Transportation Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Adaptive Large Neighborhood Search with a Constant-Time Feasibility Test for the Dial-a-Ride Problem

Timo Gschwind, Michael Drexl

http://pubsonline.informs.org/journal/trsc/

TRANSPORTATION SCIENCE
Vol. 53, No. 2, March–April 2019, pp. 480–491
ISSN 0041-1655 (print), ISSN 1526-5447 (online)

Adaptive Large Neighborhood Search with a Constant-Time Feasibility Test for the Dial-a-Ride Problem

Timo Gschwind,[a] **Michael Drexl**[a, b]

[a] Gutenberg School of Management and Economics, Johannes Gutenberg University, D-55099 Mainz, Germany; [b] Faculty of Applied Natural Sciences and Industrial Engineering, Deggendorf Institute of Technology, D-94469 Deggendorf, Germany

Contact: gschwind@uni-mainz.de, http://orcid.org/0000-0002-7715-4994 (TG)

Received: December 22, 2016
Revised: December 21, 2017
Accepted: February 25, 2018
Published Online in Articles in Advance: January 14, 2019

https://doi.org/10.1287/trsc.2018.0837

Abstract. In the dial-a-ride problem, user-specified transport requests from origin to destination points have to be served by a fleet of homogeneous vehicles. The problem variant we consider aims at finding a set of minimum-cost routes satisfying constraints on vehicle capacity, time windows, maximum route duration, and maximum user ride times. We propose an adaptive large neighborhood search (ALNS) for its solution. The key novelty of the approach is an exact amortized constant-time algorithm for evaluating the feasibility of request insertions in the repair steps of the ALNS. In addition, we use two optional improvement techniques: a local-search-based intraroute improvement of routes of promising solutions using the Balas–Simonetti neighborhood and the solution of a set covering model over a subset of all routes generated during the search. With these techniques, the proposed algorithm outperforms the state-of-the-art methods in terms of solution quality. New best solutions are found for several benchmark instances.

Funding: This research was partially funded by the Deutsche Forschungsgemeinschaft [Grant IR 122/7-1].
Supplemental Material: The online appendix is available at https://doi.org/10.1287/trsc.2018.0837.

Keywords: dial-a-ride problem • adaptive large neighborhood search • feasibility testing

1. Introduction

The dial-a-ride problem (DARP) is concerned with the transport of persons from origin to destination locations by a given vehicle fleet to minimize an objective while respecting a set of constraints related to aspects of user inconvenience. Practical applications are widespread and occur, for example, in door-to-door transport of schoolchildren, patients, and elderly or disabled people, as well as public transport services in rural areas. There are static and dynamic versions of DARPs. In the static version, all transport requests are known in advance. In the dynamic case, requests appear successively over time and must be accommodated into a route plan that is already being executed. The present paper considers what is now commonly referred to as the standard version of the DARP (introduced by Cordeau and Laporte 2003): a static setting where the objective is to minimize routing costs while satisfying constraints on vehicle capacity, time windows, maximum route duration, and maximum user ride times. Mathematical models for the DARP are developed by Cordeau (2006) and Ropke, Cordeau, and Laporte (2007). Cordeau and Laporte (2007) and Doerner and Salazar-González (2014) provide surveys on the problem.

The main contribution of this paper is an exact amortized constant-time algorithm for checking the feasibility of a given route. By "exact" we mean that the algorithm will declare a route as feasible if and only if the route is feasible. "Amortized constant time" means that the check itself takes constant time and is independent of the number of requests in the route, but that, to perform the check, auxiliary data computed in a preprocessing step are used. The preprocessing takes cubic time in the number of requests, but it is performed only once for a given solution. The resulting data are then used for all feasibility checks, that is, for all potential insertion positions of all unplanned requests. This routine is embedded in an adaptive large neighborhood search (ALNS) metaheuristic with two optional improvement techniques: a local-search-based intraroute improvement of all routes of promising solutions by means of the Balas and Simonetti (2001) neighborhood and the solution of a set covering problem over a subset of all routes generated during the search. With these techniques, the proposed algorithm outperforms the state-of-the-art methods in terms of solution quality, and new best solutions are found for several benchmark instances.

The remainder of this paper is structured as follows. In the next section, a brief overview of recent literature on DARPs is given. In Section 3, we formally describe the problem under study. Section 4 gives details on the ALNS and the improvement techniques we use. Section 5 describes our feasibility testing routine, Section 6 presents and discusses the results of the computational experiments performed with our algorithm, and Section 7 concludes this paper with an outlook on promising avenues for future research.

2. Related Work

Comprehensive surveys on the DARP literature are provided by Cordeau and Laporte (2007) and Doerner and Salazar-González (2014). In the following, we concentrate on the more recent contributions, on important problem extensions, and on work focusing on feasibility tests.

There are only few papers on exact approaches for the DARP: Cordeau (2006) and Ropke, Cordeau, and Laporte (2007) describe branch-and-cut algorithms, Ropke and Cordeau (2009) and Gschwind and Irnich (2015) present branch-and-price-and-cut procedures. Qu and Bard (2015) study an extension of the DARP where vehicles can be configured before service begins to handle different request types. They also apply a branch-and-cut-and-price algorithm. In general, the approaches based on path variables show superior performance. However, the largest instances that have been solved to optimality have no more than 96 requests and eight vehicles, and have tight time windows and ride time constraints. Therefore, heuristics are needed to solve larger and less tightly constrained instances in reasonable time.

Important contributions on heuristics for the DARP include those by Cordeau and Laporte (2003) (tabu search); Jain and Van Hentenryck (2011) (large neighborhood search); Parragh, Doerner, and Hartl (2010) (variable neighborhood search); Kirchler and Wolfler Calvo (2013) (granular tabu search); Parragh and Schmid (2013) (matheuristic combining column generation and large neighborhood search); Ritzinger, Puchinger, and Hartl (2016) (large neighborhood search with recreate procedures based on dynamic programming); Masson, Lehuédé, and Péton (2014) (large neighborhood search); Braekers, Caris, and Janssens (2014) (threshold accepting); and Chassaing, Duhamel, and Lacomme (2016) (evolutionary local search).

As for feasibility testing of a given route, most of the above papers use or slightly adapt the eight-step evaluation scheme introduced by Cordeau and Laporte (2003), which is based on the forward time slack concept introduced by Savelsbergh (1992). The scheme tries to find a feasible schedule by sequentially minimizing violations of time windows, maximum route duration, and maximum ride times. It has a worst-case running time complexity of $\mathcal{O}(r^2)$ (r being the number of stops in the route).

Hunsaker and Savelsbergh (2002) propose a routine for checking the feasibility of a route in a DARP with an additional upper bound on the waiting time. In three passes along the route, they compute and update values for the arrival, departure, and waiting time at each node. The routine has linear-time complexity, but it is only a heuristic and does not guarantee to find a feasible schedule if one exists. Haugland and Ho (2010) modify the Hunsaker and Savelsbergh (2002) heuristic and develop an exact procedure with $\mathcal{O}(r \log r)$ complexity. Tang et al. (2010) present an algorithm with a worst-case run-time complexity of $\mathcal{O}(r^2)$. They also try to gradually build up a feasible schedule directly on the network of the DARP instance. Firat and Woeginger (2011) provide an $\mathcal{O}(r)$ check. Contrary to the aforementioned approaches, these authors reformulate the scheduling problem as a system of difference constraints and reduce the problem to one of determining whether or not a special digraph associated with such a system has a negative-weight cycle. Through an appropriate formulation of the system, this test can be performed in linear time.

Chassaing, Duhamel, and Lacomme (2016) perform an empirical comparison of the feasibility-checking routines of Cordeau and Laporte (2003) and Firat and Woeginger (2011). As mentioned, the former has a theoretical time complexity quadratic in the number of stops in a route, the latter has a linear one. The computational findings of Chassaing, Duhamel, and Lacomme (2016) confirm these theoretical results.

Gschwind (2015b) derives route feasibility tests for the synchronized pickup and delivery problem (SPDP) introduced by Gschwind (2015a). The SPDP is a generalization of the DARP with maximum *and minimum* ride time constraints. Gschwind (2015b) adapts the approaches of Tang et al. (2010) and Firat and Woeginger (2011) to the SPDP. Computational results indicate that, despite the better worst-case run-time complexity of the latter approach, the former shows a superior performance in the average case.

Parragh et al. (2009) and Molenbruch et al. (2017) present tailored feasibility tests for problems where the objective function consists of or contains the minimization of the average or total user ride time. Parragh et al. (2009) modify the checking procedure of Cordeau and Laporte (2003) by changing the computation of the forward time slack to avoid an increase of ride times caused by a delay of the start of service at destination vertices. The procedure described by Molenbruch et al. (2017) is initiated with a schedule where total user ride time is at its lower bound. In five consecutive algorithmic steps, it tries to reach a feasible solution while avoiding, as far as possible, increasing the total user ride time.

