; these tasks are now *unplanned*
`// Reinsert the unplanned tasks one by one using the selected reconstruction procedure:`
for each remaining unplanned task t
  Determine the best (according to the selected reconstruction procedure) insertion positions for the pickup and the delivery location of t into each route r by trying each potential insertion position on r using either the linear- or the constant-time test
while unplanned tasks remain
  Select the task t to be inserted and the route r into which t is to be inserted according to the criterion defined by the selected reconstruction procedure and the selected insertion preference strategy
  Insert t into r
  if the constant-time procedure is used
    Update the auxiliary data structures for the route r into which the selected task t was inserted
  Update, for each remaining unplanned task, the best insertion position into the changed route r

3.3 Adaptive weight adjustment

A roulette wheel procedure with adaptive weight adjustment, similar to the one described in Ropke and Pisinger (2006), is used for selecting the destruction and reconstruction operators in each iteration. This works as follows: during segments of 100 iterations, performance scores are recorded for each operator. The scores are initialized to zero and increased by 33 if an application of the operator yields a new best solution, by 9 if the operator yields a solution x' that is better than the current solution x, and otherwise by 13 if the solution is accepted. The operator weights in a new 100-iteration segment are computed as the sum of the weights used in the preceding segment, multiplied by a factor of 0.9, and the relative scores collected in the preceding segment, multiplied by a factor of 0.1. The relative score of an operator in a segment equals the absolute score obtained in this segment divided by the number of times the operator was used in this segment. The destruction and reconstruction operators to use in an iteration are then selected with a probability corresponding to their weights.

3.4 Acceptance mechanism

A simulated annealing acceptance criterion is used. If a new solution x' is better than the one it was created from, it is accepted. Otherwise, if it has not already been generated, it is accepted with a probability of $e^{(-1)\cdot(f(x')-f(x))/t}$, where $f(s)$ is the objective function value of a solution s and t is the *temperature*. The initial value for t is set such that a solution that is five percent worse than the current solution is accepted with probability 0.5. In the course of the algorithm, t is decreased in each iteration by a factor of 0.99975. The information about already generated solutions is stored in compact form in a hash table.

Apart from the above elaborations, the decisive modification to the ALNS as described by Ropke and Pisinger (2006) is that the time window and capacity feasibility tests described in the next section are used; these take into account trailers and accessibility restrictions.

4 Feasibility tests

In the following, techniques are proposed to test the temporal and capacitive feasibility of task insertions into routes performed by single lorries or LTCs in constant time, given appropriate auxiliary data computed in a preprocessing step. (In a slight abuse of terminology, 'amortized constant time' is abbreviated by 'constant time' here and in what follows.) As will be shown, the preprocessing to determine or update the necessary auxiliary data for a route to test time window as well as capacity feasibility takes time quadratic in the number of tasks fulfilled or locations visited on the route, but it is performed only once for a given solution. The resulting data are then used for all feasibility tests, i.e., for testing all potential insertion positions of all unplanned tasks. The routines are embedded in the ALNS metaheuristic described in the previous section. They could, however, also be used within other metaheuristic or local search approaches. In this section, the following notation is used. Each task t from pickup location p to delivery location d is denoted by $t = (p, d)$ and has a capacity requirement $q^t > 0$, which means that q^t units of load must be picked up at p and $-q^t$ units must be delivered at d. The capacity requirement at each location u is denoted by q_u. Hence, $q_p > 0$ for each pickup location p, $q_d < 0$ for each delivery location d, and $q_u = 0$ for each vehicle depot or PTL u. Each location u has a single, hard time window $[a_u, b_u]$, $0 \leq a_u \leq b_u \leq T$, where T is the length of the planning horizon. The depot locations have a time window of $[0, T]$. Each task location u has a unique service time (duration) s_u, and each PTL u has a decoupling duration (including a fixed time for a potential load transfer) of s_u^{dec} and a coupling duration of s_u^{coup}. For each pair (u, v) of locations, t_{uv} denotes the travel time from u to v. Each single lorry, each LTC lorry, and each trailer has a specified one-dimensional capacity, denoted by Q_k^l and Q_k^t respectively. For a single lorry k, $Q_k^t = 0$. The symbol '$==$' serves as equality operator, '$=$' is the assignment operator, and '$x \mathrel{+}= y$' is used as shortcut for '$x = x + y$'.

The descriptions assume that feasibility of an insertion of a task $t = (p, d)$ into an existing route $r = (0, 1, \ldots, n-1, n)$, with p to be inserted directly after position (zero-based index of the route) h and d to be inserted directly after position i, is to be tested. If p cannot be reached with a trailer, r is performed by an LTC, and the trailer is attached upon leaving h, a location triple $\tilde{p} = ptl_p \to p \to ptl_p$ corresponding to a new subroute is inserted after h; similar for i and d. ptl_p is a suitable trailer parking location; similar for d. Note that p, d, ptl_p, ptl_d, $\tilde{p}$, and $\tilde{d}$ are locations, whereas h and i are indices on a route. To simplify notation, when referring to a location visited at a certain position on a route, only the index is used: for example, the start of the time window of index i, i.e., of the ith location visited on a route, is denoted by a_i, and the travel time between index i and a to-be-inserted location v is denoted by t_{iv} etc.

Indices h and i indicate positions in the route *before* p and d are inserted. Hence, if $h == i$, then d is to be inserted directly after p, or, if a triple $\tilde{p} = ptl_p \to p \to ptl_p$ is to be inserted, directly after the triple. If, however, d cannot be reached with a trailer and p is left with the trailer attached or a triple $\tilde{p}$ is to be inserted, then a triple $\tilde{d} = ptl_d \to d \to ptl_d$ is inserted. In principle, if $h == i$ and p or d must be surrounded by a decouple–couple pair, it would also be possible to surround both p and d by one pair. This might be useful for instances where many pickups are close to their deliveries. For simplicity of exposition, this additional possibility is not considered in the present paper. When this option is used, constant-time feasibility tests are just as well possible with the auxiliary data structures described in the following subsections; the formal description, though, is tedious. Moreover, in the course of an ALNS, configurations where it is beneficial that the pickup and the delivery of a task are surrounded by a decouple–couple pair will often be achieved automatically as a result of the removal steps.

Several consecutive subroutes by one LTC lorry at the same PTL are modelled by inserting a decouple–couple pair for each subroute. It is assumed that the fixed service times at PTLs are incurred also in such cases.

4.1 Time windows

In this paper, neither route duration constraints nor time-dependent costs are considered and thus there is no need to strive for minimization of route duration. Under these conditions, it is optimal regarding feasibility to consider only as-early–as-possible schedules, i.e., to assume that a vehicle always leaves a location at the earliest possible point in time; this provides the maximum possible flexibility at subsequent positions on the vehicle's route.

Testing time-window feasibility of an insertion in linear time is trivial: the locations of the to-be-inserted task are tentatively inserted (including PTLs for decoupling and coupling, if necessary); the route is traversed, starting at the depot at time zero; travel, service and waiting times are added; finally, the resulting earliest possible starts of service are compared with the location time windows.

Testing time-window feasibility in constant time is a little more involved. To do so, Savelsbergh (1992) introduced the concept of forward time slack (FTS). The FTS at a position on a route indicates by how much the earliest possible start of service at this position can be postponed without violating a time window at this or a subsequent position on the route. This idea is adapted to test the feasibility of the insertion of a pickup-and-delivery task (p, d) into a PDPTWT route $r = (0, \ldots, n)$ as follows.

First, note that a triple $\tilde{u} = ptl_u \to u \to ptl_u$ can be regarded as a *meta-location* or *segment* (cf. Irnich 2008a; Vidal et al. 2014) and handled as if it were a single location. Hence, whenever the insertion of a triple $\tilde{u}$ needs to be *tested* because the task location u cannot be reached with a trailer, the time window of the corresponding meta-location is tested. (However, when an insertion of a triple for a location u is to be actually *performed*, the sequence $ptl_u \to u \to ptl_u$ must be inserted, because the new subroute created by inserting the triple might be enlarged by an insertion of a task location in a later iteration.) The time window $[a_{\tilde{u}}, b_{\tilde{u}}]$ of a meta-location need be precomputed only once, before the start of the ALNS, for each task location u and each PTL ptl. This can be done by setting

$$\begin{aligned} a_{\tilde{u}} &= \max\left(a_{ptl}, a_u - t_{ptl,u} - s_{ptl}^{dec}\right), \\ b_{\tilde{u}} &= \min\left(b_u - t_{ptl,u} - s_{ptl}^{dec}, b_{ptl} - t_{u,ptl} - s_u - t_{ptl,u} - s_{ptl}^{dec}\right). \end{aligned}$$

If $a_{\tilde{u}} > b_{\tilde{u}}$, then ptl cannot serve as parking location for visiting u. Otherwise, the service time $s_{\tilde{u}}$ of a meta-location $\tilde{u}$ is set to

$$s_{\tilde{u}} = s_{ptl_u}^{dec} + t_{ptl_u,u} + s_u + t_{u,ptl_u} + s_{ptl_u}^{coup}.$$

The travel times to and from a meta-location $\tilde{u}$ are those to and from ptl_u. The travel costs to $\tilde{u}$ are those to ptl_u for a lorry with its trailer plus those from ptl_u to u plus those from u to ptl_u, both for a lorry without its trailer. The travel costs from $\tilde{u}$ are those from ptl_u for a lorry with its trailer. Second, the following auxiliary data are used:

e_i Earliest point in time at which service at index i can begin.

w_i Waiting time at index i, i.e., time period between arrival and beginning of service at i.

w_{ij} Cumulated waiting time between i and j, i.e., sum of waiting times at indices $i, \ldots, j$.

sl_i Slack time from 0 to i, i.e., maximal amount of time by which the departure at the start depot can be postponed from e_0 without violating any time window between 0 and i.

f_i Forward time slack from i to n, i.e., maximal amount of time by which e_i can be postponed without violating any time window from i up to the end of the route.

