On Testing Capacity Constraints in Pickup-and-Delivery Problems with Trailers in Amortized Constant Time

Technical Report LM-2018-08

Michael Drexl
Faculty of Applied Natural Sciences and Industrial Engineering
Deggendorf Institute of Technology, Deggendorf, Germany
and
Chair of Logistics Management, Gutenberg School of Management and Economics,
Johannes Gutenberg University, Mainz, Germany
E-mail: michael.drexl@th-deg.de

26th November 2018

Abstract

Efficient feasibility tests are important in many heuristics for routing problems. This paper considers several variants of pickup-and-delivery problems with trailers. Its contribution consists in the description of amortized constant-time procedures for testing observance of capacity constraints when inserting tasks into routes. It is demonstrated that the presence of vehicles with detachable trailers makes capacity feasibility tests considerably more involved.
Keywords: Pickup-and-Delivery; Trailers; Constant-Time Feasibility Test

1 Introduction

Pickup-and-delivery problems (PDPs) are concerned with the transport of goods or persons from different origins to different destinations. PDPs come in several variants and have received a lot of attention in the literature (see the surveys by Parragh et al. [19, 20], Doerner and Salazar-González [5], and Battarra et al. [1]). Whereas vehicle routing problems (VRPs, the special case of the PDP where either all pickups or all deliveries occur at a central depot) with trailers are rather well examined (see the surveys by Prodhon and Prins [21], Section 3.3, Cuda et al. [4], Section 4, and the more recent works by Parragh and Cordeau [18] and Rothenbächer et al. [23]), PDPs with trailers (PDPTs) are rarely studied in the literature, despite their practical relevance. There are several works on PDPs 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 (for example, Cheung et al. [3], Xue et al. [29], Tilk et al. [27]), but this author is aware of only two papers (Bürckert et al. [2], Drexl [6]) dealing with less-than-truckload tasks, i.e., the case where more than one task can be on a vehicle at the same time. The present paper is concerned with PDPTs of the latter type.

Heuristics based on neighbourhood search are widely used approaches for solving many different kinds of vehicle routing problem, including the various types of PDPs. A decisive aspect for the performance of such methods is their ability to quickly test the feasibility of an insertion of a customer or task into a route. There are several theoretical papers on efficient feasibility testing, for example, Hunsaker and Savelsbergh [12], Haugland and Ho [11], Firat and Woeginger [7] (dial-a-ride problems), and Masson et al. [17] (PDP with time windows and transfers). In a similar vein, the present work deals with testing whether the insertion of a task into a feasible PDPT route maintains capacity feasibility of the route.

The contribution of the paper consists in the description of amortized constant-time feasibility tests for several variants of PDPTs. 'Amortized constant-time' means that the tests require constant time, independent of the number of tasks on the route, but use auxiliary data which must be computed in a preprocessing step that does not run in constant time.
The presentation focuses on capacity tests. An efficient test for time window feasibility that can be used for all PDP(T) types considered in this paper is presented in Drexl [6], based on earlier work by Savelsbergh [26] and Masson et al. [17]. Other seminal papers on feasibility tests for routing problems are Savelsbergh [24, 25], Kindervater and Savelsbergh [16], Funke et al. [8], Irnich et al. [15], Irnich [13, 14], Vidal et al. [28], Grangier et al. [9], and Gschwind and Drexl [10], but none of these considers routing problems with trailers.
The rest of the paper is structured as follows. In the next section, the PDP(T) types of interest in this work are specified. Then, the notation used in the paper and the formal modelling of the different PDPT variants are explained in Section 3. Afterwards, Section 4 describes how capacities can be tested in linear and in amortized constant time. Section 5 concludes the paper with a summary and an outlook.

2 PDP Variants

Following the PDP classification of Battarra et al. [1], the problem variants covered in this paper are:

- One-to-one problems: each task consists in the transport of a particular *commodity* from a given pickup location to a given delivery location. The transported commodities are not interchangeable. This is obvious for passenger transport. In goods transport, applications arise, e.g., in letter mail and parcel services as well as full and less-than-truckload forwarding.
- One-to-many-to-one problems: each task consists in either the transport of a commodity from a central depot to a delivery location (linehaul tasks) or the transport from a pickup location to the depot (backhaul tasks). Practical applications are common in supplying supermarkets, beverage stores, apparel stores and, above all, in the less-than-truckload business. Several subtypes of one-to-many-to-one problems can be distinguished:
  - The VRP with mixed backhauls: linehaul and backhaul tasks can occur anywhere on a route.
  - The VRP with backhauls: all linehaul tasks on a route must be fulfilled before a pickup location of a backhaul task can be visited. In this and the previous subtype, it is possible that a route fulfils only linehaul or only backhaul tasks.
  - The VRP: the special case where all tasks are either linehaul or backhaul tasks.
  - The simultaneous PDP: any task location may require a delivery of a good from the depot and a pickup of another good for transport back to the depot; the delivery and the pickup must be performed during one visit.

  Note that the goods to be transported in one-to-many-to-one problems may be interchangeable or not. For linehaul tasks, this makes a difference. For example, in soft drink distribution, crates with full bottles are delivered to households, and empty crates are taken back. It is irrelevant whether household A receives crate 1 of a particular soft drink brand and household B receives crate 2 of the same brand or the other way round. By contrast, in freight forwarding of consignments on pallets, each loaded pallet of a linehaul task has a specific destination. For the algorithmic treatment of such problems, these issues do not matter. What counts is that the capacity requirements of each task are taken into account correctly. In reality, however, if goods are not interchangeable, the driver must ensure that the right items are delivered to linehaul customers.
- Many-to-many PDPs: goods picked up at a location can be used to fulfil demand at several other locations (e.g., bike-sharing systems, empty container movement, raw milk bulk transports). Mostly, only a single commodity is considered in this problem type. Many-to-many PDPs come in four sub-variants: it may be allowed or forbidden that a vehicle leave or enter the depot with some load.

For all these problem variants, it is sensible to consider the use of vehicles with detachable trailers. Trailers are advantageous as they increase the overall vehicle capacity. However, some locations may be accessible only by a lorry without a trailer, e.g., because of insufficient manoeuvring space. Therefore, special parking and transshipment locations (PTLs) are available where trailers can be parked while lorries visit accessibility-constrained locations. This induces a non-trivial trade-off between an enlarged vehicle capacity and the necessity of making detours to park and reattach trailers, and, as will be demonstrated in the following, makes capacity feasibility tests considerably more involved.