Berbeglia, Pesant, and Rousseau (2011) and Häeme and Hakula (2015) also deal with feasibility testing in the DARP context. However, these authors consider a dynamic setting and are concerned with determining whether or not a feasible solution for a given *problem instance* exists. This is clearly a different focus. Berbeglia, Pesant, and Rousseau (2011) apply constraint programming, and Häeme and Hakula (2015) develop an approach based on dynamic programming. The algorithms proposed by these authors are not directly applicable to the situation we consider.

Recently, several practically relevant generalizations of the DARP have been studied. These include, in particular, multiple objectives (e.g., transport costs and user inconvenience; Parragh et al. 2009), a heterogeneous fleet (e.g., vehicles with different capacities; Parragh 2011), and customers with different transport requirements (e.g., specialized equipment for accommodating elderly or handicapped people; Parragh et al. 2012). Good overviews of the literature on generalized DARPs are given by Braekers, Caris, and Janssens (2014) and Molenbruch et al. (2017). Moreover, the option of passengers changing vehicles during their trip, very common in public transport, has been studied for the DARP (Masson, Lehuédé, and Péton 2014, Schönberger 2016). Allowing such transfers introduces intricate interdependencies *between* routes and requires their synchronization in space and time. No constant-time feasibility test is known for this problem extension.

3. Problem Description

The DARP version under study here is defined on a complete directed graph $G = (V, A)$ with node set V and arc set A. The node set $V = \{0, 2n+1\} \cup P \cup D$ is composed of nodes 0 and $2n+1$, which denote the origin and destination depots, respectively; the pickup nodes $P = \{1, \dots, n\}$; and the delivery nodes $D = \{n+1, \dots, 2n\}$ of n transport requests. Each transport request $i = 1, \dots, n$ has an associated pickup node i and an associated delivery node $i+n$. For convenience, a request i is identified by its pickup node, and we use the set P also to refer to requests. A homogeneous fleet K of vehicles, each with a capacity of Q, is located at the origin depot 0 to serve the transport requests. A maximum route duration L limits the time between the start and the end of each vehicle route. Similarly, a maximum ride time L_i limiting the time between pickup and corresponding delivery is given for each request $i \in P$. With each node $i \in V$, a time window $[a_i, b_i]$, a nonnegative service duration s_i, and a demand q_i with $q_i = -q_{i+n}$ are associated. Finally, a travel time t_{ij} and a routing cost c_{ij} are associated with each arc $(i, j) \in A$. We assume that both routing costs and travel times are nonnegative and satisfy the triangle inequality.

A feasible solution of the DARP serves all transport requests exactly once using at most $|K|$ vehicle routes. Each route has to start at the origin depot and end at the destination depot, and must not take longer than L units of time. The service at each node i has to start within the time window $[a_i, b_i]$. Waiting prior to the service is allowed at any node and any time. Pickup node i and corresponding delivery node $i+n$ of each request have to be served on the same route, and the pickup must precede the delivery. Moreover, the service at the delivery node $i+n$ must start at most L_i units of time after the service at the pickup node i has been completed. The capacity of the vehicles must not be exceeded at any time. The objective is to minimize the total routing costs.

4. Adaptive Large Neighborhood Search

ALNS is one of the most successful metaheuristic approaches for a broad range of routing problems (see the survey by Pisinger and Ropke 2010). It is based on the LNS principle introduced by Shaw (1997). Basically, LNS works as follows. Given an incumbent solution, several of its elements are iteratively removed and reinserted to create a new solution, which is accepted if it improves on the best solution found so far or fulfils some other acceptance criterion. Ropke and Pisinger (2006) extended the LNS scheme by adding different removal and reinsertion operators and an adaptive operator selection scheme.

The ALNS we use closely follows their approach, but employs several additional removal procedures. We adopt all removal and reinsertion operators described in their paper (random, worst, and Shaw removal; basic greedy, regret-2, regret-3, regret-4, and regret-M reinsertion), use a roulette wheel method with adaptive weight adjustment for selecting the removal and reinsertion operators in each iteration, and apply a simulated annealing acceptance criterion. The additional removal strategies built into our ALNS are as follows. An *arc frequency history removal* heuristic, proposed by Masson, Lehuédé, and Péton (2013), aims to remove requests that appear to be badly positioned in the current solution with regard to the best-known solutions (BKSs). The heuristic keeps track of the number of times each arc in the instance graph appears in one of the m best solutions found so far. Whenever a solution enters (leaves) this elite set, the frequencies of the arcs in this solution are incremented (decremented). In a removal step, for each request, the frequencies of the arcs over which its pickup node and its delivery node are reached and left in the current solution are summed up. The requests with the lowest sums are removed, with a certain amount of randomness, as proposed by Ropke and Pisinger (2006). A *zero split removal* heuristic, proposed by Parragh, Doerner, and Hartl (2010), removes sequences of requests between two arcs where the vehicle is empty. Longer sequences are preferred. The removed requests are reinserted one by one. An *entire route removal* heuristic, as its name implies, removes one or more complete routes from a solution. This is done in a random fashion. Entire route removal is a special case of zero split removal, as a vehicle is always empty when leaving the depot and when returning to it. The worst and Shaw removal heuristics are implemented in a static and a dynamic version. In the static versions, the removal criteria are not updated after each removal of a single request, and in the dynamic versions, they are. (The removal criterion for worst removal is the difference in the cost of the current solution with and without a certain request. Shaw removal uses a relatedness measure that takes into account, among other things, the points in time when the pickup and delivery nodes of a request are visited.) Our implementation also uses an extended entire route removal heuristic that, given a number q of requests to be removed, removes $q/2$ requests by a randomly selected removal heuristic other than entire route removal, and removes at least $q/2$ further requests by means of entire route removal.

After each remove–reinsert iteration, if the obtained solution is within 5% of the best solution found so far, each route of the new solution undergoes an optimization with the Balas and Simonetti (2001) neighborhood. Given a route $R = (v_1, \dots, v_r)$ and an integer $k \geq 2$, this neighborhood consists of all routes in which, for all $i, j \in \{1, \dots, r\}$, node v_i precedes node v_j whenever $j \geq i + k$. (Note that for $k = 1$, the precedence relation is fulfilled for all i and j only by the original route; hence, $k = 1$ is not a sensible parameter setting.) The authors propose a dynamic programming algorithm that solves the problem of finding the best neighbor in time linear in r but exponential in k. To this end, a layered network is constructed with one layer of vertices for each position in the route, so that there is a one-to-one correspondence between source-sink paths in this network and routes that satisfy the required precedence constraints. The Balas–Simonetti neighborhood was initially developed for the traveling salesman problem (TSP) and the TSP with time windows (TSPTW). However, with very minor modifications that do not affect the complexity of the procedure, it can be applied to problems with other precedence constraints as well as capacity restrictions. We use the dynamic programming labeling algorithm of Gschwind and Irnich (2015) to solve the resulting shortest path problems.

At the end of the ALNS, we solve a set covering problem. The column pool used consists of those routes that were optimized with the Balas–Simonetti search or, when Balas–Simonetti search was not applied, all routes generated during the search process. As the number of routes in the pool can be quite large, we do not necessarily solve the set covering problem to optimality, but limit the running time to two minutes.

5. Feasibility of Request Insertions

The repair steps of the ALNS need to evaluate the insertion of single requests into given feasible routes. A crucial aspect of such an evaluation is testing whether or not an insertion is feasible, that is, if the resulting route is feasible. In principle, this can be done using any of the route feasibility algorithms mentioned in Section 2. However, during the ALNS, millions of insertions must be checked for feasibility, so the check should be as efficient as possible and preferably have constant time complexity. Typically, this requires a specialized procedure for the evaluation of a request insertion into a given feasible route. Such $\mathcal{O}(1)$ verification procedures for the insertion of single customers or customer requests are known for several classes of routing problems (cf. Irnich 2008, Vidal et al. 2014). Usually, these procedures require the use of some auxiliary data that have to be recomputed each time one of the routes is modified. Since the preprocessing steps are called much less frequently than the feasibility tests, an improvement of the efficiency of the feasibility test typically overcompensates an increased effort for the computation of auxiliary data.

As mentioned, a feasible DARP route must satisfy constraints on pairing and precedence, vehicle capacity, time windows, maximum route duration, and maximum ride times. Ensuring pairing and precedence of a request insertion is straightforward. The verification whether an insertion is compatible with the capacity constraint is independent of the temporal constraints and identical for the DARP and the pickup and delivery problem with time windows (PDPTW). We omit a description here and refer the reader to Vidal et al. (2014).