The first four quantities are computed for each route in a preprocessing step as follows:

$$
\begin{array}{lll}
e_0 = a_0; & e_i = \max(a_i, e_{i-1} + s_{i-1} + t_{i-1,i}); & i = 1, \ldots, n \\
w_0 = 0; & w_i = \max(0, a_i - (e_{i-1} + s_{i-1} + t_{i-1,i})); & i = 1, \ldots, n \\
w_{00} = 0; & w_{0i} = w_{0,i-1} + w_i; & i = 1, \ldots, n \\
sl_0 = b_0 - e_0; & sl_i = \min(sl_{i-1}, b_i - e_i + w_{0,i}); & i = 1, \ldots, n
\end{array}
$$

The FTS can then be computed as $f_i = \min_{j=i,\ldots,n}(sl_j)$ for $i = 0, \ldots, n$. The computation or update of the first four auxiliary data structures requires linear time in n; the FTS computation time is quadratic in n. Still, as the computational results in Sect. 5 demonstrate, this preprocessing clearly pays off.

Given these data, time-window feasibility of an insertion can be tested as described in Algorithm 3 (cf. Masson et al. 2013b). Note that it is sufficient to execute lines 1–7 of Algorithm 3 only once for each h with a given PTL ptl_p. If TestTimeWindows returns false in line 7, it makes no sense to test further insertion positions for d with h as insertion position for p or $\tilde{p}$, because neither p nor $\tilde{p}$ can be inserted after h or later on r; hence, the next position for inserting p can be considered.

Due to the limited planning horizon, if a task location not reachable by trailer is to be inserted at a certain position on a main route, i.e., when a new subroute must be created, in principle all PTLs must be tested for whether an insertion at this position is possible. This, of course, increases the runtime of an insertion heuristic. However, if only a subset of all PTLs is considered, an insertion heuristic may miss some feasible solutions, and the solution quality of the overall algorithm may deteriorate. The time window feasibility test described in Algorithm 3 receives as input a particular choice of PTL for the pickup and for the delivery location. Therefore, the test is exact in the sense that it will correctly consider the insertion of a specific triple $ptl_v \rightarrow v \rightarrow ptl_v$ feasible if and only if the insertion of this specific triple *is* feasible. If several PTLs shall be considered, Algorithm 3 must be embedded in a loop over these PTLs.

Algorithm 3 TestTimeWindows($p, \tilde{p}, d, \tilde{d}, r, h, i$)

Input: Pickup-and-delivery task $t = (p, d)$
Route $r = (0, 1, 2, \ldots, n)$
Indices of insertion positions h, i with $0 \le h \le i \le n-1$
Meta-locations $\tilde{p} = ptl_p \to p \to ptl_p$ and $\tilde{d} = ptl_d \to d \to ptl_d$ for p and d respectively, for a specific PTL ptl_p for p and a specific PTL ptl_d for d (only if r is performed by an LTC)

Result: Returns **true** if and only if inserting p or $\tilde{p}$ into r directly after index h and d or $\tilde{d}$ directly after i (or, if and only if $h == i$, after p or $\tilde{p}$) is feasible regarding all time windows, **false** otherwise

$u = p$
if p cannot be reached with trailer and trailer is currently attached
    $u = \tilde{p}$
`// Test feasibility of inserting` u `after` h
$e_u = \max(a_u, e_h + s_h + t_{h,u})$
if $e_u > b_u$
    return false
`//` Δ_{h+1} `is the time shift at` $h+1$`, the increase of` e_{h+1}`, caused by inserting` u
$\Delta_{h+1} = \max(0, e_u + s_u - e_{h+1} + t_{u,h+1})$
if $\Delta_{h+1} > f_{h+1}$
    return false
$v = d$
if d cannot be reached with trailer and trailer is currently attached
    $v = \tilde{d}$
`// Test feasibility of inserting` v `after` i
if $i > h$ `// Delivery not directly after pickup`
    $e_v = \max(a_v, a_i + \max(0, \Delta_{h+1} - w_{h+1,i}) + s_i + t_{i,v})$
    if $e_v > b_v$
        return false
    `//` Δ_{i+1} `is the time shift at` $i+1$ `caused by inserting` v
    $\Delta_{i+1} = \max(0, e_v + s_v - e_{i+1} + t_{v,i+1})$
    if $\Delta_{i+1} > f_{i+1}$
        return false
else `//` $i == h$`, i.e., delivery directly after pickup`
    $e_v = \max(a_v, e_u + s_u + t_{u,v})$
    if $e_v > b_v$
        return false
    $\Delta_{i+1} = \max(0, e_v + s_v + t_{v,i+1} - e_{i+1})$
    if $\Delta_{i+1} > f_{i+1}$
        return false
    return true

4.2 Capacities

Time-window tests are the same for single lorry as well as LTC routes: at each position on a route, the earliest start of service must lie within the time window of the respective location. By contrast, the presence of trailers requires additional capacity tests for LTC routes compared to single lorry routes. In this section, it is first described verbally what must be tested in linear- and constant-time capacity tests. Afterwards, the linear- and constant-time test routines are presented.

At each position of *single lorry routes and main routes of LTCs*, the *total load balance*, which is the difference between the load picked up on the route so far minus the load delivered so far, must be less than or equal to the lorry plus the trailer capacity.

For capacity considerations on *subroutes*, the following two quantities are relevant:

- The *minimal lorry load at decoupling*, i.e., the minimal load that must inevitably be in the lorry upon leaving the decoupling location. This load is equal to the maximum of the following two values:
  - The difference between the total load balance at the decoupling location and the trailer capacity.
  - The sum of the capacity requirements incurred by the deliveries on the subroute whose pickups lie before the subroute. (This value is nonnegative, so that the minimal lorry load at the decoupling location is nonnegative as well.)

– The *subroute load balance* at each position, i.e., the difference between the sum of the load in the lorry at the start of the subroute plus the load picked up on this subroute so far minus the load delivered on this subroute so far. (The subroute load balance can be positive, zero, or negative.)

A subroute is capacity-feasible if and only if the first quantity is less than or equal to the lorry capacity and the value of the second is nonnegative and less than or equal to the lorry capacity at each position.

4.2.1 Testing capacities in linear time

To test capacity-feasibility of an insertion in linear time, the procedure detailed in Algorithm 4 is used. For simplicity, the vehicle index k is omitted: Q^l and Q^t are used instead of Q_k^l and Q_k^t to denote the lorry and the trailer capacity.

Testing capacity in linear time for single-lorry routes is simple: the to-be-inserted task is tentatively inserted, one pass over the route is performed, and the capacity requirement at each position is added to the total load and compared with the lorry capacity (lines 2–6).

Testing capacity for LTC routes is not entirely straightforward even in linear time. As discussed above, it must be known at the start of a subroute how much load must be in the lorry to be able to perform the deliveries whose pickups are not on this subroute. This information is gathered in one forward pass over the route (lines 10–15). (In reality, it is of course not enough to have this amount of load in the lorry at the start of a subroute. It is also necessary to have the *right* commodities aboard the lorry, those that must be delivered on this subroute. This, however, has to be ensured by the driver. For algorithmic planning, it is sufficient to test whether enough loading capacity is available on the lorry.) The second pass (lines 19–36) then performs the actual capacity test on main routes and subroutes (total load at all positions, minimal load at decoupling positions, subroute load balance at all positions on subroutes).

Algorithm 4 TestCapacityLinear(r,k)

Input: Route $r = (0,1,2,\dots,n)$ with to-be-inserted task tentatively inserted, including decoupling and coupling locations where necessary, and capacity requirements $q_v \in \mathbb{Z}$, $v = 0,1,2,\dots,n$
Vehicle k (single lorry or LTC) with lorry and trailer capacities Q^l and Q^t; for single lorries, $Q^t == 0$

Result: Returns **true** if and only if lorry and, where applicable, trailer capacity of k are maintained at each index of r, **false** otherwise

```
TotalLoad = 0
if Q^t == 0 // Test for single lorries
    for v = 0,1,2,...,n
        TotalLoad += q_v
        if TotalLoad > Q^l
            return false
else // Test for LTCs
    LoadDeliveredButNotPickedUpOnSubroute = array of integers of length n+1, initialized to 0
    IndexOfLastDecouple = 0
    for v = 0,1,2,...,n
        if v corresponds to a decoupling location
            IndexOfLastDecouple = v
        if Trailer is not attached upon leaving v
            if v corresponds to a delivery and associated pickup is before current subroute
                LoadDeliveredButNotPickedUpOnSubroute[IndexOfLastDecouple] += (−1)·q_v
    MinLorryLoadSinceLastDecouple = 0
    MaxLorryLoad = 0
    MaxLorryLoadSinceLastDecouple = 0
    for v = 0,1,2,...,n
        TotalLoad += q_v
        if TotalLoad > Q^l + Q^t
            return false
        MaxLorryLoad = min(TotalLoad, Q^l)
        if v corresponds to a decoupling location
            MinLorryLoadSinceLastDecouple = max(TotalLoad − Q^t, LoadDeliveredButNotPickedUpOnSubroute[v])
            if MinLorryLoadSinceLastDecouple > Q^l
                return false
            MinLorryLoadSinceLastDecouple = max(MinLorryLoadSinceLastDecouple, 0)
            MaxLorryLoadSinceLastDecouple = MaxLorryLoad
        if Trailer is not attached upon leaving v
            MinLorryLoadSinceLastDecouple = max(MinLorryLoadSinceLastDecouple + q_v, 0)
            if MinLorryLoadSinceLastDecouple > Q^l
                return false
            MaxLorryLoadSinceLastDecouple = min(MaxLorryLoadSinceLastDecouple + q_v, Q^l)
            if q_v < 0 and MaxLorryLoadSinceLastDecouple < 0
                return false
return true
```

4.2.2 Testing capacities in constant time

To test the feasibility of the insertion of a pickup-and-delivery task (p, d) in constant time, the following data, computed for each route in a preprocessing step, can be used.