3 Notation and Modelling

The following subsections describe how each of the above PDPT variants is represented by a suitable digraph $D = (V, A)$. This is helpful to understand which insertions of tasks into routes are possible and how capacity feasibility can be tested. In all variants, each task t has a pickup capacity requirement $q_p^t \geq 0$ and a delivery capacity requirement $q_d^t \leq 0$. The vertex set V of D contains a start and an end depot vertex, s and e respectively, one pickup and one delivery vertex for each task, and one vertex for each PTL. The pickup and the delivery vertices are referred to as *task vertices*. Each vertex $v \in V$ has an associated capacity requirement $q_v \in \mathbb{R}$, which is zero for the depot and the PTL vertices. Moreover, for modelling some of the above PDPT variants, 'artificial' task vertices are used. Each task vertex is visited exactly once in a feasible solution, whereas PTL vertices can be visited more than once. All artificial vertices are reachable with a trailer. The arc set A, in principle, contains one arc for each pair of vertices, and, to model the possibility of consecutive *subroutes* (definition in next paragraph), there is a loop on each PTL vertex. Some exceptions apply: there is no arc entering s, leaving e, leading from s to a delivery vertex, from a pickup vertex to e, or from a delivery vertex to the associated pickup vertex. Moreover, there are additional restrictions on the arc sets of the different problem variants as specified in the respective subsections below.

Associated with D is a vehicle fleet made up of single lorries and LTCs. All vehicles start their routes at s and end them at e. Each vehicle k has a lorry capacity of Q_k^l and a trailer capacity of Q_k^t. For single lorries k, $Q_k^t = 0$. The route of an LTC consists of the *main route*, where the lorry pulls its trailer, and zero or more *subroutes* starting and ending at a PTL where the trailer is parked while the lorry performs one or more pickups and/or deliveries. Several consecutive subroutes can start and end at the same PTL before the trailer is re-coupled and pulled away. If the pickup of a task whose delivery is performed on a subroute has been performed before this subroute, the entire load of the task must be on the lorry at the start of the subroute. This may require a load transfer from a trailer to its lorry when decoupling. In the present paper, it is assumed that such load transfers are possible without restrictions, and that, at pickup locations reachable by trailer, the load to be picked up can be split arbitrarily between a lorry and its trailer.

Although costs and times are not relevant for capacity tests, note that the locations associated with any two vertices u and v determine the distance-dependent costs and the travel times of the arcs (u, v) and (v, u), if these exist, and are therefore relevant for the definition of the objective function and for time-window feasibility tests. With regard to the PDPTs considered here, for non-artificial ('real') vertices, travel costs and times are computed on the basis of the respective physical locations, and artificial task vertices are located at the start or the end depot, as indicated in each case.

3.1 The One-to-One PDPT

The one-to-one PDPT is the simplest case. For any task t, the associated pickup vertex has a capacity requirement of $q_p^t > 0$, and the delivery vertex requires $q_d^t < 0$ capacity units. There is an arc between each pair of vertices, with the exceptions mentioned above.

3.2 The VRPT with Mixed Backhauls

Extending the approach by Ropke and Pisinger [22] to trailers, this problem type is modelled as follows. Each linehaul task is represented by an artificial pickup vertex located at the central depot and a delivery vertex at the physical delivery location. Similarly, for each backhaul task, there is a pickup vertex at the physical pickup location and an artificial delivery vertex at the depot. For a linehaul or backhaul task t, the capacity requirements of linehaul and backhaul pickup and delivery vertices are $q_p^t > 0$ and $q_d^t < 0$. The only arcs leading to an artificial pickup vertex come from the start depot vertex or other artificial pickup vertices, the only arcs emanating from an artificial delivery vertex lead to other artificial delivery vertices or to the end depot. There is no arc from an artificial pickup to an artificial delivery vertex.

3.3 The VRPT with Backhauls

When all linehauls must be performed before any backhaul, the setup described in the previous subsection can be modified as follows. No arc exists from an artificial pickup vertex to a real pickup vertex or from a real pickup vertex to a real delivery vertex. Moreover, it must be ensured that the precedence requirements are not undermined by visiting PTLs; i.e., sequences such as $\ldots \rightarrow$ linehaul delivery $\rightarrow$ decoupling $\rightarrow$ backhaul pickup $\rightarrow$ coupling $\rightarrow$ linehaul delivery $\rightarrow \ldots$ must not be allowed. This can be achieved by storing, for each route, the position of the last linehaul delivery. When an insertion is tested and the pickup vertex is a pickup of a backhaul, i.e., a non-artificial pickup, insertion of this vertex is considered only after the last linehaul delivery position. Similarly, the last insertion position to consider for a non-artificial delivery of a linehaul is after the last non-artificial delivery. Ropke and Pisinger [22] use a different concept, that of precedences, which is applicable in the presence of trailers as well.

3.4 The VRPT

Vehicle routing problems with trailers can be represented as instances of VRPTs with mixed backhauls without requiring further modelling.

3.5 The Simultaneous PDPT

Ropke and Pisinger [22] propose the following modelling approach for the simultaneous PDP: each task location requiring a delivery and a pickup is represented by a 'delivery task' and a 'pickup task'. The delivery (pickup) vertex of each pickup (delivery) task is located at the depot. Moreover, the costs of all arcs emanating from a delivery vertex of a delivery task are set to a very high value, except for the arc leading to the pickup vertex of the partner task, which has cost zero. These high-cost arcs are necessary, because otherwise, inserting a single task would be impossible. After inserting a single task, whether a delivery or a pickup task, the next task that is inserted will always be the partner task, because this will remove a high-cost arc and will thus dramatically improve the current solution.
An alternative approach that avoids having to create two tasks for each original task, i.e., avoids doubling the instance size, and prevents difficulties with precedences in the presence of PTLs works as follows. Each task t is represented by an artificial pickup vertex located at the start depot and a delivery vertex located where the simultaneous delivery and pickup must occur. The amount to be picked up at the artificial pickup vertex equals q_d^t, the original amount to be *delivered*. The actual amount to be delivered at the delivery vertex is set to $q_p^t - q_d^t$, i.e., to (original pickup amount − original delivery amount). This means that delivery *vertices* with *positive* capacity requirement are possible (note that the delivery amount of a *task* is always non-positive), and that there can be different absolute values for the pickup and the delivery amount of a task; see the example in Section 4.2.1. No arc exists from a delivery vertex or from a PTL vertex to an artificial pickup vertex.