Checking whether or not a route $R = (v_1, \dots, v_r)$ with $v_1 = 0$ and $v_r = 2n + 1$ respects the temporal constraints of the DARP is a nontrivial problem. The main difficulty is to deal with the trade-off between serving all nodes early (promoting feasibility regarding time windows) and serving pickup nodes and the origin depot late (promoting feasibility regarding maximum route duration and ride times). Formally, one has to find a schedule $T = (\tau_1, \dots, \tau_r)$ of service times, that is, points in time τ_i when the service at v_i starts, for all $i = 1, \dots, r$, satisfying

$$a_{v_i} \leq \tau_i \leq b_{v_i} \quad \forall\, v_i \in R, \tag{1}$$

$$\tau_i + t_{i,i+1} \leq \tau_{i+1} \quad \forall\, v_i \in R, i \neq r, \tag{2}$$

$$\tau_i + L_{v_i} \geq \tau_{i^-} \quad \forall\, v_i \in R \cap P, \tag{3}$$

$$\tau_1 + L \geq \tau_r. \tag{4}$$

The notation i^- (i^+) is used to refer to the corresponding delivery (pickup) node of a request or node i. Constraints (1), (3), and (4) are constraints on time windows, maximum ride times, and maximum route duration, respectively. Consistency of the service times along the route is ensured by constraints (2). The system (1)–(4) is called the *scheduling problem* of the DARP. The set of feasible time schedules for route R is denoted by $\mathcal{T}_R$. Note that the maximum route duration L can be seen as the maximum ride time L_0 of a dummy request 0 starting and ending at the origin and destination depots, and thus be treated in the same fashion as the other ride time constraints. We use this representation for the remainder of this paper and denote by $P_0 = P \cup \{0\}$ and $D_0 = D \cup \{2n+1\}$ the sets of original requests/pickups and deliveries together with the respective nodes of the dummy request.

5.1. Constant-Time Feasibility Testing for the PDPTW

Without the maximum ride times (3) for the requests $i \in P_0$, the resulting scheduling problem (1)–(2) is straightforward to solve. It is the scheduling problem of many variants of routing problems with time windows, such as the PDPTW. Traversing a route once from origin to destination depot and serving each node as early as possible is an optimal strategy for this problem. If this does not result in a feasible schedule, the route is infeasible. This is the key property that is exploited in the constant-time feasibility test for request insertions in the PDPTW. Another main tool that is used is the so-called *forward time slack* (FTS) originally defined by Savelsbergh (1992) in the context of the TSPTW. Given a feasible schedule T for route R, the FTS F_i at a node v_i gives the maximum amount of time by which the service time τ_i at v_i can be delayed so that the resulting schedule is also feasible. For ease of notation, we omit the dependence of F_i on T as the base schedule is always clear from the context. For the scheduling problem (1)–(2), F_i is given by $F_i = \min_{i\le j\le r}\{w_{i,j} + (b_{v_j} - \tau_j)\}$, where $w_{i,j} = \sum_{i<p\le j} \tau_p - (\tau_{p-1} + t_{p-1,p})$ denotes the cumulative waiting time between nodes v_i and v_j. Note that F_i depends on the service times τ_j with $j \ge i$, but not on those τ_j with $j < i$.

Algorithm 1 (Evaluate feasibility of request insertion in PDPTW)

```
   Input: nodes $v_k$, $v_l$, request $i$
   /* determine earliest possible
       service time at $i^+$                                */
 1 $\tau'_{i^+} = \max\{a_{i^+}, \tau_k + t_{v_k,i^+}\}$
 2 if $\tau'_{i^+} > b_{i^+}$ then return false
   /* evaluate time shift at $v_{k+1}$                    */
 3 $\delta_{k+1} = (\tau'_{i^+} + t_{i^+,v_{k+1}} - \tau_{k+1})^+$
 4 if $\delta_{k+1} > F_{k+1}$ then return false
   /* determine earliest possible service
       times at $v_l$ and $i^-$                             */
 5 $\tau'_l = \tau'_l + (\delta_{k+1} - w_{k+1,l})^+$
 6 $\tau'_{i^-} = \max\{a_{i^-}, \tau'_l + t_{v_l,i^-}\}$
 7 if $\tau'_{i^-} > b_{i^-}$ then return false
   /* evaluate time shift at $v_{l+1}$                    */
 8 $\delta_{l+1} = (\tau'_{i^-} + t_{i^-,v_{l+1}} - \tau_{l+1})^+$
 9 if $\delta_{l+1} > F_{l+1}$ then return false
10 return true
```

Algorithm 1 sketches the basic course of an $\mathcal{O}(1)$ feasibility test for the PDPTW. It evaluates the insertion of a pickup node i^+ after node v_k and the corresponding delivery node i^- after node v_l into a given route R. (If $k = l$, some very minor modifications are necessary; we omit these for brevity.) Based on an as-early-as-possible schedule $T = (\tau_1, \dots, \tau_r)$ for route R, an adapted schedule $T' = (\tau'_1, \dots, \tau'_r)$ for the new route $R' = (v_1, \dots, v_k, i^+, v_{k+1}, \dots, v_l, i^-, v_{l+1}, \dots, v_r)$ is implicitly constructed as follows: The pickup node i^+ is inserted with the earliest possible service time (Line 1). Its insertion generally increases the service time at node v_{k+1} compared to τ_{k+1} (Line 3). By means of the FTS F_{k+1}, it can be checked whether this increase δ_{k+1} still allows a feasible schedule for R' (Line 4). Using δ_{k+1} and $w_{k+1,l}$, the earliest possible service time τ'_l at v_l for the extended route R' can be computed (Line 5). The impact of inserting delivery node i^- is then evaluated in the same fashion as for i^+ (Lines 6–9). If the base schedule T, the FTS values of all nodes, and the cumulative waiting times between all pairs of nodes are known, Algorithm 1 clearly runs in constant time for any v_k, v_l, and i. The computation of these numbers requires a run-time quadratic in r, but can be done in a preprocessing step.

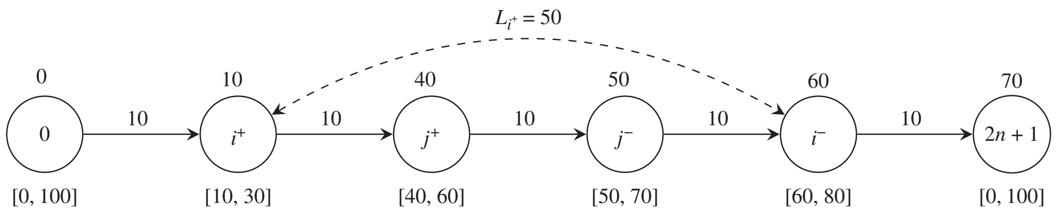
5.2. Constant-Time Feasibility Testing for the DARP

5.2.1. Basic Idea. Our $\mathcal{O}(1)$ feasibility test for request insertions in the DARP follows the basic principle of Algorithm 1: based on an as-early-as-feasible schedule T for the original route R, it tries to implicitly construct a feasible schedule T' for the extended route R'. The additional presence of maximum ride time constraints and the inherent trade-off between serving all nodes early and serving pickup nodes late, however, raises additional issues compared to the PDPTW case. First, serving all nodes as early as they can be reached is no longer an optimal strategy. Indeed, it may be beneficial to voluntarily wait and serve a pickup node i^+ later than at the earliest possible time, to prevent waiting at subsequent nodes and, hence, to decrease the ride time of the corresponding request i. It is therefore not sufficient to consider only as-early-as-possible service times for both the base schedule T and the insertion of the pickup node i^+. Second, when computing the maximum delay at a node v_i that still allows a feasible schedule, the ride times of all requests that are picked up before and delivered after v_i have to be taken into account. To cope with the latter, Cordeau and Laporte (2003) proposed the following adaptation of the FTS for the DARP:

$$F_i = \min_{i\le j\le r}\{w_{i,j} + \min(b_{v_j} - \tau_j, S_{ij})\}, \tag{5}$$

where S_{ij} is given by $L_{v_{j^+}} - (\tau_j - \tau_{j^+})$ if $v_j \in D_0, i > j^+$, and $+\infty$ otherwise. In contrast to the PDPTW, this definition of the FTS is not independent of the service times of nodes preceding v_i (cf. Gschwind 2015b). In fact, for a fixed service time τ_i, also for pickup nodes v_j with

Figure 1. Example of Nonunique Forward Time Slacks



$j < i < j^-$, service times other than the earliest possible ones must be considered, as different times may lead to different values S_{ij^-} and, thus, to potentially different values for the FTS. Obviously, later service times τ_j for such pickup nodes v_j may lead to larger values S_{ij^-} and a larger F_i. Hence, given a service time τ_i, serving all nodes v_j with $j < i$ as late as possible is an optimal strategy to maximize F_i. Evidently, for the evaluation of insertions into a route, the largest possible FTS values, that is, those resulting from an as-late-as-possible service at the preceding nodes, must be taken into account. When considering the insertion of additional nodes into route R, such an insertion before node v_i may restrict the set of feasible service times at $v_j, j < i$, and, thus, change the FTS values. Evidently, this effect also depends on the inserted node. Consider the example route $(0, i^+, j^+, i^-, j^-, 2n+1)$ given in Figure 1. Time windows, travel times between all nodes, and the maximum ride time of request i are depicted in the figure. The numbers above the nodes show an as-early-as-possible schedule for the route. Fixing the service times at nodes j^+, j^-, and i^- one at a time and determining the respective FTS values gives $F_{j^+} = F_{j^-} = F_{i^-} = 20$ (implying a service time of 30 at node i^+). When inserting between nodes i^+ and j^+ an additional node k with a nonbinding time window and travel times $t_{i^+,k} = t_{k,j^+} = 5$, the earliest possible service times and the FTS values for nodes j^+, j^-, and i^- remain unchanged. Assuming a time window of $[10, 15]$ for the inserted node k, the earliest possible service times for nodes j^+, j^-, and i^- are still unchanged. The FTS values of these nodes, however, are zero in the extended route. As a result, there is not a single FTS for each node that can be computed in a preprocessing step and that is valid for checking the feasibility of all request insertions at all positions.