1. TrailerAttached: An array of boolean values. TrailerAttached[i] indicates whether or not the trailer is attached upon leaving (the location corresponding to) index i.
2. MaxTotalLoadOfSegment: A two-dimensional array of nonnegative integers. MaxTotalLoadOfSegment[i][*offset*] stores, for an index i on a route, the maximal load balance from the start of the route at any index from i up to and including i + *offset*. In particular, MaxTotalLoadOfSegment[i][0] stores the overall load picked up but not delivered yet from the start depot to and including the location at index i.
   For example, consider the following route:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Capacity requirement	0	+40	+10	0	+10	+20	−40	+5	−10	0	−10	−20	−5	0

This route contains one subroute, which starts at index 3 and ends at index 9, i.e., the zero value at index 3 corresponds to a decoupling process at some PTL, and the zero value at index 9 represents the associated coupling process at this PTL. The load balances at indices 2–6 are $+50$, $+50$, $+60$, $+80$, and $+40$; thus, MaxTotalLoadOfSegment[2][4] $= +80$. Moreover, MaxTotalLoadOfSegment[8][0] $= +35$.

3. TotalLoadDeliveredButNotPickedUpOnSubroute: An array of nonnegative integers. If i is an index corresponding to a decoupling location, TotalLoadDeliveredButNotPickedUpOnSubroute[i] stores the overall load delivered but not picked up on the respective subroute.
4. LoadBalanceFromStartOfSubroute: An array of integers. LoadBalanceFromStartOfSubroute[i] stores, for an index i on a subroute, the positive, negative or zero load balance from the start of the subroute up to and including i.
   In the above example route, LoadBalanceFromStartOfSubroute[7] $= -5 = 10 + 20 - 40 + 5$.
5. MaxLoadBalanceFromStartOfSubroute: A two-dimensional array of nonnegative integers. MaxLoadBalanceFromStartOfSubroute[i][$offset$] stores, for an index i on a subroute, the maximum of zero and the largest load balance from the start of the subroute to any index from i up to and including $i + offset$.
   In the above example, MaxLoadBalanceFromStartOfSubroute[6][2] $= 0 = \max(0, -10, -5, -15) = \max(0,$ LoadBalanceFromStartOfSubroute[7]$)$.
6. IndexOfLastPrecedingDecouple: An array of nonnegative integers. IndexOfLastPrecedingDecouple[i] stores the index where the last decoupling that precedes i on the route occurs.
7. OffsetOfNextCoupling: An array of nonnegative integers. OffsetOfNextCoupling[i] stores the number of positions on the route from i until the next index of a coupling process.

MaxTotalLoadOfSegment and MaxLoadBalanceFromStartOfSubroute can be filled using a nested forward pass, i.e., by iterating over all indices $j \geq i$ for each index i on the route. All other data structures described above can be filled or updated by passing through a route once. This means that all necessary preprocessing data for a route can be computed in quadratic time in the number of tasks on the route.

Given these data, the capacity feasibility of an insertion of a task $t = (p, d)$ into an existing route r, with p to be inserted directly after position (zero-based index of the route) h and d to be inserted directly after position i, can be tested as described in Algorithm 5. It is evident that the algorithm itself runs in constant time, i.e., its runtime is independent of the number of tasks or the number of locations visited on route r.

Algorithm 5 TestCapacityConstant(p,d,r,h,i,k)

Input: Pickup-and-delivery task $t=(p,d)$ with capacity requirement $q>0$
Route $r=(0,1,2,\dots,n)$
Indices of insertion positions h,i with $0\le h\le i\le n-1$
Vehicle k (single lorry or LTC) with lorry and trailer capacities Q^l and Q^t; for single lorries, $Q^t==0$

Result: Returns **true** if and only if inserting p or a triple $\tilde{p}=ptl_p\to p\to ptl_p$ into r directly after index h and d or a triple $\tilde{d}=ptl_d\to d\to ptl_d$ directly after i (or, if and only if $h==i$, after p or $\tilde{p}$) is feasible regarding lorry and trailer capacity, **false** otherwise

```
1  // Evaluate feasibility of insertion regarding total capacity
2  if Q^l + Q^t < MaxTotalLoadOfSegment[h][i − h] + q
3  |  return false
4  else if Q^t == 0
5  |  return true
6  // Evaluate feasibility of insertion regarding lorry capacity
7  if i > h // Delivery not directly after pickup
8  |  // Evaluate feasibility of insertion of pickup
9  |  if TrailerAttached[h] == false // Trailer not attached when leaving h
10 |  |  ind = IndexOfLastPrecedingDecouple[h]
11 |  |  MinLoadAtDecouple = max(MaxTotalLoadOfSegment[ind][0] − Q^t,
12 |  |                        TotalLoadDeliveredButNotPickedUpOnSubroute[ind])
13 |  |  LoadAfterPickup = MinLoadAtDecouple + LoadBalanceFromStartOfSubroute[h] + q
14 |  |  offset = OffsetOfNextCoupling[h + 1]
15 |  |  if h + OffsetOfNextCoupling[h] ≥ i
16 |  |  |  offset = i − h
17 |  |  if LoadAfterPickup + MaxLoadBalanceFromStartOfSubroute[h + 1][max(0,offset − 1)] > Q^l
18 |  |  |  return false
19 |  // Evaluate feasibility of insertion of delivery
20 |  if TrailerAttached[i] == false
21 |  |  if i − h ≥ OffsetOfNextCoupling[h] // Delivery not on same subroute as pickup
22 |  |  |  ind = IndexOfLastPrecedingDecouple[i]
23 |  |  |  MinLoadAtDecouple = max(MaxTotalLoadOfSegment[ind][0] − Q^t,
24 |  |  |                        TotalLoadDeliveredButNotPickedUpOnSubroute[ind])
25 |  |  |  if MinLoadAtDecouple + MaxLoadBalanceFromStartOfSubroute[ind][i − ind] + q > Q^l
26 |  |  |  |  return false
27 else // i == h, i.e., delivery directly after pickup
28 |  if TrailerAttached[h] == false
29 |  |  ind = IndexOfLastPrecedingDecouple[h]
30 |  |  MinLoadAtDecouple = max(MaxTotalLoadOfSegment[ind][0] − Q^t,
31 |  |                        TotalLoadDeliveredButNotPickedUpOnSubroute[ind])
32 |  |  if MinLoadAtDecouple + LoadBalanceFromStartOfSubroute[h] + q > Q^l
33 |  |  |  return false
34 return true
```

Note that, to test the capacity constraints, it is irrelevant whether or not the pickup and/or the delivery location of the task to be inserted must be surrounded by a decouple–couple pair for insertion at the position in question, as decoupling and coupling processes have a capacity requirement of zero.

Note further that, similar to the situation in Algorithm 3, if TestCapacityConstant returns false from line 3 or line 18, it is unnecessary to consider further potential insertion positions for d for the current insertion position of p. Instead, the next position for inserting p can be considered. Hence, it is sufficient here to execute lines 2–20 of Algorithm 5 only once for each h.

5 Computational experiments

5.1 Benchmark instances

To this author's knowledge, there are no benchmark instances for the PDPTWT as studied in this paper. Therefore, a set of instances has been created to perform computational experiments with solution procedures. A well-known and widely used set of benchmark instances for pickup-and-delivery problems with time windows and without trailers has been proposed by Li and Lim (2003) and is available at www.sintef.no/

projectweb/top/pdptw/li-lim-benchmark. This set comprises six classes of instances, with 100, 200, 400, 600, 800, and 1000 task locations, and thus with 50, 100, 200, 300, 400, and 500 tasks respectively. The instances have been derived from the Solomon instances for the vehicle routing problem with time windows (Solomon 1987), and in analogy to the original data, the Li and Lim instances are also partitioned into six classes LC1, LC2, LR1, LR2, LRC1, and LRC2 according to structural characteristics as follows: 'C' stands for geographically clustered tasks which, for the PDPTW and the PDPTWT, also means that the pickup and the delivery location of a task are close together; 'R' stands for geographically randomly distributed tasks; '1' stands for restrictive time windows so that only few tasks per route are possible; and '2' stands for less restrictive time windows and a longer planning horizon, which makes longer routes (routes covering more tasks) possible. Each instance has a homogeneous fleet, and start and end depot location of the vehicles coincide.

As pointed out by Derigs et al. (2013, p. 544), some benchmark instances for vehicle routing problems with trailers are constructed such that there is no need to use lorry–trailer combinations at all, because the capacity of the lorry is high enough for transporting the entire demand and/or the time windows are so restrictive that a vehicle cannot serve many customers. This has also been observed when trying to modify the Li and Lim instances for use with trailers. Therefore, the benchmark instances for the PDPTWT have two vehicle classes: lorry–trailer combinations and single lorries. The single lorries have artificially high fixed cost, so that they are used only when necessary to ensure that all tasks are covered. Such cases can occur when the time windows of a task are so tight that there is not enough time to decouple the trailer to visit the pickup or the delivery task.

With this in mind, the Li and Lim instances have been adapted to the PDPTWT as follows:

- Every even-numbered location (as listed in the original Li and Lim instance file) is reachable by trailer, i.e., locations 0, 2, 4, 6…; the odd-numbered ones are not.
- Starting with location 0 (the depot), every second location that is reachable by trailer may be used for parking and transshipment; i.e., for locations 0, 4, 8, 12,…, a PTL is created. This means that the number of PTLs is approximately half the number of tasks.
- As mentioned, the time windows of task locations are generally too short in the Li and Lim instances, so that no LTCs are used. Therefore, each original time window $[a_u, b_u]$ of a task location u is enlarged to $a_u = \max(0, a_u - \mathit{TWShift})$ and $b_u = \min(b_u + \mathit{TWShift}, T)$, where $\mathit{TWShift} = \lfloor 100 + (\mathit{AvgPickupTime} + \mathit{AvgDeliveryTime})/2 \rfloor$, and *AvgPickupTime* and *AvgDeliveryTime* respectively indicate the arithmetic mean of the service times at pickup and at delivery locations as indicated in the original files, rounded down to the nearest integer.
- The time windows of parking and transshipment locations are set to the complete planning horizon, i.e., to the time window of the depot. According to the author's practical experience, this is a mild and realistic assumption.
- The decoupling and coupling service times at PTLs are set to *AvgPickupTime* and *AvgDeliveryTime* respectively.
- The number of single lorries as well as the number of LTCs is considered unlimited.