3.6 The Many-to-Many PDPT

Similar to the modelling of backhauling problems, a task is represented by an artificial vertex located at the depot, and similar to the approach for the simultaneous PDP(T), the absolute values of the pickup and the delivery amount of a task may differ. If the task consists of a pickup (delivery) at a vertex, the artificial vertex is located at the end (start) depot. The capacity requirement at an artificial vertex is zero; its partner vertex has the original positive (if it is a pickup) or negative (if it is a delivery) capacity requirement. The only arcs leading to an artificial pickup vertex come from the start depot or other artificial pickup vertices, the only arcs emanating from an artificial delivery vertex lead to other artificial delivery vertices or the end depot. Moreover, there is no arc from an artificial pickup to an artificial delivery vertex.

Many-to-many PDP(T)s are special in (at least) two respects. First, as mentioned, four subvariants are possible: it may be allowed or forbidden that a vehicle leave or enter the depot with some load.

When no loading at the start depot is allowed but load may be brought to the end depot, it should be tested in a preprocessing step whether the total load to be picked up is at least as much as the total load to be delivered. If this is not the case, no feasible solution exists. Moreover, on each route, sufficiently many pickups must be performed before any deliveries are possible, and it must be ensured in the capacity tests that the total load of a single lorry or an LTC is always non-negative. Likewise, if it is forbidden to bring load to the end depot, but loading at the start depot is allowed, it should be tested in advance whether the total load to be delivered is at least as much as the total load to be picked up, and one or more delivery tasks must be inserted into an empty route before any pickup task. If both conditions are required, a route with one task is necessarily infeasible, and the insertion of a task into a feasible route also makes the route infeasible. This can be handled by allowing (penalized) infeasible solutions in a surrounding algorithm that uses the capacity test. Then, the return value of the test should not be simply 'true' or 'false', but the amount by which the vehicle capacity and/or the zero load balance at the start and/or end depot is/are violated.

Second, many-to-many PDP(T)s are special because they possess the property, rarely encountered in vehicle routing problems, that *removing* a task from a feasible route can make the route infeasible. This holds true irrespective of whether or not it is allowed to pick up or deliver load at the depot. For example, consider the following route, performed by a single lorry with capacity 100 (the artificial pickup and delivery vertices at the depot are omitted for simplicity):

Vertex	s	1	2	3	4	5	6	e
Capacity requirement	0	+50	+50	–50	–50	+50	–30	0

Removing vertex 5 makes the route infeasible even if loading at the depot is allowed. In the present paper, however, the focus is on (feasible) insertions, so issues arising when removing tasks are not discussed further here.

4 Feasibility Tests

Generally speaking, for a route to be feasible regarding capacity constraints, it must be ensured that the vehicle capacity is not exceeded at any vertex, and that enough load can be aboard the LTC (when on the main route) or the lorry (when on a subroute) to satisfy any subsequent negative capacity requirements. Put differently, it must be ensured on the main route and on all subroutes that, when reaching a vertex with positive capacity requirement, the load in the LTC or the lorry plus this positive capacity requirement does not exceed the LTC or the lorry capacity; likewise, when reaching a vertex with negative capacity requirement, enough load must be in the LTC or the lorry to satisfy this capacity requirement. It will become clear in the following that in particular the latter aspect is non-trivial on subroutes and, for the many-to-many PDPT, also on the main route.

The algorithms and data structures presented in this section are extensions of those described in Drexl [6] for the one-to-one PDPT. The algorithmic descriptions assume that the feasibility of an insertion of a task t with associated pickup vertex p and delivery vertex d, written as $t = (p,d)$, into a feasible route $r = (0,1,\ldots,n-1,n)$ which is performed by single lorry or LTC k is to be tested. Vertex p is to be inserted directly after position (zero-based index of the route) h; d is to be inserted directly after position/index i. If p cannot be reached with a trailer, r is performed by an LTC, and the trailer is attached when leaving h, a location triple $\tilde{p} = ptl_p \rightarrow p \rightarrow ptl_p$ corresponding to a new subroute must be inserted after h; similar for i and d. ptl_p is a PTL vertex; similar for d. Note that p, d, ptl_p, and ptl_d are vertices, whereas h and i are indices on a route. However, to simplify notation, when referring to a vertex visited at a certain position on a route, only the index is used: for example, the capacity requirement of the vertex at index i is denoted by q_i.
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 $\tilde{p} = ptl_p \rightarrow p \rightarrow ptl_p$ must 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 $\tilde{d} = ptl_d \rightarrow d \rightarrow ptl_d$ is inserted. If $h == i$ and p and/or d must be surrounded by a decouple-couple pair, it would also be possible to surround both p and d by one such pair, but this makes no difference for capacity feasibility. Several consecutive subroutes by one LTC lorry at the same PTL are modelled by inserting a decouple-couple pair for each subroute. For simplicity, the vehicle index k is omitted: Q^l and Q^t are used instead of Q^l_k and Q^t_k to denote the lorry and the trailer capacity. The capacity requirements of the task $t = (p,d)$ to be inserted are indicated by q_p and q_d. For each problem type, q_p and q_d are determined from the original capacity requirements of task t as described in Sections 3.1–3.6. In the pseudocode, the symbol '==' serves as equality operator, '=' is the assignment operator, and 'x += y' is used as shortcut for '$x = x + y$'.