The key idea to deal with these difficulties in a feasibility test is as follows. First, we sequentially try to insert the pickup and delivery nodes of request i after nodes v_k and v_l at the earliest times they can be reached (to facilitate time-window feasibility), checking whether or not the time shifts at the respective successor nodes v_{k+1} and v_{l+1} are feasible using FTS information F_{k+1} and F_{l+1}. Second, if this does not result in a feasible schedule, the algorithm tries to decrease the ride times of request i or requests v_j with $j < k < j^-$ or $j < l < j^-$ by serving the respective pickup nodes later. This possibly leads to later service times also at the succeeding nodes. Again, it must be checked whether or not the resulting time shifts at the nodes v_{k+1} and v_{l+1} are feasible. To test the feasibility of these time shifts, given the service times at v_{k+1} and v_{l+1}, the latest possible service times at all nodes preceding v_{k+1} and v_{l+1} are either computed explicitly or considered implicitly in the respective FTS information. To enable the latter, FTS information depending on the current latest possible service times at v_k and v_l is used.

5.2.2. Detailed Description of the Test.

To describe the feasibility test in more detail, it is convenient to introduce some additional notation and results that are needed for the presentation of the algorithm. For each pair of nodes $v_i, v_j \in R, i < j$, denote by $\Gamma_{ij} = \sum_{i \le p < j} t_{v_p, v_{p+1}}$ the cumulative travel time between nodes v_i and v_j along route R. Instead of the FTS F_i at a node $v_i \in R$, that is, the maximum amount of time by which the service time τ_i at a node v_i can be delayed such that a feasible schedule exists, we directly consider the latest feasible service time $\bar{\tau}_i = \max_{T \in \mathcal{T}_R} \tau_i$. By definition of FTS, $\tau_i + F_i = \bar{\tau}_i$, so that considering F_i or $\bar{\tau}_i$ is equivalent. Furthermore, we denote by $\bar{\tau}_i(t_k) = \max_{T \in \mathcal{T}_R(t_k)} \tau_i$ with $\mathcal{T}_R(t_k) = \{T \in \mathcal{T}_R : \tau_k \le t_k\}$ the latest feasible service time at node v_i such that node v_k is served no later than t_k. Similarly, let $\mathcal{T}_R(t_k, t_l) = \{T \in \mathcal{T}_R : \tau_k \le t_k, \tau_l \le t_l\}$ and $\bar{\tau}_i(t_k, t_l) = \max_{T \in \mathcal{T}_R(t_k, t_l)} \tau_i$. The following properties hold for the functions $\bar{\tau}_i$, $\bar{\tau}_i(t_k)$, and $\bar{\tau}_i(t_k, t_l)$:

Proposition 1. *Let $R = (v_1, \ldots, v_r)$ be a feasible route, and $t_k^{\min}$, $t_k^{\max}$, $t_l^{\min}$, $t_l^{\max}$ be the smallest and largest t_k and t_l such that $\mathcal{T}_R(t_k) \ne \emptyset$ and $\mathcal{T}_R(t_l) \ne \emptyset$. Then,*

(i) $\bar{T} = (\bar{\tau}_1, \ldots, \bar{\tau}_r) \in \mathcal{T}_R$ and $\bar{T}(t_k) = (\bar{\tau}_1(t_k), \ldots, \bar{\tau}_r(t_k)) \in \mathcal{T}_R(t_k)$ for all $v_k \in R$, $t_k^{\min} \le t_k \le t_k^{\max}$;

(ii) $\bar{\tau}_i(t_k) = \min\{h_i^1, h_i^2 + t_k\}$ for all $v_i, v_k \in R$, $t_k^{\min} \le t_k \le t_k^{\max}$, for certain constants h_i^1 and h_i^2;

(iii) $\bar{\tau}_i(t_k, t_l) = \min\{\bar{\tau}_i(t_k), \bar{\tau}_i(t_l)\}$ for all $v_i, v_k, v_l \in R$, $t_k^{\min} \le t_k \le t_k^{\max}$, $t_l^{\min} \le t_l \le t_l^{\max}$.

Proof. (i) $\bar{T}$ is feasible if and only if it satisfies constraints (1)–(4). By definition of $\bar{\tau}_i$, there exists for each $v_i \in R$ a feasible schedule $T^i = (\tau_1^i, \ldots, \bar{\tau}_i, \ldots, \tau_r^i)$ with $\tau_j^i \le \bar{\tau}_j$ for all $j \ne i$. It follows immediately that $\bar{T}$ satisfies constraints (1). Using $\bar{\tau}_i + t_{v_i, v_{i+1}} \le \tau_{i+1}^i$ and $\tau_{i+1}^i \le \bar{\tau}_{i+1}$, it follows that $\bar{T}$ satisfies constraints (2).

Similarly, $\tau_i^{i^-} + L_{v_i} \geq \bar{\tau}_{i^-}$ and $\bar{\tau}_i \geq \tau_i^{i^-}$ imply that constraints (3) and, by the same argument, also constraint (4) hold for $\bar{T}$. In the same fashion, it can be shown that $\bar{T}(t_k) \in \mathcal{T}_R(t_k)$.

(ii) For $k = r$, the result is included in Lemma 2 of Gschwind (2015a). The generalization of that proof to the case $k \neq r$ is obvious.

(iii) From $\mathcal{T}_R(t_k, t_l) \subseteq \mathcal{T}_R(t_k)$ and $\mathcal{T}_R(t_k, t_l) \subseteq \mathcal{T}_R(t_l)$, it follows immediately that $\bar{\tau}_i(t_k, t_l) \leq \bar{\tau}_i(t_k)$ and $\bar{\tau}_i(t_k, t_l) \leq \bar{\tau}_i(t_l)$. Consequently, $\bar{\tau}_i(t_k, t_l) \leq \min\{\bar{\tau}_i(t_k), \bar{\tau}_i(t_l)\}$ for all $v_i, v_k, v_l \in R, t_k^{\min} \leq t_k \leq t_k^{\max}, t_l^{\min} \leq t_l \leq t_l^{\max}$. To show that also $\bar{\tau}_i(t_k, t_l) \geq \min\{\bar{\tau}_i(t_k), \bar{\tau}_i(t_l)\}$ holds, let $\tau_i^{\min}(t_k, t_l) = \min\{\bar{\tau}_i(t_k), \bar{\tau}_i(t_l)\}$ and consider the schedule $T^{\min}(t_k, t_l) = (\tau_1^{\min}(t_k, t_l), \ldots, \tau_r^{\min}(t_k, t_l))$. It is clear that $\tau_k^{\min}(t_k, t_l) \leq t_k$ and $\tau_l^{\min}(t_k, t_l) \leq t_l$ hold and that $T^{\min}(t_k, t_l)$ fulfills (1). Moreover, by definition of $\bar{\tau}_i(t_k)$ and $\bar{\tau}_i(t_l)$, there exist for each $v_i \in R$ feasible schedules $T^{i,k} = (\tau_1^{i,k}(t_k), \ldots, \bar{\tau}_i(t_k), \ldots, \tau_r^{i,k}(t_k))$ with $\tau_j^{i,k} \leq \bar{\tau}_j(t_k)$ and $T^{i,l} = (\tau_1^{i,l}(t_l), \ldots, \bar{\tau}_i(t_l), \ldots, \tau_r^{i,l}(t_l))$ with $\tau_j^{i,l} \leq \bar{\tau}_j(t_l)$ for all $j \neq i$. Combining $\bar{\tau}_i(t_k) + t_{v_i, v_{i+1}} \leq \tau_{i+1}^{i,k}(t_k)$ and $\tau_{i+1}^{i,k}(t_k) \leq \bar{\tau}_{i+1}(t_k)$ with $\bar{\tau}_i(t_l) + t_{v_i, v_{i+1}} \leq \tau_{i+1}^{i,l}(t_l)$ and $\tau_{i+1}^{i,l}(t_l) \leq \bar{\tau}_{i+1}(t_l)$, it follows that $T^{\min}(t_k, t_l)$ satisfies constraints (2). Similar arguments to those used in the proof of (i) can be applied to show that $T^{\min}(t_k, t_l)$ respects constraints (3) and (4), too. Consequently, $T^{\min}(t_k, t_l) \in \mathcal{T}_R(t_k, t_l)$, and by definition of $\bar{\tau}_i(t_k, t_l)$, we have that $\bar{\tau}_i(t_k, t_l) \geq \tau_i^{\min}(t_k, t_l) = \min\{\bar{\tau}_i(t_k), \bar{\tau}_i(t_l)\}$. This completes the proof. □

We split the feasibility test of inserting a request i into two parts. The first is the evaluation of the insertion of the pickup node i^+ after v_k. The second is the evaluation of the insertion of the delivery node i^- after v_l. It is clear that if i^+ cannot be feasibly inserted at a specific position, neither position for the insertion of the delivery node i^- results in a feasible insertion of the request. Moreover, the insertion of the pickup node i^+ provides some information that can be reused for the evaluation of the insertion of the delivery node i^- at any considered position.