- Single lorries are assigned a fixed cost of 1000; LTCs have no fixed cost.
- As in the Li and Lim instances, Euclidean distances are used for travel times as well as travel costs. For LTCs, travel times and costs are the same whether or not the trailer is currently attached.
- Capacities of single and LTC lorries are set to the vehicle capacity specified in the respective original instance; trailer capacities are set to 150% of the lorry capacity.

There is an arc between two locations u and v, i.e., a location v can be visited directly after a location u, unless $a_u + t_u^s + t_{uv} > b_v$, where $t_0^s = 0$ for the depot location 0, $t_u^s = s_u$ for all task locations u, and $t_u^s = \min(AvgPickupTime, AvgDeliveryTime)$ for all PTLs u.

Table 1 shows the distribution of the number of instances of the different types and basic instance characteristics. Note that the number of tasks differs slightly between instances of the same size class in the Li and Lim instances. Therefore, the values in the columns from 'No. locations' to 'No. Arcs' are averages, too. By construction of the instances, the column 'No. Tasks' also indicates the number of task locations reachable by trailer. Lorry capacities are the same for all instances of the same size class for each type.

5.2 Results

The code was programmed in C++ and compiled with Microsoft Visual Studio Enterprise 2017, Version 15.5.3. The experiments were run on a workstation with the Windows 10 Education operating system, an Intel Xeon E5-1660 v3 @ 3.00 GHz CPU, and 64 GB RAM in single-thread mode. The parameters used in the ALNS are listed in Table 3 in the "Appendix".

To assess the relative performance of the linear- and the constant-time test, 10,000 ALNS iterations were performed with both tests for all instances of size classes 100, 200, and 400, i.e., those with at most 200 tasks. For the larger instances, computation times using the linear-time test became too long, so that, for size classes 600, 800, and 1000, only the constant-time test was used. For the linear-time test, the time windows are tested together with the capacities in the loop of line 3 or 19 in Algorithm 4. The constant-time test first examines time window feasibility, then capacities. Aggregated results are shown in Table 2; detailed results by instance are given in Tables 4, 5, 6, 7, 8 and 9 in the "Appendix".

The most important finding that can be read from Table 2 is that the speedup of the constant-time test compared to the linear one is considerable for all instance types and ranges from a factor of nine to a factor of 142, with an average of 38. This demonstrates that the effort of implementing the constant-time tests is well justified.

Further insights that can be obtained from the data in Table 2 are:

- The larger the instance, the higher is the iteration number where the best solution was found.
- The number of routes in the best solution found can differ significantly between instances of the same size class and type.

Table 1 Instance characteristics

Size class	Type	No. instances	Average				Length planning horizon	Average				Lorry capacity
			No. tasks	No. locations	No. PTLs	No. arcs		Length time window	Pickup service time	Delivery service time	Capacity requirement	
100												
	LC1	9	53	135	27	16,309	1236	584	80	90	19	200
	LC2	8	51	130	26	13,964	3390	1131	86	90	18	700
	LR1	12	53	134	27	17,633	230	209	9	10	15	200
	LR2	11	51	129	26	15,492	1000	562	10	10	15	1000
	LRC1	8	53	136	27	17,961	240	220	9	10	17	200
	LRC2	8	51	130	26	15,528	960	501	10	10	17	1000
	All	56	52	132	27	16,222	1100	513	32	34	16	543
200												
	LC1	10	105	266	53	63,713	1351	628	81	90	18	200
	LC2	10	101	256	51	54,852	3598	1191	87	90	19	700
	LR1	10	105	264	53	63,138	634	341	9	10	17	200
	LR2	10	101	255	51	55,630	2535	959	10	10	17	1000
	LRC1	10	105	266	53	64,724	634	312	9	10	18	200
	LRC2	10	101	255	51	55,989	2535	755	10	10	18	1000
	All	60	103	260	52	59,674	1881	698	34	37	18	550

Table 1 continued

Size class	Type	No. instances	Average				Length planning horizon	Average				Lorry capacity
			No. tasks	No. locations	No. PTLs	No. arcs		Length time window	Pickup service time	Delivery service time	Capacity requirement	
400												
	LC1	10	210	527	106	248,962	1501	652	82	90	18	200
	LC2	10	203	510	102	217,304	3693	1177	87	90	19	700
	LR1	10	209	525	105	241,818	804	384	9	10	18	200
	LR2	10	202	507	102	218,047	3213	1135	10	10	18	1000
	LRC1	10	208	523	105	246,120	765	337	9	10	18	200
	LRC2	10	203	509	102	220,785	3060	827	10	10	18	1000
	All	60	206	517	104	232,172	2173	752	34	37	18	550
600												
	LC1	10	315	791	158	551,278	1496	653	81	90	18	200
	LC2	10	305	764	153	483,976	3815	1197	87	90	19	700
	LR1	10	314	788	158	512,648	1206	476	9	10	18	200
	LR2	10	303	759	152	476,558	4823	1527	10	10	18	1000
	LRC1	10	314	788	158	517,217	1206	391	9	10	18	200
	LRC2	10	303	760	152	472,827	4823	1056	10	10	18	1000
	All	60	309	775	155	502,417	2895	883	34	37	18	550

Table 1 continued

Size class	Type	No. instances	Average				Length planning horizon	Average				Lorry capacity
			No. tasks	No. locations	No. PTLs	No. arcs		Length time window	Pickup service time	Delivery service time	Capacity requirement	
800												
	LC1	10	419	1051	210	946,974	1676	676	82	90	18	200
	LC2	10	406	1018	204	855,818	3811	1200	87	90	19	700
	LR1	10	418	1049	210	871,608	1688	573	9	10	18	200
	LR2	10	404	1012	203	836,128	6751	1989	10	10	18	1000
	LRC1	10	418	1048	210	848,080	1573	431	9	10	18	200
	LRC2	10	404	1012	203	815,230	6289	1224	10	10	18	1000
	All	60	412	1032	207	862,306	3631	1016	34	37	18	550
1000												
	LC1	10	525	1314	263	1,420,904	1824	684	82	90	18	200
	LC2	10	508	1272	255	1,325,717	3914	1210	87	90	19	700
	LR1	10	523	1309	262	1,325,967	1925	617	9	10	18	200
	LR2	10	504	1263	253	1,290,588	7697	2195	10	10	18	1000
	LRC1	10	524	1312	263	1,228,887	1821	453	9	10	18	200
	LRC2	8	505	1266	253	1,252,281	7284	1492	10	10	18	1000
	All	58	515	1290	258	1,309,291	3967	1095	35	38	18	534
All												
	All	354	267	670	134	497,857	2617	828	34	36	18	546

Table 2 Aggregated computational results (minimum/average/maximum)

Size class	Type	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test	Ratio runtime linear/constant
100								
	LC1	1850/5505/9249	10/11/11	10/11/11	10/10/11	6/8/10	42/44/47	10/11/12
	LC2	282/3956/8699	4/4/4	4/4/4	3/4/4	2/3/4	71/78/90	35/37/39
	LR1	4794/8111/9850	11/11/11	11/11/11	10/11/12	6/7/9	39/42/45	9/9/10
	LR2	414/5977/9916	2/3/4	2/3/4	2/3/5	1/3/5	75/112/156	33/47/58
	LRC1	4938/8062/9916	11/12/12	11/12/12	11/12/14	6/8/11	39/42/43	9/9/10
	LRC2	2474/5334/9999	3/3/4	3/3/4	3/4/5	2/3/4	74/92/110	32/39/43
	All	282/6276/9999	2/7/12	2/7/12	2/7/14	1/5/11	39/68/156	9/25/58
200								
	LC1	5116/8530/9980	20/21/21	20/21/21	20/21/22	12/14/16	148/157/169	11/12/12
	LC2	2762/5676/9606	6/7/8	6/7/8	6/7/9	4/6/8	240/280/350	36/39/41
	LR1	4497/7663/9645	8/10/11	8/10/11	8/12/16	7/9/11	175/198/229	19/22/26
	LR2	294/7799/9949	3/4/5	3/4/5	5/8/15	3/7/13	375/617/987	69/90/121
	LRC1	6133/8675/9878	9/11/12	9/11/12	10/12/15	7/10/13	175/190/230	18/21/27
	LRC2	212/6869/9759	4/5/6	4/5/6	5/10/18	5/9/17	314/482/657	53/69/82
	All	212/7535/9980	3/9/21	3/9/21	5/12/22	3/9/17	148/321/987	11/42/121

Table 2 continued

Size class	Type	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test	Ratio runtime linear/constant
400								
	LC1	7750/9320/9998	36/39/42	36/39/42	35/40/42	19/25/29	468/494/521	11/11/12
	LC2	5247/8429/9977	12/13/14	12/13/14	13/14/15	9/12/14	735/854/1048	34/36/38
	LR1	6366/8456/9987	15/18/21	15/18/21	17/21/26	13/18/20	538/630/763	19/24/31
	LR2	4193/8404/9961	5/7/9	5/7/9	13/19/25	12/17/22	1198/1943/3432	76/101/142
	LRC1	6771/8677/9989	15/21/24	15/21/24	17/23/27	14/18/21	493/554/681	18/21/29
	LRC2	3927/8600/9968	7/9/12	7/9/12	13/22/27	13/19/23	940/1394/2434	59/78/108
	All	3927/8648/9998	5/18/42	5/18/42	13/23/42	9/18/29	468/978/3432	11/45/142
600								
	LC1	8262/9523/9958	57/61/64	57/61/64	55/61/65	34/40/45	778/820/868	
	LC2	8330/9425/9964	19/21/22	19/21/22	20/24/32	17/22/28	1092/1259/1505	
	LR1	3047/8285/9981	18/25/31	17/24/31	23/34/47	21/30/37	1010/1163/1452	
	LR2	7062/8658/9915	7/10/12	7/10/12	28/42/60	25/35/48	2216/3584/5803	
	LRC1	7960/9290/9991	18/28/32	18/28/32	25/32/36	22/26/31	876/994/1391	
	LRC2	5731/9177/9995	8/12/17	8/12/17	29/44/61	25/37/53	1742/2394/4247	
	All	3047/9060/9995	7/26/64	7/26/64	20/40/65	17/32/53	778/1702/5803	