4.1 Testing Capacities in Linear Time

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 visited vertex is added to the total load and compared with the lorry capacity, cf. lines 16–20 in Algorithm 1. By contrast, testing capacity in the presence of trailers is not entirely straightforward. The procedure detailed in Algorithm 1 can be used. The procedure covers all problem types considered in this paper, because large parts of the routine apply to all types. As indicated in the pseudocode, lines 2–15 can be omitted for problem types other than many-to-many, and lines 23–30 can be ignored for many-to-many-problems.
For problem types other than many-to-many, 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 23–30) and stored in an array LoadDeliveredButNotPickedUpOnSubroute. Note that, in practice, it is not sufficient to have the correct *amount* of load in the lorry at the start of a subroute. It is also necessary to have the right *commodities* on the lorry, those that must be delivered on this subroute, but this must be ensured by the driver.
For many-to-many problems, one forward pass over the route is performed to determine how much load must be picked up at the depot. This amount is determined by the most negative load balance at any vertex on the route. If loading at the start depot is not allowed and a negative load balance occurs at a vertex, the insertion is infeasible (lines 2–8). If load may be picked up at the start depot, a second pass over the route (lines 17–20 or 34–52) tests whether there is sufficient capacity taking this load into account. If no load may be brought to the end depot, but the load balance from any index to the end of the route is positive, the route is infeasible as well (lines 9–14). Moreover, for many-to-many problems, it must be ensured for all subroutes that at each non-artificial delivery vertex, sufficient load is available on the lorry to perform the delivery. This load may have been picked up earlier on the subroute or before the subroute. It is possible

that not enough load has been picked up earlier on the subroute and that, at the same time, not enough load has been picked up before the subroute. This is tested in lines 50–52.

4.2 Testing capacities in constant time

In the following, it is described how capacity feasibility can be tested in constant time. Algorithm 2 is for the one-to-one and the one-to-many-to-one problem types. The constant-time test for many-to-many PDPTs differs significantly from the tests for the other problem types. Thus, it is presented separately, in Algorithm 3.

4.2.1 One-to-One and One-to-Many-to-One PDPTs

The following data structures are used in the test for the one-to-one and the one-to-many-to-one PDPT variants. These were introduced by Drexl [6] for the one-to-one PDPT.

1. MaxTotalLoadOfSegment[i][*offset*] is 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] is 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 in a one-to-one PDPT:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Capacity requirement	0	+30	+10	0	+20	–30	+10	+15	–10	–10	0	–20	–15	0

   This route contains a subroute that starts at index 3 and ends at index 10. The load balances at indices 1 to 6 are +30, +40, +40, +60, +30, and +40; thus, MaxTotalLoadOfSegment[1][5] = +60. Moreover, MaxTotalLoadOfSegment[7][0] = +55.
2. TrailerAttached[i] is true if the trailer is attached when leaving i, false otherwise.
3. IndexOfLastPrecedingDecouple[i] stores the index of the last decoupling that precedes i.
4. If i is the start of a subroute, LoadDeliveredButNotPickedUpOnSubroute[i] is the sum of the pickup amounts of those tasks on the subroute whose pickups lie before i.
5. If i is on a subroute, LoadBalanceFromStartOfSubroute[i] is the positive, negative or zero load balance from the start of the subroute up to and including i.
   In the above example route, LoadBalanceFromStartOfSubroute[9] = $-5 = 20 - 30 + 10 + 15 - 10 - 10$.
6. OffsetOfNextCoupling[i] stores the number of positions from i to the next index of a coupling process.
7. If i is on a subroute, MaxLoadBalanceFromStartOfSubroute[i][*offset*] stores 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 example, MaxLoadBalanceFromStartOfSubroute[5][3] = +15 = $\max(0, -10, 0, +15, +5) = \max(0,$ LoadBalanceFromStartOfSubroute[7]).

As an example of the simultaneous PDPT, consider the following five tasks:

Task	t_1	t_2	t_3	t_4	t_5
Original amount to be delivered	+5	+20	+10	+12	+16
Original amount to be picked up	+10	+15	+10	+18	+13
q_p^t	+5	+20	+10	+12	+16
q_d^t	5	–5	0	+6	–3

Assume these five tasks are fulfilled by the following route:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Task	–	t_1	t_2	t_3	t_4	t_5	t_5	–	t_1	t_3	t_2	–	t_4	–
Vertex type	s	P	P	P	P	P	D	Dec	D	D	D	Co	D	e
Artificial?	No	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No
Capacity requirement	0	+5	+20	+10	+12	+16	–3	0	+5	0	–5	0	+6	0

Algorithm 1 TestCapacityLinear(r, k)

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

Result: Returns **true** iff lorry and, if applicable, trailer capacity of k are maintained at each vertex of r; **false** otherwise

```
TotalLoad = 0
if Problem type is many-to-many
    LoadBalance = 0
    for v = 1,2,3,...,n-1
        LoadBalance += q_v
        TotalLoad = min(TotalLoad, LoadBalance)
        if Loading at start depot is not allowed and TotalLoad < 0
            return false
    if Unloading at end depot is not allowed
        LoadBalance = 0
        for v = n-1, n-2, n-3, ..., 1
            LoadBalance += q_v
            if LoadBalance > 0
                return false
    TotalLoad = (-1) · TotalLoad
if Q^t == 0 // Test for single lorries
    for v = 1,2,3,...,n-1
        TotalLoad += q_v
        if TotalLoad > Q^l
            return false
else // Test for LTCs
    LoadDeliveredButNotPickedUpOnSubroute = array of integers of length n+1, initialized to 0
    if Problem type is not many-to-many
        IndexOfLastDecouple = 0
        for v = 1,2,3,...,n-1
            if v is a decoupling vertex
                IndexOfLastDecouple = v
            if Trailer is not attached upon leaving v
                if v is a delivery vertex and corresponding pickup v' < IndexOfLastDecouple
                    LoadDeliveredButNotPickedUpOnSubroute[IndexOfLastDecouple] += q_v'
    MinLorryLoadSinceLastDecouple = 0
    MaxLorryLoad = 0
    MaxLorryLoadSinceLastDecouple = 0
    for v = 1,2,3,...,n-1
        TotalLoad += q_v
        if TotalLoad > Q^l + Q^t
            return false
        MaxLorryLoad = min(TotalLoad, Q^l)
        if v is a decoupling vertex
            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 MaxLorryLoadSinceLastDecouple < 0
                return false
return true
```

This route contains a subroute that starts at index 7 and ends at index 11. The load balances at indices 2 to 10 are +25, +35, +47, +63, +60, +60, +65, +65, and +60; thus, MaxTotalLoadOfSegment[2][8] = +65. Moreover, MaxTotalLoadOfSegment[12][0] = +66, LoadBalanceFromStartOfSubroute[10] = 0, and MaxLoadBalanceFromStartOfSubroute[10] = +5 = max(0, +5, +5, 0) = LoadBalanceFromStartOfSubroute[8] = LoadBalanceFromStartOfSubroute[9].

The decisive aspect for the feasibility test of simultaneous PDPTs is that the delivery amount is not necessarily the negative of the pickup amount. In algorithm 2, this is taken into account in lines 6–8 and 32–36.