5.2.3. Evaluation of Pickup Insertion. Algorithm 2 details the insertion of a pickup node i^+ after node v_k. It first tries to insert i^+ as early as it can be reached (Lines 1 and 2) and determines the resulting new earliest service time at the successor node v_{k+1} (Line 3). Fixing time τ'_{k+1}, the service times at nodes i^+ and v_k are increased one after the other by as much as possible (Lines 4 and 5), that is, their latest feasible starts of service are computed. Finally, the time shift at node $k+1$ is tested for feasibility using the bound $\bar{\tau}_{k+1}(\tau'_k)$ (Line 6). Note again that by means of $\bar{\tau}_{k+1}(\tau'_k)$, it is implicitly considered that all nodes preceding v_k are served as late as possible given service time τ'_k, which is the latest possible service time at v_k given service time τ'_{k+1} at v_{k+1}. Note further that if the test in Line 6 fails, there is no possibility to repair this defect, and the insertion of i^+ between v_k and v_{k+1} is infeasible: the value of the bound $\bar{\tau}_{k+1}(\tau'_k)$ can be increased only by increasing τ'_k. However, if there is some waiting time between v_k and v_{k+1}, then either the service at v_k or i^+ is already restricted (because of their maximization given τ'_{k+1} in Lines 4 and 5), so that delaying it is not feasible. If there is no waiting time between v_k and v_{k+1}, then an increase of τ'_k leads to the same increase of τ'_{k+1}, so that the condition in Line 6 is still violated.

Algorithm 2 (Evaluate insertion of pickup)

```
  Input: node $v_k$, request $i$
  /* determine earliest possible
       service time at $i^+$                                  */
1 $\tau'_{i^+} = \max\{a_{i^+}, \tau_k + t_{v_k, i^+}\}$
2 if $\tau'_{i^+} > b_{i^+}$ then return false
  /* determine earliest possible $\tau'_{k+1}$                */
3 $\tau'_{k+1} = \max\{\tau_{k+1}, \tau'_{i^+} + t_{i^+, v_{k+1}}\}$
  /* maximize $\tau'_k$ and $\tau'_{i^+}$ for earliest possible
       service time at $v_{k+1}$                               */
4 $\tau'_{i^+} = \min\{b_{i^+}, \tau'_{k+1} - t_{i^+, v_{k+1}}\}$
5 $\tau'_k = \min\{\bar{\tau}_k, \tau'_{i^+} - t_{v_k, i^+}\}$
  /* evaluate time shift at $v_{k+1}$                          */
6 if $\tau'_{k+1} > \bar{\tau}_{k+1}(\tau'_k)$ then return false
7 return true
  Output: service times $\tau'_k$, $\tau'_{i^+}$, $\tau'_{k+1}$
```

5.2.4. Evaluation of Delivery Insertion. Algorithm 3 describes the insertion of a delivery node i^- after node v_l. It uses the information about the earliest service times τ'_k, τ'_{i^+}, and τ'_{k+1} provided by the feasibility test for the insertion of the corresponding pickup node i^+ after node v_k in Algorithm 2. First, by means of τ'_{k+1} and $\Gamma_{k+1,l}$, the new earliest start of service τ'_l at v_l is computed (Line 1). Then, delivery node i^- is inserted as early as it can be reached if feasible (Lines 2 and 3), and the new earliest service time at node v_{l+1} is determined (Line 4). If there is waiting time before node i^-, the service time τ'_l at v_l has to be maximized to obtain its latest feasible start of service given time τ'_{i^-} (Line 5). Note that τ'_l is also restricted by the ride times of requests that are picked up prior to v_l, which is ensured by the bound $\bar{\tau}_l(\tau'_k)$. Note further that it is not necessary to increase the service time at i^-. This can be seen as follows. If there is no waiting time between nodes i^- and v_{l+1}, then τ'_{i^-} is already maximal given time τ'_{l+1}. In the case of positive waiting time between i^- and v_{l+1}, the service time at node v_{l+1} does not change compared to the service time τ_{l+1} in the base schedule for R. Such a delay of zero is obviously feasible, that is, $\bar{\tau}_{l+1}(\tau'_k, \tau'_l) \geq \tau_{l+1}$ for any feasible τ'_k, τ'_l.

Given times τ'_k and τ'_l, the time shift at node v_{l+1} is then evaluated (Lines 6 and 7). A delay at v_{l+1} that seems to be too large, however, does not yet imply that the insertion is infeasible. In fact, it may be possible to repair this defect. Recall that up to now the pickup node i^+ has been inserted such that the succeeding node v_{k+1} is served as early as it can be feasibly reached, thereby also restricting the possible service times at pickup nodes preceding v_{k+1}. Allowing a later service time at v_{k+1} may reduce the ride times of requests that are picked up before and delivered after v_{k+1}, which might in turn allow a feasible schedule with service at node v_{l+1} at time τ'_{l+1}. Formally, this means trying to increase the bound $\bar{\tau}_{l+1}(\tau'_k, \tau'_l)$ on the service time τ'_{l+1} by increasing τ'_k or τ'_l while keeping τ'_{l+1} constant. Note first that it is impossible to restore feasibility by delaying only τ'_l. This can be seen by distinguishing two cases. If there is waiting time between nodes v_l and i^-, then τ'_l is constrained by $\bar{\tau}_l(\tau'_k)$ and can be increased only if τ'_k is also increased. If there is no waiting time before node i^-, then delaying the service at node v_l leads to an increase of τ'_{i^-} and τ'_{l+1} by the same amount (note that $\Delta_{l+1} > 0$ implies zero waiting time between nodes i^- and v_{l+1}), and, hence, the augmented time shift at v_{l+1} is again infeasible. Therefore, the algorithm always needs to increase the service time at v_k to restore feasibility with respect to the time shift at v_{l+1}. Moreover, a repair can be successful only if there is enough waiting time w_{k+1,i^-} between nodes v_{k+1} and i^- on the route, as delaying the service time τ'_k at v_k by Δ_{l+1} increases the bound $\bar{\tau}_{l+1}(\tau'_k, \tau'_l)$ by at most Δ_{l+1}, and if w_{k+1,i^-} is smaller than Δ_{l+1}, it also increases the service time τ'_{l+1}, implying that a feasible schedule cannot exist.

The repair step is detailed in Lines 7–14. It first checks whether the repair can be successful, that is, whether the service times at v_k and i^+ can be feasibly delayed by Δ_{l+1} and whether there is enough waiting time (Line 8). Then, the service times at v_k, i^+, and v_{k+1} are increased (Lines 9–11), and the new time shift at v_{k+1} is evaluated (Line 12). Note that after Line 8, there cannot be any waiting time between nodes v_k, i^+, and v_{k+1}. This is because otherwise, it would not be possible to increase the service times at v_k or i^+, as they are at their maximal value given time τ'_{k+1} due to their maximization within the insertion of the pickup node i^+ (Lines 4 and 5 of Algorithm 2). Finally, the new maximum service time τ'_l given the increased value $\bar{\tau}_l(\tau'_k)$ is determined (Line 13), and it is checked whether the bound $\bar{\tau}_{l+1}(\tau'_k, \tau'_l)$ could be sufficiently improved to allow a feasible insertion (Line 14).