Table 2 continued

Size class	Type	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test	Ratio runtime linear/constant
800								
	LC1	7969/9154/9965	73/81/87	70/80/87	70/79/84	47/53/58	1081/1141/1202	
	LC2	7271/9407/9942	27/28/30	24/27/30	31/36/48	23/30/39	1484/1754/1923	
	LR1	7632/9187/9998	22/32/39	19/27/36	28/41/57	26/37/48	1406/1628/2083	
	LR2	7592/9470/9942	7/13/17	7/12/16	30/58/81	26/50/71	2816/4581/7905	
	LRC1	8637/9521/9989	26/40/47	24/36/40	35/43/50	32/37/43	1116/1286/1743	
	LRC2	6887/8658/9992	12/16/21	12/15/20	35/68/103	32/58/79	2040/2893/4923	
	All	6887/9233/9998	7/35/87	7/33/87	28/54/103	23/44/79	1081/2214/7905	
1000								
	LC1	5994/8963/9999	92/103/110	88/95/103	88/94/108	62/68/78	1310/1477/1597	
	LC2	7300/9098/9988	34/36/38	32/35/36	40/48/58	33/42/52	1956/2196/2581	
	LR1	8048/9504/9997	27/40/50	22/32/40	37/54/68	34/48/55	1822/2151/2800	
	LR2	8723/9477/9910	10/16/21	10/14/17	43/69/99	42/62/83	3275/5497/1002	
	LRC1	7727/9518/9998	33/51/59	29/41/48	50/57/71	42/48/60	1698/1888/2665	
	LRC2	8358/9499/9934	14/20/24	13/18/21	56/89/118	50/75/97	2937/4035/6626	
	All	5994/9338/9999	10/45/110	10/40/103	37/68/118	33/56/97	1310/2834/1002	
All								
	All	212/8366/9999	2/23/110	2/22/103	2/34/118	1/28/97	39/1359/1002	9/38/142

- As LTC routes have no fixed cost, most routes are actually LTC routes. This also shows that the instances are a suitable test bed for routing problems with trailers (remember the comment on p. 12).
- In particular for the larger instances with long planning horizon and wide time windows, the number of subroutes greatly exceeds the number of LTC routes, meaning that the average LTC route performs more than one subroute. Most PTLs are used only once.
- As was to be expected, the runtimes for the instances with more tasks per route, i.e., fewer routes, are consistently higher than those for the other instances.
- For the LR and LRC instances, the speedup obtained by the constant-time test increases with increasing instance size; for the LC instances, this is not the case.
- The speedup is significantly greater for the instances with more tasks per route (classes with '2').

6 Conclusions and outlook

This paper has studied the PDPTWT, a routing problem which aims at fulfilling a set of transport tasks between pickup and delivery locations, subject to time window constraints and accessibility restrictions, by means of a fleet consisting of single lorries and lorry–trailer combinations. Procedures to test the temporal and capacitive feasibility of inserting a task into an existing route have been presented. Given adequate data computed in a preprocessing step, these procedures run in constant time. They have been embedded in an adaptive large neighbourhood search algorithm for the heuristic solution of the PDPTWT. A comprehensive set of benchmark instances has also been created. The results of computational experiments are presented which show significant speedups that can be realized with the constant-time feasibility test. Topics for further research abound.

As the focus of the research presented here was on efficient feasibility testing, not on solution quality, many options exist regarding *algorithmic refinements* to improve solution quality of the ALNS. First of all, local and/or very large-scale neighbourhood search routines could be added, as done, e.g., by Derigs et al. (2013) and Gschwind and Drexl (2019). Also, matheuristic components, e.g., solving a set-covering problem with all generated routes at the end of the ALNS, cf. Parragh and Schmid (2013), Villegas et al. (2013), could be helpful. Another refinement would be to add a splitting procedure based on dynamic programming that finds optimal PTLs for given routes, cf. Prins (2004) and Villegas et al. (2011). Finally, it could be beneficial to allow infeasible solutions in the course of the algorithm. This is a strategy not commonly applied with ALNS, but it has been applied successfully with other metaheuristics for tightly constrained problems (Wen et al. 2009; Vidal et al. 2013) and might thus be useful for PDPTWT instances with tight time windows.

Regarding *modelling extensions*, many additional practically relevant constraints could be taken into account. Two particularly interesting extensions are loading constraints such as last-in-first-out, and the impossibility of transferring load between an

LTC lorry and its trailer. Of special relevance in connexion with constant-time feasibility tests are limits on route duration and on the time or the number of intermediate stops between the pickup and the delivery of a task. Load-dependent service times require an optimization of the load transfer amounts from lorry to trailer at decoupling and coupling locations, a considerable additional intricacy. Time-dependent costs (and route duration constraints, too) lead to the difficult situation that an as-early–as-possible schedule need no longer be optimal (Savelsbergh 1992), thus violating a fundamental assumption on which the feasibility tests described in the present paper are based. Also *other variants of pickup-and-delivery problems*, such as one-to-many-to-one problems [also called vehicle routing problems with backhauls, Irnich et al. (2014)], many-to-many, and simultaneous PDPs (Battarra et al. 2014), lend themselves to consider a fleet containing trailers.

Furthermore, in many pickup-and-delivery applications, the possibility or even the requirement to split tasks exists [cf. the survey by Drexl (2012) and the more recent papers by Masson et al. (2013b) and Grangier et al. (2016)]. This means that a task $t = (p, d)$ can be decomposed into two subtasks or legs, (p, tl) and (tl, d) at transshipment locations tl. The legs of split tasks can be performed by different vehicles, and this creates an interdependence between routes: changes in one route may make one or several or all other routes infeasible. This interdependence requires a synchronization regarding time and load and, when trailers are considered, leads to the *PDPTWT with synchronization*.

Of course, all of the above extensions and variants can also be considered in a *dynamic and/or stochastic context*, where some information becomes known only after execution of a route plan has begun and/or some data are known only in the form of random variables, cf. Berbeglia et al. (2010) and Flatberg et al. (2005). Finally, there is yet no *exact algorithm* for solving the PDPTWT. Computing optimal solutions to larger PDPTWT instances is surely a challenging but worthwhile endeavour.

Acknowledgements Open Access funding provided by Projekt DEAL.

Funding The author has received no funding for this work.

Compliance with ethical standards

Conflict of interest The author herewith confirms that there are no conflict of interest, neither financial nor non-financial.

Appendix

The subsequent Table 3 specifies the parameter settings of the ALNS used for the computational experiments. The following Tables 4, 5, 6, 7, 8 and 9 present the detailed computational results for each of the benchmark instances with these settings. Table 2 was compiled based on these data. Euclidean distances were computed with full double precision, and the objective function values were rounded to three digits.

Table 3 ALNS parameter settings

Parameter	Value
Value for computing start temperature	5
Cooling rate	0.99975
Maximum number of iterations between update of performance statistics	100
Score 1	33
Score 2	9
Score 3	13
Score update factor	0.1
Absolute parameter for determining minimal number of tasks to be removed per iteration	30
Absolute parameter for determining maximal number of tasks to be removed per iteration	60
Relative parameter for determining minimal number of tasks to be removed per iteration	0.1
Relative parameter for determining maximal number of tasks to be removed per iteration	0.4
Randomization degree of worst removal heuristics	3
Randomization degree of Shaw removal heuristics	6
Randomization degree of arc frequency history removal heuristic	6
Randomization degree of zero split removal heuristic	6
Randomization degree of subroute removal heuristic	6
Distance weight parameter of Shaw removal heuristics	9
Time weight parameter of Shaw removal heuristics	3
Load weight parameter of Shaw removal heuristics	2
Number of solutions to be considered for arc frequency history removal	50
Number of iterations (termination criterion)	10,000

Table 4 Detailed computational results size class 100

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)	Ratio runtime linear/constant
lc101	970.473	6326	11	11	10	9	42	10.7
lc102	944.132	6396	10	10	10	9	44	10.4
lc103	1015.599	8318	11	11	10	8	42	10.6
lc104	951.527	3546	10	10	10	9	46	11.6
lc105	986.038	5441	10	10	10	7	43	11.1
lc106	972.096	5194	11	11	10	8	44	10.8
lc107	1013.618	1850	11	11	11	9	45	10.7
lc108	998.764	9249	11	11	11	10	46	10.7
lc109	959.115	3228	10	10	10	6	47	11.3
lc201	691.886	3004	4	4	4	4	71	37.7
lc202	716.538	7761	4	4	4	4	79	35.5
lc203	725.034	282	4	4	4	3	80	36.1
lc204	659.023	8699	4	4	4	4	90	36.1
lc205	661.215	4978	4	4	3	3	71	35.7
lc206	664.417	1897	4	4	4	2	76	38.1
lc207	679.853	1363	4	4	3	3	77	37.5
lc208	676.502	3667	4	4	3	3	79	38.8
lr101	1138.993	7753	11	11	11	7	43	9.6
lr102	1171.374	4794	11	11	11	6	45	10
lr103	1223.4	4819	11	11	10	7	40	9.1
lr104	1090.987	8552	11	11	10	6	41	9.6