MaxTotalLoadOfSegment and MaxLoadBalanceFromStartOfSubroute can be initialized and updated with a nested loop, by iterating over all indices $j \geq i$ for each index i. The other data structures can be determined in a single loop. Hence, computing the preprocessing data is possible in quadratic time in the number of tasks on the route.

Algorithm 2 uses these data to test 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. It is evident that the running time of the algorithm is constant, independent of the number of vertices visited on the route.

Algorithm 2 returns false in line 4 if the total capacity is less than the maximal total load between the pickup and the delivery plus the pickup amount. For the simultaneous PDPT, false is returned in line 8 if the former condition is fulfilled or the total capacity is less than the maximal load between the delivery and the end of the route plus the pickup plus the delivery amount. The latter condition must be tested because, as explained in Section 3.5, for the simultaneous PDPT the sum of the pickup and the delivery amount indicates the change of load at deliveries.

The algorithm returns false in line 23 if p is to be inserted on a subroute and the minimal load that must be in the lorry when leaving the decoupling vertex plus the load balance from the start of the subroute up to h plus q_p plus the maximal load balance, counted from the beginning of the subroute, from $h+1$ to i or to the end of the subroute, whichever vertex is visited earlier, exceeds the lorry capacity.

False is also returned if d is to be inserted on a subroute and the minimal load that must be in the lorry when leaving the decoupling vertex plus the maximal load balance from the start of the subroute up to i plus q_p exceeds the lorry capacity (line 31).

Moreover, for the simultaneous PDPT, similar to the situation on the main route, the algorithm returns false in line 36 if the minimal load aboard the lorry when leaving i plus q_d plus the maximal load balance from $i+1$ to the end of the subroute exceeds the lorry capacity.

Finally, algorithm 2 returns false in line 43 if d is to be inserted directly after p on a subroute and the total load upon leaving h plus q_p exceeds the lorry capacity.

If Algorithm 2 returns false from line 4, 8, or 23, further potential insertion positions for d need not be tested with the current insertion position of p. Instead, the next potential position for inserting p can be considered. Hence, it is sufficient to execute lines 2–23 of Algorithm 2 only once for each h.

Note also that lines 14–23 and 37–43 in Algorithm 2 are never reached for the simultaneous PDPT, as a pickup is never on a subroute.

4.2.2 The Many-to-Many PDPT

In the capacity test for the many-to-many PDPT, all data structures from Section 4.2.1 are employed, except for LoadDeliveredButNotPickedUpOnSubroute. In addition, the following data structures are used:

For testing the total vehicle capacity:

1. OffsetsToWhereActualTotalLoadIsMaximal indicates, for each index i on the route, the non-negative offset from i to the index where the overall load in the vehicle, not taking into account what might have to be picked up at the depot, is maximal.
2. MaxActualTotalLoad stores the maximal physical load from each index i until the end of the route, not taking into account any load that might have to be picked up at the depot. This load

Algorithm 2 TestCapacityConstantOneToOneAndOneToManyToOnePDPT(p,d,r,h,i,k)

Input: Pickup-and-delivery task $t=(p,d)$ with capacity requirements $q_p \geq 0$ and $q_d \in \mathbb{R}$
Route $r=(0,1,2,\ldots,n)$
Indices of insertion positions h,i with $0 \leq h \leq i \leq n-1$
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** iff inserting p or a triple $\tilde{p} = ptl_p \rightarrow p \rightarrow ptl_p$ into r after index h and d or a triple $\tilde{d} = ptl_d \rightarrow d \rightarrow ptl_d$ after i (or, iff $h == i$, after p or $\tilde{p}$) is feasible regarding lorry and trailer capacity, **false** otherwise

```
// Evaluate feasibility of insertion regarding total capacity
if Problem type is not simultaneous
    if Q^l + Q^t < MaxTotalLoadOfSegment[h][i − h] + q_p
        return false
else
    if Q^l + Q^t < max{MaxTotalLoadOfSegment[h][i − h] + q_p,
                       MaxTotalLoadOfSegment[i][n − i] + q_p + q_d}
        return false
if Q^t == 0
    return true
// Evaluate feasibility of insertion regarding lorry capacity
if i > h // Delivery not directly after pickup
    // Evaluate feasibility of insertion of pickup
    if TrailerAttached[h] == false // Trailer not attached when leaving h
        ind = IndexOfLastPrecedingDecouple[h]
        MinLoadAtDecouple = max(MaxTotalLoadOfSegment[ind][0] − Q^t,
                                LoadDeliveredButNotPickedUpOnSubroute[ind])
        LoadAfterPickup = MinLoadAtDecouple + LoadBalanceFromStartOfSubroute[h] + q_p
        offset = OffsetOfNextCoupling[h + 1]
        if h + OffsetOfNextCoupling[h] ≥ i
            offset = i − h
        if LoadAfterPickup + MaxLoadBalanceFromStartOfSubroute[h + 1][max(0, offset − 1)] > Q^l
            return false
    // Evaluate feasibility of insertion of delivery
    if TrailerAttached[i] == false
        if i − h ≥ OffsetOfNextCoupling[h] // Delivery not on same subroute as pickup
            ind = IndexOfLastPrecedingDecouple[i]
            MinLoadAtDecouple = max(MaxTotalLoadOfSegment[ind][0] − Q^t,
                                    LoadDeliveredButNotPickedUpOnSubroute[ind])
            if MinLoadAtDecouple + MaxLoadBalanceFromStartOfSubroute[ind][i − ind] + q_p > Q^l
                return false
            if Problem type is simultaneous
                offset = OffsetOfNextCoupling[i + 1]
                if MinLoadAtDecouple + q_p + LoadBalanceFromStartOfSubroute[i]
                  + q_d + MaxLoadBalanceFromStartOfSubroute[i + 1][offset] > Q^l
                    return false
else // i == h, i.e., delivery directly after pickup
    if TrailerAttached[h] == false
        ind = IndexOfLastPrecedingDecouple[h]
        MinLoadAtDecouple = max(MaxTotalLoadOfSegment[ind][0] − Q^t,
                                LoadDeliveredButNotPickedUpOnSubroute[ind])
        if MinLoadAtDecouple + LoadBalanceFromStartOfSubroute[i] + q_p > Q^l
            return false
return true
```

is aboard the vehicle at index $i + \mathsf{OffsetsToWhereActualTotalLoadIsMaximal}[i]$.
As an example, consider the following route:

Index	0	1	2	3	4	5	6	7	8	9
Capacity requirement	0	+50	+10	–70	+30	+25	–5	+15	–25	0

For this route, $\mathsf{MaxActualTotalLoad}[3] = +65 = 0 + 30 + 25 - 5 + 15$, and $\mathsf{OffsetsToWhereActualTotalLoadIsMaximal}[3] = 4$.