Algorithm 3 (Evaluate insertion of delivery)

```
   Input: nodes $v_k$, $v_l$, request $i$, service
          times $\tau'_k$, $\tau'_{i^+}$, $\tau'_{k+1}$
   /* determine earliest service
        times at $v_l$, $i^-$, and $v_{l+1}$                          */
 1 $\tau'_l = \max\{\tau_l, \tau'_{k+1} + \Gamma_{k+1,l}\}$
 2 $\tau'_{i^-} = \max\{a_{i^-}, \tau'_l + t_{l,i^-}\}$
 3 if $\tau'_{i^-} > b_{i^-}$ then return false
 4 $\tau'_{l+1} = \max\{\tau_{l+1}, \tau'_{i^-} + t_{i^-,v_{l+1}}\}$
   /* maximize $\tau'_l$ for earliest possible
        service time at $i^-$ given $\tau'_k$ at $k$             */
 5 $\tau'_l = \min\{\bar{\tau}_l(\tau'_k), \tau'_{i^-} - t_{v_l,i^-}\}$
   /* evaluate time shift at $l+1$ and try to
       increase $\bar{\tau}_{l+1}(\cdot)$ if necessary                 */
 6 $\Delta_{l+1} = \tau'_{l+1} - \bar{\tau}_{l+1}(\tau'_k, \tau'_l)$
 7 if $\Delta_{l+1} > 0$ then
 8     if $\tau'_k + \Delta_{l+1} > \bar{\tau}_k$ or $\tau'_{i^+} + \Delta_{l+1} > b_{i^+}$
         or $\Delta_{l+1} > w_{k+1,i^-}$ then return false
 9     $\tau'_k += \Delta_{l+1}$
10     $\tau'_{i^+} += \Delta_{l+1}$
11     $\tau'_{k+1} += \Delta_{l+1}$
       /* evaluate new time shift at $\tau'_{k+1}$             */
12     if $\tau'_{k+1} > \bar{\tau}_{k+1}(\tau'_k)$ then return false
       /* maximize $\tau'_l$ given new service
          time $\tau'_k$ at $k$                                  */
13     $\tau'_l = \min\{\bar{\tau}_l(\tau'_k), \tau'_{i^-} - t_{l,i^-}\}$
       /* evaluate time shift at $\tau'_{l+1}$                */
14     if $\tau'_{l+1} > \bar{\tau}_{l+1}(\tau'_k, \tau'_l)$ then return false
   /* evaluate ride time of request $i$ and try
      to decrease it if necessary                         */
15 $\Delta_i = \tau'_{i^-} - (\tau'_{i^+} + L_i)$
16 if $\Delta_i > 0$ then
17     if $\tau'_{i^+} + \Delta_i > b_{i^+}$ or $\Delta_i > w_{k+1,i^-}$ then return false
18     $\tau'_{i^+} += \Delta_i$
19     $\tau'_{k+1} += \Delta_i$
       /* maximize $\tau'_k$ given new service
          time $\tau'_{i^+}$ at $i^+$                             */
20     $\tau'_k = \min\{\bar{\tau}_k, \tau'_{i^+} - t_{k,i^+}\}$
       /* evaluate new time shift at $\tau'_{k+1}$             */
21     if $\tau'_{k+1} > \bar{\tau}_{k+1}(\tau'_k)$ then return false
22 return true
```

So far, Algorithm 3 has ensured that the current implicitly built schedule T' for the extended route R' respects the time windows (1) of all nodes $V_j \in R'$, the consistency constraints (2) of the service times, and the maximum ride times of requests $v_j \in P_0 \cup R$. It remains to check whether the insertion is feasible also with respect to the maximum ride time constraint of the inserted request i. If the current service times τ'_{i^+} and τ'_{i^-} violate the maximum ride time by $\Delta_i > 0$, a repair procedure (Lines 16–21) similar to the one for the time shift at v_{l+1} as described above tries to decrease the ride time of i by serving i^+ later (i^- is already served as early as possible). Prerequisites are that the service at i^+ can be postponed by Δ_i units of time and that there is sufficient waiting time between nodes v_{k+1} and i^- (Line 17). The algorithm then increases the service times τ'_{i^-} and τ'_{k+1} accordingly (Lines 18 and 19). Recall that there cannot be any waiting time at node v_{k+1}. Finally, τ'_k is maximized given the new service time at i^- (Line 20), and the new start of service τ'_{k+1} at node v_{k+1} has to be checked for feasibility using the bound $\bar{\tau}_{k+1}(\tau'_k)$.

Note that there are two special cases for the insertion positions that require slightly different handling compared to the general procedure. The first is the insertion of pickup node i^+ and delivery node i^- in direct succession, that is, $k = l$. The second is the insertion of i^+ and i^- with only one node between them, that is, $k + 1 = l$. In the former case, the feasibility test is much simpler, and the entire insertion can be checked similarly to the insertion of the pickup node in the general case detailed in Algorithm 2. The only additional step is between Lines 2 and 3 where the insertion of the delivery node i^- is performed, computing its service time as $\tau'_{i^-} = \max\{a_{i^-}, \tau'_{i^+} + t_{i^+,i^-}\}$. Note that in this case the maximum ride time constraint of request i is always respected if an appropriate time-window tightening in the preprocessing is performed. Then, in Lines 3–5, "i^+" must be replaced with "i^-." Moreover, as in the general case, it is not necessary to maximize the service times at i^- given time τ'_{k+1}. This follows from the argument given in the description of Algorithm 3. In the second special case ($k + 1 = l$), the insertion of the pickup node i^+ is performed as described in Algorithm 2. For the insertion of the delivery node i^-, the following slight modifications have to be made in Algorithm 3: Line 1 is obsolete, as the service time at node $v_{k+1} = v_l$ is already known from the insertion of the pickup node. In Line 5 of the algorithm, the service time τ'_l is increased. This may introduce some waiting time between nodes i^+ and v_{k+1}. Consequently, in Lines 8 and 17, the value w_{i^+,i^-} needs to be considered instead of w_{k+1,i^-}. (Recall that in the general case $k+1 > l$, it is ensured that there is no waiting time between nodes i^+ and $k+1$ at that point of the feasibility test.) Furthermore, Lines 11 and 19 change to $\tau'_{k+1} = \max\{\tau'_{k+1}, \tau'_{i^+} + t_{i^+,v_{k+1}}\}$.

5.2.5. Proof of the Constant-Time Property. We now show that the entire feasibility test runs in $\mathcal{O}(1)$. Note first that all computations of Algorithms 2 and 3, that is, service time updates and comparisons, can be performed in constant time provided that all values used within these computations can be retrieved in constant time. Evidently, all input data, that is, time-window borders $a_\cdot$ and $b_\cdot$, travel times $t_{\cdot,\cdot}$, and maximum ride times $L_\cdot$, are available in constant time. The service times $\tau_i, i \in R$, of the base schedule T and the cumulative travel times Γ_{ij} between all pairs of nodes $v_i, v_j \in R$, $i < j$, are computed in a preprocessing step. The latter also allow the constant-time computation of the waiting times between all pairs of nodes $v_i, v_j \in R', i < j$, in the extended schedule using the formula $w_{ij} = \tau'_j - \tau'_i - \Gamma_{ij}$. Finally, the preprocessing ensures that the bounds $\bar{\tau}_i(t_k)$ and $\bar{\tau}_i(t_k, t_l)$ can be retrieved in constant time for any $v_i, v_k, v_l \in R$ and any feasible t_k and t_l. As a result, checking the feasibility of request insertions for the DARP can be done in $\mathcal{O}(1)$ time.

The computations and the computational effort of the preprocessing step are as follows. The determination of the cumulative travel times Γ_{ij} between all pairs of nodes v_i, $v_j \in R$, $i < j$, in $\mathcal{O}(r^2)$ is straightforward. A feasible as-early-as-possible base schedule $T = (\tau_1, \ldots, \tau_r)$ for route R can be computed using the algorithm of Tang et al. (2010), which requires $\mathcal{O}(r^2)$ computation time. Because of Proposition 1(ii), it is sufficient to store the two values $\bar{\tau}_i(\tau_k)$ and $\bar{\tau}_i(\bar{\tau}_k)$ to provide in constant time the values of the bounds/functions $\bar{\tau}_i(t_k)$ for any $v_i, v_k \in R, t_k \geq \tau_k$, via the expression $\bar{\tau}_i(t_k) = \min\{\bar{\tau}_i(\tau_k) + (t_k - \tau_k), \bar{\tau}_i(\bar{\tau}_k)\}$. Proposition 1(iii) guarantees that also the values $\bar{\tau}_i(t_k, t_l)$ with $v_i, v_k, v_l \in R$, $t_k \geq \tau_k, t_l \geq \tau_l$, are then available in constant time. For the computation of $\bar{\tau}_i(\tau_k)$ and $\bar{\tau}_i(\bar{\tau}_k)$, note first that $\bar{\tau}_i = \bar{\tau}_i(\bar{\tau}_k)$ holds for all $v_i, v_k \in R$. Moreover, the schedule $\bar{T} = (\bar{\tau}_1, \ldots, \bar{\tau}_r)$, where all nodes are served at their latest feasible time, is feasible (Proposition 1(i)). This schedule can be computed with quadratic time complexity by using a modified version of the algorithm of Tang et al. (2010) in which all passes over the route are traversed in the opposite direction, so that the algorithm provides an as-late-as-possible instead of an as-early-as-possible schedule. The values $\bar{\tau}_i(\tau_k)$ are determined in an analogous fashion. When fixing the service time at node v_k to τ_k, the schedule $\bar{T}(\tau_k) = (\bar{\tau}_1(\tau_k), \ldots, \bar{\tau}_r(\tau_k))$ is feasible according to Proposition 1(i). Again, the modified algorithm of Tang et al. (2010) can be used to compute the values $\bar{\tau}_i(\tau_k), v_i \in R$, in quadratic time. As this has to be done for each node $v_k \in R$, the entire preprocessing requires $\mathcal{O}(r^3)$ computation time. Theorem 1 summarizes the main result of this section.