Table 4 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)	Ratio runtime linear/constant
lr105	1136.728	9054	11	11	10	6	43	9.6
lr106	1152.816	9850	11	11	11	9	40	9.4
lr107	1216.95	9236	11	11	11	8	40	9.2
lr108	1162.082	6254	11	11	10	6	39	8.8
lr109	1127.926	9309	11	11	12	6	43	9.9
lr110	1109.671	9538	11	11	12	6	40	9.6
lr111	1207.751	9313	11	11	12	9	45	9.5
lr112	1162.763	8856	11	11	11	7	42	9.7
lr201	1041.385	7809	4	4	4	3	75	33.2
lr202	1071.902	7699	3	3	3	3	91	40.6
lr203	969.654	7084	3	3	5	5	107	45.6
lr204	846.491	4767	2	2	2	2	142	57.6
lr205	992.878	7503	3	3	4	3	98	43.1
lr206	975.402	9916	3	3	5	5	103	44.3
lr207	884.184	3222	2	2	2	1	139	57.1
lr208	734.643	414	2	2	2	1	156	57.1
lr209	928.528	2202	3	3	4	4	104	44.4
lr210	938.639	5383	3	3	3	2	109	48.5
lr211	833.859	9747	3	3	3	2	104	45.4
lrc101	1395.622	8464	12	12	12	8	42	8.9
lrc102	1505.317	9916	12	12	12	11	42	8.7

Table 4 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)	Ratio runtime linear/constant
lrc103	1257.967	8890	11	11	11	6	42	9.2
lrc104	1197.082	6581	11	11	11	8	43	9.6
lrc105	1502.65	9346	12	12	14	10	43	8.9
lrc106	1440.623	4938	12	12	12	9	41	8.8
lrc107	1315.167	8371	11	11	12	8	41	9
lrc108	1284.733	7992	11	11	11	7	39	9.2
lrc201	1311.424	9792	4	4	5	4	74	32.1
lrc202	1192.597	5102	3	3	3	2	82	36.5
lrc203	1020.702	9999	3	3	3	3	97	40.2
lrc204	819.319	3083	3	3	3	3	108	41
lrc205	1220.71	5258	4	4	4	2	76	34.6
lrc206	1175.625	2474	3	3	4	4	93	39.9
lrc207	1108.881	2517	3	3	3	3	94	41.3
lrc208	885.852	4444	3	3	3	3	110	42.7

Table 5 Detailed computational results size class 200

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)	Ratio runtime linear/constant
lc1_2_1	3093.29	6607	21	21	21	14	148	11.1
lc1_2_2	3611.993	9872	21	21	22	15	152	11.3
lc1_2_3	3436.951	9878	20	20	21	13	155	11.3
lc1_2_4	3210.84	8006	20	20	21	12	169	12.2
lc1_2_5	3141.552	9966	21	21	22	15	156	11.3
lc1_2_6	3169.339	9874	21	21	21	16	156	11.5
lc1_2_7	3145.988	6885	20	20	20	12	158	11.6
lc1_2_8	3103.838	5116	21	21	21	14	157	11.7
lc1_2_9	3449.571	9980	21	21	21	14	162	11.7
lc1_2_10	3399.748	9111	21	21	22	13	161	11.8
lc2_2_1	2175.912	5701	8	8	9	8	240	36.2
lc2_2_2	2170.04	5847	7	7	7	6	273	37.8
lc2_2_3	2013.353	2762	7	7	7	6	300	38.2
lc2_2_4	1890.106	9606	6	6	6	5	350	39.2
lc2_2_5	1988.849	9398	6	6	6	6	251	38.9
lc2_2_6	2078.159	5568	6	6	6	4	259	38
lc2_2_7	1997.97	4334	6	6	7	6	266	38.5
lc2_2_8	2051.189	5528	7	7	7	6	285	40.7
lc2_2_9	2030.965	3779	7	7	7	6	284	39.2
lc2_2_10	1990.755	4238	6	6	6	5	293	40.1

Table 5 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)	Ratio runtime linear/constant
lr1_2_1	3264.972	7485	10	10	10	8	175	18.8
lr1_2_2	3216.481	7291	10	10	12	9	187	20.3
lr1_2_3	2951.496	8292	10	10	14	11	195	21.9
lr1_2_4	2474.494	4992	9	9	11	7	229	25.8
lr1_2_5	3256.285	9645	11	11	11	7	185	19.1
lr1_2_6	3068.128	4497	11	11	16	11	203	22.1
lr1_2_7	2677.863	8742	10	10	11	8	198	22.3
lr1_2_8	2475.545	9575	8	8	8	7	219	26.2
lr1_2_9	3091.304	8916	10	10	11	9	186	20.4
lr1_2_10	2757.828	7195	10	10	15	11	198	21.9
lr2_2_1	3479.459	8296	5	5	5	3	375	69.4
lr2_2_2	3295.378	9884	4	4	8	8	524	83.2
lr2_2_3	3052.134	8236	4	4	15	13	617	84.6
lr2_2_4	2325.207	7696	4	4	8	8	840	96.6
lr2_2_5	3311.055	8910	4	4	8	7	500	87.7
lr2_2_6	3279.246	9949	4	4	10	8	570	85.3
lr2_2_7	2712.065	9178	3	3	6	6	714	100.9
lr2_2_8	1962.325	294	4	4	6	5	987	120.6
lr2_2_9	3253.515	7558	4	4	7	6	522	89.4
lr2_2_10	2756.037	7993	4	4	7	5	517	85
lrc1_2_1	2892.339	6809	12	12	13	13	175	17.8
lrc1_2_2	2794.063	6133	11	11	13	12	178	19.8

Table 5 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)	Ratio runtime linear/constant
lrc1_2_3	2663.717	8898	9	9	13	11	203	22.8
lrc1_2_4	2442.196	9414	9	9	11	10	230	26.8
lrc1_2_5	3102.716	9166	11	11	11	9	180	18.9
lrc1_2_6	2812.237	8063	12	12	15	12	176	18.9
lrc1_2_7	2826.656	9688	11	11	11	7	187	20.3
lrc1_2_8	2635.937	9878	10	10	10	7	184	20.2
lrc1_2_9	2593.87	9457	10	10	13	10	185	20.7
lrc1_2_10	2553.282	9247	10	10	12	9	206	22.1
lrc2_2_1	2861.418	8416	6	6	12	9	314	53.3
lrc2_2_2	2577.954	6528	6	6	10	9	364	54.5
lrc2_2_3	2393.727	212	5	5	7	6	500	72.4
lrc2_2_4	2241.657	2418	4	4	7	7	657	82.3
lrc2_2_5	2859.009	8398	5	5	18	17	466	72.1
lrc2_2_6	2668.582	9234	5	5	12	9	439	66.1
lrc2_2_7	2544.252	9759	5	5	9	9	434	61.1
lrc2_2_8	2419.297	9696	4	4	8	8	526	73.9
lrc2_2_9	2272.824	6722	4	4	5	5	527	75
lrc2_2_10	2181.054	7304	4	4	11	11	593	75.8

Table 6 Detailed computational results size class 400

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)	Ratio runtime linear/constant
lc1_4_1	8157.242	9894	40	40	42	29	468	10.9
lc1_4_2	8326.408	9606	39	39	39	24	490	11.4
lc1_4_3	8181.272	9487	38	38	40	24	511	11.9
lc1_4_4	7998.63	9611	36	36	35	23	521	12.3
lc1_4_5	8452.623	8748	42	42	42	28	472	11
lc1_4_6	7965.205	7750	40	40	40	25	482	11.2
lc1_4_7	8101.512	9001	40	40	42	28	491	11.3
lc1_4_8	8065.265	9911	39	39	39	26	483	11.1
lc1_4_9	8606.076	9197	38	38	39	27	506	11.4
lc1_4_10	8063.12	9998	37	37	39	19	516	11.5
lc2_4_1	4634.553	8424	13	13	13	9	735	33.9
lc2_4_2	4718.921	9331	13	13	14	9	830	34.9
lc2_4_3	4732.656	5581	13	13	15	13	943	35.3
lc2_4_4	4832.596	5247	13	13	15	12	1048	35.8
lc2_4_5	4793.259	9977	14	14	14	13	760	34.8
lc2_4_6	4330.294	8316	12	12	13	11	810	36.2
lc2_4_7	4455.024	9622	13	13	14	14	830	36.6
lc2_4_8	4542.666	8399	13	13	13	11	843	38
lc2_4_9	4796.374	9561	13	13	15	12	863	36.5

Table 6 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)	Ratio runtime linear/constant
lc2_4_10	4542.475	9835	13	13	15	13	880	37.8
lr1_4_1	7600.421	9901	21	21	24	20	538	18.9
lr1_4_2	7122.111	8661	20	20	20	19	589	21.4
lr1_4_3	6293.12	9987	17	17	19	18	650	24.5
lr1_4_4	5453.027	9352	15	15	17	13	756	28.9
lr1_4_5	7197.859	6366	20	20	26	20	544	22.8
lr1_4_6	6924.309	9903	18	18	20	16	614	22.3
lr1_4_7	6248.194	6972	17	17	23	19	660	26
lr1_4_8	5421.693	7130	15	15	18	15	763	30.5
lr1_4_9	6931.926	9867	19	19	20	17	574	20.8
lr1_4_10	6615.693	6418	18	18	23	20	616	22.9
lr2_4_1	8661.925	9136	9	9	15	12	1198	76.2
lr2_4_2	7568.071	9942	8	8	19	16	1528	83.5
lr2_4_3	6753.73	4193	8	8	25	22	1981	97.3
lr2_4_4	5311.441	9961	6	6	18	16	2868	122
lr2_4_5	7686.022	8490	8	8	17	16	1400	93.2
lr2_4_6	6735.141	4596	8	8	20	17	1678	93.4
lr2_4_7	6054.436	8945	6	6	15	13	2238	114.8
lr2_4_8	5226.712	8953	5	5	13	12	3432	142.5
lr2_4_9	7196.682	9888	8	8	24	22	1508	94
lr2_4_10	6732.745	9932	8	8	24	22	1599	93.7