3. MostNegativeLoadBalanceFromStart indicates, for each index i, the most negative load balance (or zero if there are only pickups up to i) from the start of the route to i. The term 'most negative' is used instead of 'smallest' or 'minimal' to point out that all values are less than or equal to zero. A 'minimal' or 'smallest' element of an arbitrary set of numbers may be positive.
   In the example, $\mathsf{MostNegativeLoadBalanceFromStart}[2] = 0$, and $\mathsf{MostNegativeLoadBalanceFromStart}[3] = -10$.
4. MostNegativeLoadBalancesFromPosToEndOfRoute specifies, for each index i, the most negative load balance (or zero if there are only pickups) from i until the end of the route.
   In the example, for indices 3, 4, and 6, MostNegativeLoadBalancesFromPosToEndOfRoute equals -70, 0, and -15.
5. $\mathsf{LoadBalanceFromTo}[i][\mathit{offset}]$ is the load balance from i to $i + \mathit{offset}$.
   In the example, $\mathsf{LoadBalanceFromTo}[2][2] = -30$, $\mathsf{LoadBalanceFromTo}[3][4] = -5$, and $\mathsf{LoadBalanceFromTo}[4][1] = +55$.

For testing lorry capacities on subroutes:

1. $\mathsf{MostNegativeLoadBalanceOnSubrouteFromTo}[i][\mathit{offset}]$ is the most negative load balance from i until $i + \mathit{offset}$ on the subroute where i is visited, counted from the beginning of the subroute.
   As an example, consider the following route:

Index	0	1	2	3	4	5	6	7	8	9
Capacity requirement	0	+50	0	–20	+30	–25	–5	0	–25	0

   This route contains a subroute that starts at index 2 and ends at index 7. $\mathsf{MostNegativeLoadBalanceOnSubrouteFromTo}[2][\mathit{offset}]$ equals zero for $\mathit{offset} = 0$ and -20 for $\mathit{offset} = 1, \ldots, 4$. For index 4 and $\mathit{offset} = 0$, 1, and 2, the values are 0, -15, and -20.
2. $\mathsf{MostNegativeLoadBalanceOnSubrouteFromAnyVertexBeforeUntilPos}[i]$ specifies the most negative load balance from any vertex that precedes i on the subroute, including i, up to i.
   In the example, the values for indices 2–6 are 0, -20, 0, -25, -30.
3. $\mathsf{MostNegativeLoadBalanceFromIUntilEndOfSubroute}[i]$ indicates the most negative load balance on the segment from any index i until the end of the subroute, counted from i.
   In the above example, the values of MostNegativeLoadBalanceFromIUntilEndOfSubroute, for indices 2–6, are -20, -20, 0, -30, and -5.

LoadBalanceFromTo and MostNegativeLoadBalanceOnSubrouteFromTo 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; the other data structures can be filled or updated in one pass. As explained in Section 4.2, $\mathsf{MaxTotalLoadOfSegment}[i][\mathit{offset}]$ is the maximal load balance from the start of the route at any index from i up to and including $i + \mathit{offset}$. For the many-to-many problem where loading at the start depot is allowed, MaxTotalLoadOfSegment also takes into account the load that must be picked up at the depot, by adding this load to each component of MaxTotalLoadOfSegment. Computing the load to be picked up at the depot can be done in one additional forward pass over the route, which does not change the time complexity of determining the preprocessing data. Hence, the preprocessing data for a many-to-many-PDPT route can be computed in quadratic time in the number of tasks on the route. With these data structures, capacity feasibility can be tested as described in Algorithm 3. It is easy to see that the algorithm itself runs in constant time.

Algorithm 3 returns false in line 8 if the maximal load in the vehicle, disregarding any load brought from the depot, at any index from and including h until the end of the route (i.e., $\mathsf{MaxActualTotalLoad}[h]$) plus q_p plus the rest of the load that still needs to be picked up at the depot

even after inserting p exceeds the total vehicle capacity. To exemplify this, consider the following route and assume $h = 3$ and $q_p = 40$:

Index	0	1	2	3	4	5	6	7
Capacity requirement	0	–10	+10	+10	–30	+30	–100	0

It is easy to see that 90 units of load must be picked up at the depot for the route to be feasible, 10 units of which are needed at index 1, 10 at index 4, and 70 at index 6. The maximal load in the vehicle anywhere after h, not taking into account load brought from the depot, is 30 at index 5. Up to this point, 20 units of load picked up at the depot have been unloaded. Hence, the expression in line 5 equals $(-1)\cdot(-90)+(-20) = +70$. The value to be subtracted from q_p in line 6 is $(-1)\cdot(-90)+(-10) = +80$. This value is greater than q_p, so that the overall value of line 6 is negative, and this means that the entire amount q_p can be used to substitute load brought from the depot. This, in turn, means that inserting p does not increase the load at any index, which implies that the insertion is feasible. Now assume that the capacity requirement at index 6 is -50. Then, 40 units must be picked up at the depot, and line 5 equals $(-1)\cdot(-40)+(-20) = +20$. In line 6, $+30$ must be subtracted from q_p, so that the value of line 6 is $+40-30 = +10$. This means that 30 units of q_p can be used to help satisfy the negative capacity requirements at indices 4 and 6, so that the 30 units picked up at the depot to this end are no longer necessary. This, in turn, means that inserting p after h increases the load from $h+1$ onwards by 10 units, which may or may not violate the total vehicle capacity.

The return false in line 11 is self-explanatory.

The return false in line 19 has the following rationale: $q_p \leq 0$ implies that $|q_d| > 0$. To satisfy this negative capacity requirement, additional load can be picked up at the depot if this is allowed and if there is free capacity up to index i. In any case, for the insertion of d to be feasible, it must still be possible to satisfy all negative capacity requirements after i, which, in other words, means that the maximal possible load after d must be at least as much as the absolute value of the most negative load balance computed from i until the end of the route.