Theorem 1. *For the DARP, the feasibility of an insertion of a request i into a given feasible route $R = (v_1, \ldots, v_r)$ can be checked in constant time. The computational effort for preparing the necessary auxiliary data in a preprocessing step is $\mathcal{O}(r^3)$.*

6. Computational Results

The algorithm described in this paper was implemented in C++ and compiled with Visual Studio 2013 in Windows 7 (64 bit). The parameters used in the ALNS are detailed in the online appendix. IBM ILOG CPLEX 12.6.2 was used for solving the set covering problems. Here, we changed only two default settings: CPLEX was run in single-thread mode, and the MIP emphasis switch was set to emphasize feasibility over optimality. All experiments were performed on a computer with an Intel Core i7-5930K CPU clocked at 3.50 GHz and with 64 GB of main memory.

6.1. Instances

There are two standard sets of DARP benchmark instances commonly used in the literature. The first one was proposed by Cordeau and Laporte (2003) and is referred to as the pr set. It comprises 20 instances with up to 144 requests and 13 vehicles. The second set encompasses 42 instances in two groups (a and b) with at most 96 requests and eight vehicles and was created by Ropke, Cordeau, and Laporte (2007), enlarging the set introduced by Cordeau (2006). The latter instances, because of their limited size and their tight time windows and ride time constraints, can all be solved to optimality in a short time: with a branch-and-price-and-cut algorithm by Gschwind and Irnich (2015), the average computation time is below 30 seconds on a desktop computer. This is faster than any of the heuristics described in the literature (including ours), so these instances are of limited interest for the evaluation of heuristic procedures. For the former instances, which are less tightly constrained, no optimal solutions are known.

6.2. Analysis of Results

In this section, we first discuss how the time consumption of our constant-time feasibility test compares empirically to that of the classical eight-step procedure by Cordeau and Laporte (2003) used by most approaches from the literature. Then, we analyze the impact of different setups of our ALNS algorithm on solution quality and running time. Finally, we compare the ALNS to state-of-the-art DARP heuristics from the literature.

6.2.1. Empirical Evaluation of the Constant-Time Feasibility Test. To evaluate our procedure empirically, we performed five runs over all benchmark instances with the pure ALNS (25,000 iterations, no Balas–Simonetti search, no set covering) using (i) the constant-time test, (ii) the classical eight-step procedure by Cordeau and Laporte (2003) (which has quadratic worst-case complexity), and (iii) an improved version of the latter test. In this improved version, a constant-time PDPTW check (cf. again Vidal et al. 2014) that disregards the ride time constraints is performed first. If this check returns "infeasible," the route is obviously infeasible also for the DARP; otherwise, the Cordeau and Laporte (2003) procedure is called. Compared to the standard test, our algorithm showed speedup factors ranging between 5 and 72, with an average of 32. Relative to the improved Cordeau–Laporte test, the speedup factors were still between 1.5 and 12, with an average of 3.8. In general, the speedup factors increased with increasing instance size. Hence, the empirical evaluation confirmed the theoretical superiority of our approach.

6.2.2. Impact of Different Configurations of the Presented ALNS. We experimented with the following parameters: number of ALNS iterations (25, 50, and 75,000), Balas–Simonetti local search ($k \in \{0, 2, \ldots, 6\}$, where 0 means that no Balas–Simonetti search is performed, and $k = 1$ is excluded for the reasons explained in Section 4), and solution of set covering problem at the end (yes/no; time limit of 120 seconds). We chose a full factorial design for the experiments and evaluated all benchmark instances for all 36 combinations of these parameters. The effects of the different configurations on the pr instances are visualized in Figure 2. Detailed tables can be found in the online appendix.

As can be seen from Figure 2 (left), computation times increase linearly with increasing number of ALNS iterations and exponentially with growing Balas–Simonetti k value. Solving the set covering problem had almost no effect, as the time limit was not reached in most cases. In Figure 2 (right), it can be seen that increasing the Balas–Simonetti parameter tends to reduce the average gap. Moreover, irrespective of the number of iterations and the Balas–Simonetti parameter, solving a set covering problem at the end pays off and always reduces the average gaps considerably. It can also be seen, however, that in several cases the gap increases with increasing k value. We attribute this to the lower diversification potential of larger k values: the larger k, the more often single routes are optimal

Figure 2. (Color online) Computation Times (Left) and Average Relative Gaps to BKS (Right) for Different Algorithm Configurations on pr Instances

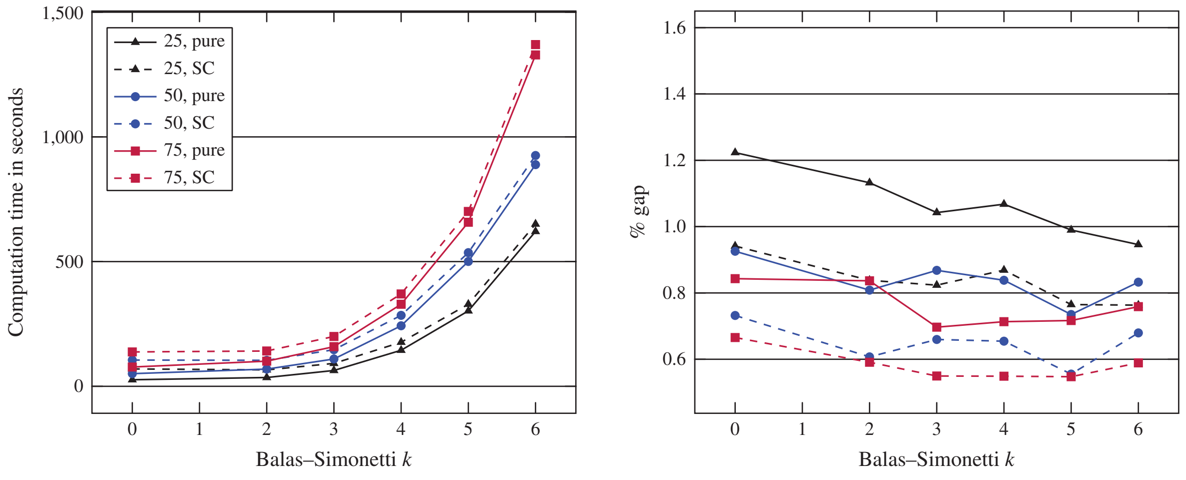


Table 1. Aggregated Results for pr Instances

		Braekers, Caris, and Janssens (TA)			Chassaing, Duhamel, and Lacomme (ELS)			`(50, no BS, no SC)`			`(75, k=3, SC)`		
Instance	BKS	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time	Best	Avg	Time
pr01	190.02	**190.02**	190.02	16.6	**190.02**	190.02	15.0	**190.02**	190.02	8.2	**190.02**	190.02	14.5
pr02	301.34	**301.34**	301.34	42.0	**301.34**	301.34	75.0	**301.34**	301.34	16.7	**301.34**	301.34	38.4
pr03	532.00	532.10	533.54	48.8	532.43	533.86	138.0	**532.00**	532.37	28.8	**532.00**	532.00	55.2
pr04	570.25	577.16	580.52	74.6	570.54	574.47	442.2	571.41	574.34	48.1	570.29	570.86	142.2
pr05	**625.64**	629.80	632.06	89.2	630.82	637.59	724.2	632.44	635.26	69.8	629.14	632.67	296.2
pr06	**783.78**	797.78	800.68	107.0	792.80	796.10	1,315.2	792.97	795.25	95.3	788.86	790.25	370.0
pr07	291.71	292.23	292.23	22.6	**291.71**	292.96	28.2	**291.71**	291.71	10.2	**291.71**	291.71	19.0
pr08	487.84	490.94	491.00	48.6	491.60	493.16	160.8	491.97	493.36	30.1	489.89	491.44	70.7
pr09	**653.94**	662.64	666.65	72.2	672.86	681.35	675.0	660.55	665.63	55.3	658.90	661.37	151.2
pr10	**845.47**	853.98	860.83	114.4	857.36	860.68	1,279.8	855.73	862.35	96.5	854.60	858.65	369.1
pr11	164.46	**164.46**	164.46	23.8	**164.46**	164.46	16.8	**164.46**	164.46	9.4	**164.46**	164.46	16.5
pr12	295.66	295.69	296.06	51.4	**295.66**	295.72	82.2	295.96	296.52	18.8	**295.66**	296.18	44.5
pr13	484.83	488.61	490.03	76.2	489.00	490.70	222.0	485.82	488.68	33.8	**484.83**	485.15	94.9
pr14	529.33	534.99	540.99	117.0	531.08	531.98	612.0	532.15	534.85	57.0	530.88	531.94	220.0
pr15	**573.56**	581.46	584.33	155.2	578.44	580.23	1,195.8	582.07	584.07	89.0	576.88	577.27	469.6
pr16	**725.22**	743.56	747.19	180.6	731.25	736.59	1,939.2	739.67	742.80	117.6	737.09	740.12	687.9
pr17	248.21	249.33	249.33	34.0	**248.21**	248.21	34.80	**248.21**	248.21	11.4	**248.21**	248.21	22.1
pr18	458.73	461.77	462.38	81.0	461.21	462.40	259.2	461.12	463.51	35.0	461.48	461.64	99.7
pr19	**592.23**	598.23	600.63	146.4	595.39	597.53	745.8	596.91	599.33	69.6	594.14	596.02	343.5
pr20	**783.81**	795.08	801.89	162.8	796.60	803.99	1,887.0	785.70	793.10	114.7	784.57	790.01	472.2
Avg		0.81%	1.16%	83.2	0.64%	1.04%	592.4	0.57%	0.93%	50.8	0.35%	0.55%	199.8

Notes. "BKS" is the best known solution. New BKSs found during our experiments are marked in bold. "Best" is the best solution over five runs. BKSs found or confirmed by an algorithm are marked in bold. "Avg" is the average solution over five runs. "Time" is the CPU time in seconds for one run (unscaled). The last row shows average gaps to BKSs for the Best and Avg columns, and average CPU times over five runs for Time columns.

for the requests they contain. Therefore, the input to the next ALNS iteration after a Balas–Simonetti local search is more likely to be the same as in a preceding iteration. Using an elbow criterion, the best tradeoff between computation time and solution quality is obtained for $k = 3$: computation times start to rise sharply for larger values, while the improvements in solution quality are minor.