Table 6 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)	Ratio runtime linear/constant
lrc1_4_1	7147.617	9910	24	24	24	21	493	17.9
lrc1_4_2	6410.331	6771	21	21	24	19	546	20.6
lrc1_4_3	6093.626	8966	19	19	24	19	580	23.4
lrc1_4_4	5340.786	9989	15	15	17	14	681	29.1
lrc1_4_5	7131.683	9877	23	23	27	20	512	17.8
lrc1_4_6	6542.183	9589	22	22	24	16	515	19.3
lrc1_4_7	6635.2	6959	21	21	25	18	547	19.6
lrc1_4_8	6390.169	9938	22	22	23	21	550	19.7
lrc1_4_9	6440.317	7864	20	20	21	16	550	20.7
lrc1_4_10	5877.081	6906	19	19	20	15	565	22.2
lrc2_4_1	6755.846	9929	12	12	27	23	940	59
lrc2_4_2	6372.873	5437	10	10	22	20	1184	69.9
lrc2_4_3	5277.173	8829	8	8	13	13	1489	81.4
lrc2_4_4	4656.752	9893	7	7	24	22	2434	108.1
lrc2_4_5	6432.971	9520	10	10	20	19	1079	65.4
lrc2_4_6	6318.373	3927	10	10	27	21	1094	67.4
lrc2_4_7	5902.167	9607	9	9	21	17	1230	73.5
lrc2_4_8	5560.296	9767	8	8	21	18	1424	81.6
lrc2_4_9	5289.767	9968	8	8	15	15	1467	85.1
lrc2_4_10	5264.808	9125	8	8	25	22	1598	88.1

Table 7 Detailed computational results size class 600

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)
lc1_6_1	16,466.091	9762	62	62	62	43	778
lc1_6_2	16,521.563	9763	61	61	61	37	799
lc1_6_3	16,629.909	8754	58	58	58	38	826
lc1_6_4	16,045.254	9958	57	57	55	34	857
lc1_6_5	17,075.448	9886	64	64	61	45	789
lc1_6_6	17,092.862	9858	64	64	65	44	820
lc1_6_7	16,287.753	9772	61	61	59	36	818
lc1_6_8	17,413.197	9934	62	62	63	43	815
lc1_6_9	18,047.01	9281	62	62	63	35	834
lc1_6_10	17,327.674	8262	59	59	61	41	868
lc2_6_1	10,207.809	9964	22	22	25	22	1092
lc2_6_2	9563.355	8330	19	19	21	17	1204
lc2_6_3	9454.697	9235	20	20	26	22	1352
lc2_6_4	8949.587	9736	20	20	20	19	1505
lc2_6_5	9761.939	9942	20	20	23	22	1137
lc2_6_6	9809.889	9927	22	22	32	28	1189
lc2_6_7	9437.012	9493	21	21	25	23	1216
lc2_6_8	9172.899	8737	21	21	24	20	1263
lc2_6_9	9478.359	9545	21	21	24	23	1263
lc2_6_10	8853.279	9338	20	20	23	21	1367

Table 7 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)
lr1_6_1	16,971.485	9266	31	31	47	37	1010
lr1_6_2	17,411.229	5466	27	25	34	30	1074
lr1_6_3	14,309.551	9961	24	24	31	28	1229
lr1_6_4	12,432.56	8843	18	17	23	21	1406
lr1_6_5	16,143.579	7255	28	28	37	34	1016
lr1_6_6	16,917.078	9557	25	23	34	30	1119
lr1_6_7	13,302.893	9981	23	23	28	26	1215
lr1_6_8	11,141.02	9976	19	19	33	31	1452
lr1_6_9	15,254.064	9497	28	28	36	30	1021
lr1_6_10	16,886.003	3047	26	25	35	34	1088
lr2_6_1	18,358.684	8917	12	12	42	34	2216
lr2_6_2	15,947.769	8049	12	12	45	37	2843
lr2_6_3	14,063.91	7062	10	10	41	37	3651
lr2_6_4	11,140.27	9007	7	7	28	25	5599
lr2_6_5	17,091.142	7637	11	11	60	48	2596
lr2_6_6	15,925.03	9745	10	10	43	32	3230
lr2_6_7	13,519.317	9566	9	9	45	38	4105
lr2_6_8	10,320.749	7862	7	7	29	26	5803
lr2_6_9	15,666.296	9915	10	10	37	29	2819
lr2_6_10	15,595.795	8820	11	11	54	45	2975
lrc1_6_1	13,855.237	9714	32	32	35	25	876
lrc1_6_2	12,973.273	9991	29	29	35	28	969
lrc1_6_3	12,659.233	9834	25	25	34	30	1117

Table 7 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)
lrc1_6_4	9902.372	9652	18	18	25	22	1391
lrc1_6_5	13,645.705	7960	32	32	36	31	898
lrc1_6_6	13,554.014	8787	30	30	32	24	901
lrc1_6_7	12,950.53	8703	29	29	35	27	933
lrc1_6_8	13,145.138	8813	28	28	30	26	931
lrc1_6_9	13,227.627	9645	28	28	29	24	951
lrc1_6_10	12,424.974	9805	26	26	32	23	977
lrc2_6_1	14,628.817	9995	17	17	61	53	1742
lrc2_6_2	13,179.837	9591	14	14	59	50	2131
lrc2_6_3	11,013.262	9799	11	11	32	27	2705
lrc2_6_4	9414.274	9639	8	8	29	25	4247
lrc2_6_5	13,483.235	9753	15	15	39	32	1802
lrc2_6_6	13,981.647	9152	13	13	47	39	1865
lrc2_6_7	12,920.265	8311	11	11	34	28	2220
lrc2_6_8	12,120.2	9915	12	12	43	40	2291
lrc2_6_9	13,734.873	5731	11	11	50	41	2321
lrc2_6_10	12,396.543	9886	10	10	44	35	2618

Table 8 Detailed computational results size class 800

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)
lc181	29,779.459	9751	84	84	84	58	1081
lc182	34,001.982	7969	81	79	76	56	1129
lc183	31,269.812	8397	78	78	79	53	1148
lc184	31,909.582	8508	73	70	70	47	1202
lc185	32,090.767	9405	87	87	84	55	1136
lc186	30,526.773	9015	84	84	83	57	1152
lc187	30,846.858	8924	82	82	80	51	1123
lc188	31,051.107	9965	82	82	83	54	1141
lc189	31,594.426	9834	79	79	75	50	1148
lc1810	31,669.092	9775	79	79	79	52	1147
lc281	15,159.593	9597	28	28	32	25	1484
lc282	16,926.027	9467	30	30	48	39	1628
lc283	19,443.635	8930	28	24	33	32	1727
lc284	17,848.417	9571	27	25	32	26	1922
lc285	14,943.285	9821	27	27	31	23	1631
lc286	15,451.011	9651	29	29	43	34	1766
lc287	15,270.417	9942	27	26	32	26	1835
lc288	14,743.008	9902	28	28	31	28	1823
lc289	14,756.891	9916	28	28	32	27	1804

Table 8 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)
lc2810	15,605.859	7271	29	29	43	36	1923
lr181	41,949.098	9001	38	27	48	42	1431
lr182	32,628.581	9008	37	32	48	45	1513
lr183	30,878.496	9914	29	23	39	35	1686
lr184	21,989.1	9897	22	19	34	34	2082
lr185	32,183.407	9916	39	36	57	48	1406
lr186	33,190.601	9998	35	28	36	30	1515
lr187	25,346.191	9803	29	27	35	33	1684
lr188	20,062.252	7632	23	21	28	26	2083
lr189	34,384.747	8366	37	31	48	43	1406
lr1810	31,676.044	8330	34	27	41	35	1471
lr281	33,711.23	7592	17	16	81	71	2816
lr282	28,988.585	9681	13	13	59	55	3589
lr283	23,193.891	9942	12	11	51	43	4739
lr284	18,490.427	9455	8	8	34	31	7834
lr285	29,125.35	9666	15	14	70	58	3105
lr286	24,148.337	9663	14	14	58	47	3967
lr287	22,686.727	9918	12	12	62	56	5224
lr288	17,010.234	9518	7	7	30	26	7905
lr289	27,858.035	9842	13	13	66	54	3235
lr2810	25,980.059	9423	14	14	70	54	3395

Table 8 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)
lrc181	33,874.714	8637	47	40	48	40	1116
lrc182	29,331.441	9941	44	39	43	35	1230
lrc183	24,651.073	8960	36	33	42	35	1411
lrc184	19,563.444	9296	26	24	35	32	1743
lrc185	32,054.57	9468	45	39	43	38	1179
lrc186	30,576.31	9699	44	38	42	38	1225
lrc187	29,794.101	9970	43	39	50	43	1208
lrc188	26,477.22	9989	39	36	45	37	1268
lrc189	22,577.839	9984	38	38	46	37	1224
lrc1810	24,871.446	9263	37	34	39	35	1258
lrc281	25,832.234	6887	21	20	75	59	2040
lrc282	24,480.59	9191	17	14	51	50	2551
lrc283	19,833.313	9929	16	15	53	44	3133
lrc284	14,549.916	9218	12	12	35	32	4923
lrc285	23,944.816	7576	19	19	103	79	2408
lrc286	24,865.12	8616	17	15	87	76	2401
lrc287	22,520.066	6938	16	16	76	64	2543
lrc288	21,990.194	8369	16	15	84	71	2848
lrc289	19,734.446	9866	15	15	77	68	2864
lrc2810	19,399.705	9992	13	13	39	39	3223

Table 9 Detailed computational results size class 1000

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)
lc1101	65,073.718	9695	108	94	89	64	1310
lc1102	67,649.754	7571	103	90	93	70	1369
lc1103	55,315.955	9803	95	90	93	69	1516
lc1104	53,303.065	7549	92	88	88	63	1521
lc1105	61,452.807	9950	105	95	91	65	1426
lc1106	59,887.447	9518	110	103	96	68	1465
lc1107	55,160.075	9995	105	102	98	72	1494
lc1108	63,357.039	9999	105	96	94	69	1512
lc1109	61,419.643	5994	108	103	108	78	1557
lc11010	54,890.505	9554	97	93	90	62	1597
lc2101	22,527.689	9988	35	34	40	33	1956
lc2102	23,792.805	7876	36	35	51	46	2169
lc2103	23,031.139	9819	34	34	48	42	2321
lc2104	25,118.021	9545	34	32	46	39	2581
lc2105	24,710.473	9475	37	35	41	37	2073
lc2106	23,694.035	9704	36	35	46	40	1995
lc2107	25,279.518	9221	38	36	58	52	2179
lc2108	25,028.226	7300	37	35	49	42	2237
lc2109	25,348.545	8748	37	36	55	50	2259
lc21010	21,934.962	9304	36	36	44	35	2185