The return false in line 35 can be explained as follows. The minimal load aboard the lorry when leaving a decoupling vertex must be the maximum of the following quantities: (i) the absolute value of the most negative load balance on the subroute up to and including h, (ii) the absolute value of the most negative load balance on the subroute from $h+1$ until the end of the subroute minus q_p, and (iii) the total load in the vehicle when leaving the decoupling vertex minus the trailer capacity. Then, when p is inserted, the load after p is the sum of this minimal load, the load balance from the start of the subroute up to h, and q_p. If this sum exceeds the lorry capacity, the insertion is infeasible.

The algorithm returns false in line 40 if the absolute value of the most negative load balance on the subroute on which d is to be inserted, computed from any index before i up to i, plus $|q_d|$ exceeds the lorry capacity, as this quantity is a lower bound for the amount of load that must be aboard the lorry after decoupling when d shall be inserted after i.

Finally, Algorithm 3 returns false in line 51 when, after inserting d directly behind i, not enough load can be in the lorry to satisfy the most negative load balance on the subroute after i. This is tested as follows. The load that can be in the lorry at the decoupling vertex equals the load balance from the start of the route up to the decoupling vertex plus the maximal additional load that can be picked up at the depot. The load at i is the minimum of the lorry capacity and the load at the decoupling vertex plus the most positive load balance from the start of the subroute up to i plus the most negative load balance on the subroute from any vertex before up to i. The load after visiting d is then equal to the load at i plus q_d, and this load must be non-negative and greater than or equal to the absolute value of the most negative load balance from $i+1$ to the end of the subroute.

Note that lines 21 ff. are valid whether or not loading at the start depot is allowed.

An alternative feasibility testing procedure for the many-to-many PDP without trailers is described by Kindervater and Savelsbergh [16].

Algorithm 3 TestCapacityConstantManyToManyPDPT(p,d,r,h,i,k)

Input: Pickup-and-delivery task (p,d) with capacity requirements $q_p \geq 0$ and $q_d \leq 0$
Route $r = (0,1,2,\ldots,n)$
Indices of insertion positions h,i with $0 \leq h \leq i \leq n-1$
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** iff inserting p or a triple $\tilde{p} = ptl_p \rightarrow p \rightarrow ptl_p$ into r after index h and d or a triple $\tilde{d} = ptl_d \rightarrow d \rightarrow ptl_d$ after i (or, iff $h == i$, after p or $\tilde{p}$) is feasible regarding lorry and trailer capacity, **false** otherwise

```
// Evaluate feasibility of insertion regarding total capacity
if q_p > 0
    offset = OffsetsToWhereActualTotalLoadIsMaximal[h]
    if MaxActualTotalLoad[h]
        +((−1)·MostNegativeLoadBalanceFromStart[n] + MostNegativeLoadBalanceFromStart[h+offset])
        +(q_p − ((−1)·MostNegativeLoadBalanceFromStart[n] + MostNegativeLoadBalanceFromStart[h]))
        > Q^l + Q^t
        return false
    if Unloading at end depot is not allowed
        if max(0, LoadBalanceFromTo[0][h]) + q_p + LoadBalanceFromTo[h+1][n−(h+1)] > 0
            return false
else
    MaxAdditionalLoadThatCanBePickedUpAtDepot = 0
    if Loading at start depot is allowed
        MaxAdditionalLoadThatCanBePickedUpAtDepot = Q^l + Q^t − MaxTotalLoadOfSegment[0][i]
    MaxTotalLoadAfterDelivery =
        MaxTotalLoadOfSegment[i][0] +MaxAdditionalLoadThatCanBePickedUpAtDepot + q_d
    if MaxTotalLoadAfterDelivery < max(0,(−1)·MostNegativeLoadBalancesFromPosToEndOfRoute[i+1])
        return false
// Evaluate feasibility of insertion regarding lorry capacity
if Q^t > 0 and i > h  // Vehicle is an LTC and delivery is not directly after pickup
    // Evaluate feasibility of insertion of pickup
    if q_p > 0 and TrailerAttached[h] == false
        ind = IndexOfLastPrecedingDecouple[h]
        MinLoadAtDecouple = (−1)·MostNegativeLoadBalanceOnSubrouteFromTo[ind][h−ind]
        offset = OffsetOfNextCoupling[h+1]
        MinLoadAtDecouple =
            max(MinLoadAtDecouple,(−1)·MostNegativeLoadBalanceOnSubrouteFromTo[h+1][offset]−q_p)
        MinLoadAtDecouple = max(MinLoadAtDecouple, MaxTotalLoadOfSegment[ind][0] − Q^t)
        LoadAfterPickup = MinLoadAtDecouple + LoadBalanceFromStartOfSubroute[h] + q_p
        offset = OffsetOfNextCoupling[h+1]
        if h + OffsetOfNextCoupling[h] ≥ i
            offset = i − h
        if LoadAfterPickup + MaxLoadBalanceFromStartOfSubroute[h+1][max(0,offset−1)] > Q^l
            return false
    // Evaluate feasibility of insertion of delivery
    if q_d < 0 and TrailerAttached[i] == false
        ind = IndexOfLastPrecedingDecouple[i]
        if (−1)·MostNegativeLoadBalanceOnSubrouteFromAnyVertexBeforeUntilPos[i] + (−1)·q_d > Q^l
            return false
        MaxAdditionalLoadThatCanBePickedUpAtDepot = 0
        if Loading at start depot is allowed
            MaxAdditionalLoadThatCanBePickedUpAtDepot = Q^l + Q^t − MaxTotalLoadOfSegment[0][ind]
        PossibleLorryLoadAtDecouple =
            MaxTotalLoadOfSegment[ind][0] + MaxAdditionalLoadThatCanBePickedUpAtDepot
        LoadAtPredOfDelivery =
            min(Q^l, PossibleLorryLoadAtDecouple + MaxLoadBalanceFromStartOfSubroute[ind][i−ind])
            +MostNegativeLoadBalanceOnSubrouteFromAnyVertexBeforeUntilPos[i]
        LoadAfterDelivery = LoadAtPredOfDelivery + q_d
        if LoadAfterDelivery < max(0,(−1)·MostNegativeLoadBalanceFromIUntilEndOfSubroute[i+1])
            return false
return true
```

5 Conclusion and Outlook

This paper has studied several variants of the PDPT, a routing problem where transport tasks between pickup and delivery locations must be fulfilled by a fleet of capacity-constrained single lorries and lorry-trailer combinations subject to accessibility restrictions at locations. Procedures to test the capacitive feasibility of inserting a task into an existing route have been presented. The procedures run in constant time, given appropriate auxiliary data that can be computed in a preprocessing step.