6.2.3. Performance of the ALNS Compared to the State of the Art. Overall, the best-performing DARP heuristics we are aware of are the threshold accepting procedure of Braekers, Caris, and Janssens (2014) and the evolutionary local search by Chassaing, Duhamel, and Lacomme (2016). We therefore compare our ALNS with these two approaches. Because of the short running times all three approaches need for solving the benchmark instances, we do not attempt to scale the computation times reported in the different papers. Anyway, no CPU benchmark data are available for the system Braekers, Caris, and Janssens (2014) used.

Over five runs on the less interesting instance sets a and b, the Braekers, Caris, and Janssens (2014) algorithm finds 40 out of 42 optimal solutions, and the average gap over all runs is 0.01%. Chassaing, Duhamel, and Lacomme (2016) find 33 optimal solutions, and the average gap is 0.06%. Our heuristic has a comparable solution quality: with our best configuration `(75, k=3, SC)` (75,000 ALNS iterations with Balas–Simonetti $k = 3$ and set covering), we find 37 optimal solutions, and the average gap is 0.05%.

On the harder instance set pr, our heuristic performs even better. During all experiments performed while writing this paper, we discovered new best-known solutions for 8 of these 20 instances and confirmed the BKSs for the remaining 12. As can be seen in Table 1, already the pure ALNS with 50,000 iterations outperforms the state of the art with respect to solution quality, yielding better average gaps. The `(75, k=3, SC)` configuration further improves on these results, with average gaps almost halved and more BKSs confirmed (eight compared to three and six).

7. Conclusion

In this paper, we have described a constant-time procedure for checking the feasibility of a given route of the standard dial-a-ride problem, where a set of minimum-cost routes satisfying constraints on pairing and precedence, vehicle capacity, time windows, maximum route duration, and maximum user ride time is sought. Computational tests comparing the constant-time check with the classical eight-step procedure by Cordeau and Laporte (2003) used by most approaches from the literature empirically confirm the theoretical superiority of our approach, which provides an average speedup factor of 3.8. We have embedded the procedure into an ALNS metaheuristic and used two optional improvement techniques based on local search with the Balas and Simonetti (2001) neighborhood and on the solution of a set covering model containing a subset of the routes generated during the search process. The method is competitive with state-of-the-art DARP heuristics regarding solution quality and running time. On the classical benchmark set of Cordeau and Laporte (2003), it outperforms all previous solution approaches.

In future work, we intend to generalize our feasibility test to extensions of the DARP. In particular, the DARP with transfers as described by Masson, Lehuédé, and Péton (2014) and the synchronized pickup and delivery problem introduced by Gschwind (2015a) are of interest in this respect.

References

Balas E, Simonetti N (2001) Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS J. Comput.* 13(1):56–75.

Berbeglia G, Pesant G, Rousseau LM (2011) Checking the feasibility of dial-a-ride instances using constraint programming. *Transportation Sci.* 45(3):399–412.

Braekers K, Caris A, Janssens GK (2014) Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. *Transportation Res. Part B: Methodological* 67:166–186.

Chassaing M, Duhamel C, Lacomme P (2016) An ELS-based approach with dynamic probabilities management in local search for the dial-a-ride problem. *Engrg. Appl. Artificial Intelligence* 48:119–133.

Cordeau JF (2006) A branch-and-cut algorithm for the dial-a-ride problem. *Oper. Res.* 54(3):573–586.

Cordeau JF, Laporte G (2003) A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Res. Part B: Methodological* 37(6):579–594.

Cordeau JF, Laporte G (2007) The dial-a-ride problem: Models and algorithms. *Ann. Oper. Res.* 153(1):29–46.

Doerner KF, Salazar-González JJ (2014) Pickup-and-delivery problems for people transportation. Vigo D, Toth P, eds. *Vehicle Routing: Problems, Methods, and Applications* (Society for Industrial and Applied Mathematics, Philadelphia), 193–212.

Firat M, Woeginger GJ (2011) Analysis of the dial-a-ride problem of Hunsaker and Savelsbergh. *Oper. Res. Lett.* 39(1):32–35.

Gschwind T (2015a) A comparison of column-generation approaches to the synchronized pickup and delivery problem. *Eur. J. Oper. Res.* 247(1):60–71.

Gschwind T (2015b) Route feasibility testing and forward time slack for the synchronized pickup and delivery problem. Technical Report LM-2015-01, Gutenberg School of Management and Economics, Johannes Gutenberg University, Mainz, Germany.

Gschwind T, Irnich S (2015) Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. *Transportation Sci.* 49(2):335–354.

Häeme L, Hakula H (2015) A maximum cluster algorithm for checking the feasibility of dial-a-ride instances. *Transportation Sci.* 49(2):295–310.

Haugland D, Ho SC (2010) Feasibility testing for dial-a-ride problems. Chen B, ed. *Algorithmic Aspects in Information and Management. AAIM* 2010. Lecture Notes in Computer Science, Vol. 6124 (Springer, Berlin), 170–179.

Hunsaker B, Savelsbergh M (2002) Efficient feasibility testing for dial-a-ride problems. *Oper. Res. Lett.* 30(3):169–173.

Irnich S (2008) A unified modeling and solution framework for vehicle routing and local search-based metaheuristics. *INFORMS J. Comput.* 20(2):270–287.

Jain S, Van Hentenryck P (2011) Large neighborhood search for dial-a-ride problems. Lee J, ed. *Principles and Practice of Constraint Programming—CP* 2011. Lecture Notes in Computer Science, Vol. 6876 (Springer, Berlin), 400–413.

Kirchler D, Wolfler Calvo R (2013) A granular tabu search algorithm for the dial-a-ride problem. *Transportation Res. Part B: Methodological* 56:120–135.

Masson R, Lehuédé F, Péton O (2013) An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Sci.* 47(3):344–355.

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

Molenbruch Y, Braekers C, Caris A, Vanden Berghe G (2017) Multi-directional local search for a bi-objective dial-a-ride problem in patient transportation. *Comput. Oper. Res.* 77:58–71.

Parragh SN (2011) Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. *Transportation Res. Part C: Emerging Tech.* 19(5):912–930.

Parragh SN, Schmid V (2013) Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Comput. Oper. Res.* 40(1):490–497.

Parragh SN, Doerner KF, Hartl RF (2010) Variable neighborhood search for the dial-a-ride problem. *Comput. Oper. Res.* 77(6):58–71.

Parragh SN, Cordeau JF, Doerner KF, Hartl RF (2012) Models and algorithms for the heterogeneous dial-a-ride problem with driver-related constraints. *OR Spectrum* 34(3):593–633.

Parragh SN, Doerner KF, Hartl RF, Gandibleux X (2009) A heuristic two-phase solution approach for the multi-objective dial-a-ride problem. *Networks* 54(4):227–242.

Pisinger D, Ropke S (2010) Large neighborhood search. Gendreau M, Potvin JY, eds. *Handbook of Metaheuristics*, 2nd ed., International Series in Operations Research and Management Science, Vol. 146 (Springer, Boston), 399–419.

Qu Y, Bard JF (2015) A branch-and-price-and-cut algorithm for heterogeneous pickup and delivery problems with configurable vehicle capacity. *Transportation Sci.* 49(2):254–270.

Ritzinger U, Puchinger J, Hartl RF (2016) Dynamic programming based metaheuristics for the dial-a-ride problem. *Ann. Oper. Res.* 236(2):341–358.

Ropke S, Cordeau JF (2009) Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Sci.* 43(3):267–286.

Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Sci.* 40(4):455–472.

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.

Savelsbergh MWP (1992) The vehicle routing problem with time windows: Minimizing route duration. *ORSA J. Comput.* 4(2):146–154.

Schönberger J (2016) Scheduling constraints in dial-a-ride problems with transfers: A metaheuristic approach incorporating a cross-route scheduling procedure with postponement opportunities. *Public Transport* 9(1–2):243–272.

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, Glasgow, UK.

Tang J, Kong Y, Lau H, Ip AWH (2010) A note on "Efficient feasibility testing for dial-a-ride problems." *Oper. Res. Lett.* 38(5):405–407.

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.