Table 9 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)
lr1101	59,691.009	9975	50	35	61	51	1822
lr1102	48,538.5	9997	45	39	64	53	1909
lr1103	44,318.348	9671	37	29	53	48	2215
lr1104	32,194.887	9130	27	22	37	34	2739
lr1105	56,018.817	9783	49	40	68	55	1837
lr1106	45,459.665	9881	42	35	49	43	2026
lr1107	41,921.754	9202	35	26	48	43	2305
lr1108	35,527.347	9499	28	23	48	46	2800
lr1109	48,613.159	9857	46	39	53	50	1888
lr11010	47,691.898	8048	41	32	59	52	1968
lr2101	51,690.254	9868	21	17	99	83	3275
lr2102	45,995.484	9789	18	16	70	64	4190
lr2103	36,607.173	9896	16	13	43	42	5416
lr2104	27,180.136	9254	12	12	59	54	9069
lr2105	46,941.618	9595	18	15	77	67	3775
lr2106	41,616.661	8730	17	15	70	61	4741
lr2107	34,561.985	9174	14	12	64	59	6095
lr2108	26,083.772	9832	10	10	50	46	10,024
lr2109	43,679.631	8723	18	15	86	72	4052
lr21010	41,890.994	9910	16	13	75	72	4336
lrc1101	56,643.406	9370	59	44	57	42	1698
lrc1102	48,314.071	9827	56	48	71	60	1814

Table 9 continued

Instance	Objective function value	Iteration where best solution was found	No. routes	No. LTC routes	No. subroutes	No. PTLs used	Runtime ALNS with constant-time test (s)
lrc1103	39,926.935	9798	46	39	51	45	2051
lrc1104	32,640.504	7727	33	29	50	45	2665
lrc1105	54,919.086	9631	58	47	58	44	1717
lrc1106	55,289.112	9903	55	39	55	50	1708
lrc1107	46,741.471	9827	50	39	51	42	1758
lrc1108	50,108.697	9452	51	39	61	52	1790
lrc1109	46,641.948	9651	50	39	52	45	1803
lrc11010	39,637.235	9998	47	42	59	51	1871
lrc2101	39,684.964	9595	24	20	101	86	2937
lrc2102	37,322.263	9847	23	20	102	81	3435
lrc2103	30,658.094	9759	19	18	80	69	4325
lrc2104	25,217.808	9934	14	13	56	50	6626
lrc2105	38,071.655	9897	21	16	88	74	3442
lrc2106	33,867.241	9521	21	21	118	97	3379
lrc2107	33,795.104	9081	18	16	83	73	3720
lrc21010	33,794.739	8358	18	16	85	71	4416

References

Baldacci R, Bartolini E, Mingozzi A (2011) An exact algorithm for the pickup and delivery problem with time windows. Oper Res 59(2):414–426
Battarra M, Cordeau JF, Iori M (2014) Pickup-and-delivery problems for goods transportation. In: Toth P, Vigo D (eds) Vehicle routing: problems, methods, and applications. Society for Industrial and Applied Mathematics, Philadelphia, pp 161–191
Bent R, Van Hentenryck P (2006) A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Comput Oper Res 33:875–893
Berbeglia G, Cordeau JF, Laporte G (2010) Dynamic pickup and delivery problems. Eur J Oper Res 202(1):8–15
Bürckert H, Fischer K, Vierke G (2000) Holonic transport scheduling with TELETRUCK. Appl Artif Intell 14:697–725
Cheung R, Shi N, Powell W, Simão H (2008) An attribute-decision model for cross-border drayage problem. Transp Res Part E 44(2):217–234
Cuda R, Guastaroba G, Speranza M (2015) A survey on two-echelon routing problems. Comput Oper Res 55:185–199
Derigs U, Pullmann M, Vogel U (2013) Truck and trailer routing—problems, heuristics and computational experience. Comput Oper Res 40(2):536–546
Doerner K, Salazar-González J (2014) Pickup-and-delivery problems for people transportation. In: Vigo D, Toth P (eds) Vehicle routing: problems, methods, and applications. Society for Industrial and Applied Mathematics, Philadelphia, pp 193–212
Drexl M (2011) Branch-and-price and heuristic column generation for the generalized truck-and-trailer routing problem. J Quant Methods Econ Bus Admin 12:5–38
Drexl M (2012) Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. Transp Sci 46(3):297–316
Flatberg T, Hasle G, Kloster O, Nilssen E, Riise A (2005) Dynamic and stochastic aspects in vehicle routing—a literature survey. Technical report STF90A05413, SINTEF
Funke B, Grünert T, Irnich S (2005) Local search for vehicle routing and scheduling problems: review and conceptual integration. J Heurist 11(4):267–306
Grangier P, Gendreau M, Lehuédé F, Rousseau LM (2016) An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. Eur J Oper Res 254(1):80–91
Gschwind T, Drexl M (2019) Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. Transp Sci 53(2):480–491
Irnich S (2008a) Resource extension functions: properties, inversion, and generalization to segments. OR Spect 30(1):113–148
Irnich S (2008b) A unified modeling and solution framework for vehicle routing and local search-based metaheuristics. INFORMS J Comput 20(2):270–287
Irnich S, Funke B, Grünert T (2006) Sequential search and its application to vehicle-routing problems. Comput Oper Res 33(8):2405–2429
Irnich S, Schneider M, Vigo D (2014) Four variants of the vehicle routing problem. In: Toth P, Vigo D (eds) Vehicle routing: problems, methods, and applications. Society for Industrial and Applied Mathematics, Philadelphia, pp 241–271
Kindervater G, Savelsbergh M (1997) Vehicle routing: handling edge exchanges. In: Aarts E, Lenstra J (eds) Local search in combinatorial optimization. Wiley, Chichester, pp 337–360
Li H, Lim A (2003) A metaheuristic for the pickup and delivery problem with time windows. Int J Artif Intell Tools 12:173–186
Lin S, Yu V, Lu C (2011) A simulated annealing heuristic for the truck and trailer routing problem with time windows. Expert Syst Appl 38(12):15244–15252
Masson R, Lehuédé F, Péton O (2013a) An adaptive large neighborhood search for the pickup and delivery problem with transfers. Transp Sci 47(3):344–355
Masson R, Lehuédé F, Péton O (2013b) Efficient feasibility testing for request insertion in the pickup and delivery problem with transfers. Oper Res Lett 41(3):211–215
Parragh S, Cordeau JF (2017) Branch-and-price and adaptive large neighborhood search for the truck and trailer routing problem with time windows. Comput Oper Res 83:28–44

Parragh S, Doerner K, Hartl R (2008a) A survey on pickup and delivery models part I: transportation between customers and depot. J Betriebswirtschaft 58(1):21–51
Parragh S, Doerner K, Hartl R (2008b) A survey on pickup and delivery models part II: transportation between pickup and delivery locations. J Betriebswirtschaft 58(2):81–117
Parragh S, Doerner K, Hartl R (2010) Variable neighborhood search for the dial-a-ride problem. Comput Oper Res 77(6):58–71
Parragh S, Schmid V (2013) Hybrid column generation and large neighborhood search for the dial-a-ride problem. Comput Oper Res 40(1):490–497
Pisinger D, Ropke S (2010) Large neighborhood search. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics, second edition. International series in operations research and management science, vol 146. Springer, Boston, pp 399–419
Prins C (2004) A simple and effective evolutionary algorithm for the vehicle routing problem. Comput Oper Res 31(12):1985–2002
Prodhon C, Prins C (2014) A survey of recent research on location-routing problems. Eur J Oper Res 238(1):1–17
Ropke S, Cordeau JF (2009) Branch and cut and price for the pickup and delivery problem with time windows. Transp Sci 43(3):267–286
Ropke S, Cordeau JF, Laporte G (2007) Models and branch-and-cut algorithms for pickup-and-delivery problems with time windows. Networks 49(4):258–272
Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp Sci 40(4):455–472
Rothenbächer AK, Drexl M, Irnich S (2018) Branch-and-price-and-cut for the truck-and-trailer routing problem with time windows. Transp Sci 52(5):1174–1190
Savelsbergh M (1985) Local search in routing problems with time windows. Ann Oper Res 4(1):285–305
Savelsbergh M (1990) An efficient implementation of local search algorithms for constrained routing problems. Eur J Oper Res 47(1):75–85
Savelsbergh M (1992) The vehicle routing problem with time windows: minimizing route duration. ORSA J Comput 4(2):146–154
Shaw P (1997) A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report. Department of Computer Science, University of Strathclyde
Solomon M (1987) algorithms for the vehicle routing and scheduling problems with time window constraints. Oper Res 35:254–265
Tilk C, Bianchessi N, Drexl M, Irnich S, Meisel F (2018) Branch-and-price-and-cut for the active-passive vehicle-routing problem. Transp Sci 52(2):300–319
Vidal T, Crainic TG, Gendreau M, Prins C (2013) A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. Comput Oper Res 40(1):475–489
Vidal T, Crainic TG, Gendreau M, Prins C (2014) A unified solution framework for multi-attribute vehicle routing problems. Eur J Oper Res 234(3):658–673
Villegas J, Prins C, Prodhon C, Medaglia A, Velasco N (2011) A GRASP with evolutionary path relinking for the truck and trailer routing problem. Comput Oper Res 38(9):1319–1334
Villegas J, Prins C, Prodhon C, Medaglia A, Velasco N (2013) A matheuristic for the truck and trailer routing problem. Eur J Oper Res 230(2):231–244
Wen M, Larsen J, Clausen J, Cordeau JF, Laporte G (2009) Vehicle routing with cross-docking. J Oper Res Soc 60:1708–1718
Xue Z, Zhang C, Lin W, Miao L, Yang P (2014) A tabu search heuristic for the local container drayage problem under a new operation mode. Transp Res Part E 62:136–150