In their survey, Battarra et al. [1] write (p. 181): 'When looking at the pickup and delivery literature as a whole, one cannot fail to notice that there exists a very large number of problem variants which differ in their structure but nevertheless share many similarities. One can thus hope to see the development of general modelling and solution techniques capable of handling multiple variants with a unified framework.' The author hopes that the present paper constitutes a step in this direction.

In any case, there is ample opportunity for further research: throughout the paper at hand, it was assumed that the goods to be transported are homogeneous and load can be split arbitrarily between a lorry and its trailer. This is true for many practical situations, most notably, for the transport of liquids, bulk cargo, and palleted goods. However, in all problem variants, the transported goods might just as well consist of a number of distinct items with different sizes, so that load splitting between lorry and trailer is possible only in discrete amounts. The transfer of load between a lorry and its trailer can even be completely forbidden or impossible for technical reasons. Taking this into account would require major changes to the procedures described above. It is thus beyond the scope of the present work and left for future research.

References

[1] 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. 161–191.

[2] Bürckert H, Fischer K, Vierke G (2000): *Holonic Transport Scheduling with TELETRUCK*. Applied Artificial Intelligence 14: 697–725.

[3] Cheung R, Shi N, Powell W, Simão H (2008): *An Attribute-Decision Model for Cross-Border Drayage Problem*. Transportation Research Part E 44(2): 217–234.

[4] Cuda R, Guastaroba G, Speranza M (2015): *A Survey on Two-Echelon Routing Problems*. Computers & Operations Research 55: 185–199.

[5] 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. 193–212.

[6] Drexl M (2018): *On the One-to-One Pickup-and-Delivery Problem with Time Windows and Trailers*. Technical Report 1816, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz.

[7] Firat M, Woeginger G (2011): *Analysis of the Dial-a-Ride Problem of Hunsaker and Savelsbergh*. Operations Research Letters 39(1): 32–35.

[8] Funke B, Grünert T, Irnich S (2005): *Local Search for Vehicle Routing and Scheduling Problems: Review and Conceptual Integration*. Journal of Heuristics 11(4): 267–306.

[9] 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*. European Journal of Operational Research 254(1): 80–91.

[10] Gschwind T, Drexl M (2016): *Adaptive Large Neighborhood Search with a Constant-Time Feasibility Test for the Dial-a-Ride Problem*. Transportation Science, to appear.

[11] Haugland D, Ho S (2010): *Feasibility Testing for Dial-a-Ride Problems*. In: Chen B (ed): *Algorithmic Aspects in Information and Management. AAIM 2010*. Lecture Notes in Computer Science 6124. Springer, Berlin. 170–179.

[12] Hunsaker B, Savelsbergh M (2002): *Efficient Feasibility Testing for Dial-a-Ride Problems*. Operations Research Letters 30(3): 169–173.

[13] Irnich S (2008): *Resource Extension Functions: Properties, Inversion, and Generalization to Segments*. OR Spectrum 30(1): 113–148.

[14] Irnich S (2008): *A Unified Modeling and Solution Framework for Vehicle Routing and Local Search-Based Metaheuristics*. INFORMS Journal on Computing 20(2): 270–287.

[15] Irnich S, Funke B, Grünert T (2006): *Sequential Search and its Application to Vehicle-Routing Problems*. Computers & Operations Research 33(8): 2405–2429.

[16] Kindervater G, Savelsbergh M (1997): *Vehicle Routing: Handling Edge Exchanges*. In: Aarts E, Lenstra J (eds): *Local Search in Combinatorial Optimization*. Wiley, Chichester. 337–360.

[17] Masson R, Lehuédé F, Péton O (2013): *Efficient Feasibility Testing for Request Insertion in the Pickup and Delivery Problem with Transfers*. Operations Research Letters 41(3): 211–215.

[18] Parragh S, Cordeau JF (2017): *Branch-and-Price and Adaptive Large Neighborhood Search for the Truck and Trailer Routing Problem with Time Windows*. Computers & Operations Research 83: 28–44.

[19] Parragh S, Doerner K, Hartl R (2008): *A Survey on Pickup and Delivery Models Part I: Transportation between Customers and Depot*. Journal für Betriebswirtschaft 58(1): 21–51.

[20] Parragh S, Doerner K, Hartl R (2008): *A Survey on Pickup and Delivery Models Part II: Transportation between Pickup and Delivery Locations*. Journal für Betriebswirtschaft 58(2): 81–117.

[21] Prodhon C, Prins C (2014): *A Survey of Recent Research on Location-Routing Problems*. European Journal of Operational Research 238(1): 1–17.

[22] Ropke S, Pisinger D (2006): *A Unified Heuristic for a Large Class of Vehicle Routing Problems with Backhauls*. European Journal of Operational Research 171(3): 750–775.

[23] Rothenbächer AK, Drexl M, Irnich S (2018): *Branch-and-Price-and-Cut for the Truck-and-Trailer Routing Problem with Time Windows*. Transportation Science 52(5): 1174–1190.

[24] Savelsbergh M (1985): *Local Search in Routing Problems with Time Windows*. Annals of Operations Research 4(1): 285–305.

[25] Savelsbergh M (1990): *An Efficient Implementation of Local Search Algorithms for Constrained Routing Problems*. European Journal of Operational Research 47(1): 75–85.

[26] Savelsbergh M (1992): *The Vehicle Routing Problem with Time Windows: Minimizing Route Duration*. ORSA Journal on Computing 4(2): 146–154.

[27] Tilk C, Bianchessi N, Drexl M, Irnich S, Meisel F (2018): *Branch-and-Price-and-Cut for the Active-Passive Vehicle-Routing Problem*. Transportation Science 52(2): 300–319.

[28] Vidal T, Crainic TG, Gendreau M, Prins C (2014): *A Unified Solution Framework for Multi-Attribute Vehicle Routing Problems*. European Journal of Operational Research 234(3): 658–673.

[29] 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*. Transportation Research Part E 62: 136–150.