This article was downloaded by: [130.64.11.153] On: 21 June 2017, At: 19:02
Publisher: Institute for Operations Research and the Management Sciences (INFORMS)
INFORMS is located in Maryland, USA

Transportation Science

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

Branch-and-Price-and-Cut for the Active-Passive Vehicle-Routing Problem

Christian Tilk, Nicola Bianchessi, Michael Drexl, Stefan Irnich, Frank Meisel

To cite this article:
Christian Tilk, Nicola Bianchessi, Michael Drexl, Stefan Irnich, Frank Meisel (2017) Branch-and-Price-and-Cut for the Active-Passive Vehicle-Routing Problem. Transportation Science

Published online in Articles in Advance 21 Jun 2017

. https://doi.org/10.1287/trsc.2016.0730

Full terms and conditions of use: http://pubsonline.informs.org/page/terms-and-conditions

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright © 2017, INFORMS

Please scroll down for article—it is on subsequent pages

INFORMS is the largest professional society in the world for professionals in the fields of operations research, management science, and analytics.
For more information on INFORMS, its publications, membership, or meetings visit http://www.informs.org

RIGHTSLINK

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

TRANSPORTATION SCIENCE
Articles in Advance, pp. 1–20
ISSN 0041-1655 (print), ISSN 1526-5447 (online)

Branch-and-Price-and-Cut for the Active-Passive Vehicle-Routing Problem

Christian Tilk,[a] Nicola Bianchessi,[a] Michael Drexl,[a] Stefan Irnich,[a] Frank Meisel[b]

[a] Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, 55128 Mainz, Germany; [b] Faculty of Business, Economics and Social Sciences, University Kiel, 24118 Kiel, Germany

Contact: tilk@uni-mainz.de (CT); nbianche@uni-mainz.de (NB); drexl@uni-mainz.de (MD); irnich@uni-mainz.de (SI); meisel@bwl.uni-kiel.de (FM)

Received: September 18, 2015
Revised: February 2016; June 2016
Accepted: August 31, 2016
Published Online in Articles in Advance: June 21, 2017

https://doi.org/10.1287/trsc.2016.0730

Copyright: © 2017 INFORMS

Abstract. This paper presents a branch-and-price-and-cut algorithm for the exact solution of the active-passive vehicle-routing problem (APVRP). The APVRP covers a range of logistics applications where pickup-and-delivery requests necessitate a joint operation of active vehicles (e.g., trucks) and passive vehicles (e.g., loading devices such as containers or swap bodies). The objective is to minimize a weighted sum of the total distance traveled, the total completion time of the routes, and the number of unserved requests. To this end, the problem supports a flexible coupling and decoupling of active and passive vehicles at customer locations. Accordingly, the operations of the vehicles have to be synchronized carefully in the planning. The contribution of the paper is twofold: First, we present an exact branch-and-price-and-cut algorithm for this class of routing problems with synchronization constraints. To our knowledge, this algorithm is the first such approach that considers explicitly the temporal interdependencies between active and passive vehicles. The algorithm is based on a nontrivial network representation that models the logical relationships between the different transport tasks necessary to fulfill a request as well as the synchronization of the movements of active and passive vehicles. Second, we contribute to the development of branch-and-price methods in general, in that we solve, for the first time, an ng-path relaxation of a pricing problem with linear vertex costs by means of a bidirectional labeling algorithm. Computational experiments show that the proposed algorithm delivers improved bounds and solutions for a number of APVRP benchmark instances. It is able to solve instances with up to 76 tasks, four active, and eight passive vehicles to optimality within two hours of CPU time.

Funding: This research was funded by the Deutsche Forschungsgemeinschaft under [Grants IR 122/5-2 and DR 963/2-1].
Supplemental Material: The online appendix is available at https://doi.org/10.1287/trsc.2016.0730.

Keywords: vehicle-routing • synchronization • branch-and-price • linear vertex costs

1. Introduction

Many applications in the area of transport logistics involve problems in which the execution of transport requests calls for a joint operation of different resources such as trucks, tractors, drivers, trailers, semitrailers, swap bodies, containers, or accompanying staff. A typical example is found in the transportation of containerized goods, where not just a manned truck but also an empty container is required for executing a request. Further examples are found in the healthcare sector or in the security industry, where the transportation of patients and valuable items must be accompanied by medics and security guards, respectively. In general, we can distinguish two classes of transport resources. The first class is constituted by means of transport that can move on their own from one location to another such as, for example, manned trucks. We refer to these resources as *active vehicles*. The second class consists of resources that cannot move autonomously but require an active vehicle for being repositioned. This class comprises trailers, semitrailers, all kinds of loading devices, equipment, and accompanying staff. For simplicity, we refer to all these resources jointly as *passive vehicles*.

In the literature on vehicle-routing problems (VRPs, see Irnich, Toth, and Vigo 2014 for an overview), the distinction of active and passive vehicles is usually ignored, and operations are planned for active vehicles only (cf. the recent survey by Lahyani, Khemakhem, and Semet 2015). This restricts the applicability of the developed models and algorithms to real-world problems in which either passive vehicles do not play a role or where active and passive vehicles are paired to fixed units. Although the latter eases the solution of the VRP, it may hinder an effective utilization of the resources. If, for example, a manned truck (active vehicle) and an

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

empty container (passive vehicle) are considered a unit in operations planning, the truck and its driver have to wait at a customer location while the container is being stuffed. Therefore, to support a more flexible use of such resources, Meisel and Kopfer (2014) introduced what we denote here as the active-passive vehicle-routing problem (APVRP), in which an explicit distinction between active and passive vehicles is made. This distinction enables the modeling of complex transport operations in which an active vehicle carries a passive vehicle (e.g., an empty container) to some pickup location, drops it off there, and leaves this location for performing transports of other passive vehicles elsewhere. Later, when the container has been stuffed, the same or some other active vehicle returns to the customer, picks up the container, and carries it to the delivery location. The problem introduces multiple interdependencies between vehicles and raises the need to synchronize the operations and the movements of active and passive vehicles in time and space. According to the survey by Drexl (2012), such a combination of synchronization requirements is rarely addressed in the VRP literature. This fact is in marked contrast to the aforementioned practical relevance of the APVRP. It therefore seems appropriate to devote further studies to this generic and complex problem.

Our paper addresses this research gap and provides a twofold contribution: First, we present an exact algorithm for the APVRP, which, to our knowledge, is the first branch-and-price-and-cut approach that considers explicitly the temporal interdependencies between active and passive vehicles and the resulting synchronization requirements. Because of these synchronization requirements, the adaptation of the branch-and-price-and-cut concept to the APVRP is not straightforward. Our algorithm is based on an extended set-partitioning formulation, which, in turn, uses a nontrivial network representation that models the logical relationships between the different transport tasks necessary to fulfill a request as well as the synchronization of the movements of active and passive vehicles. Second, we contribute to the development of branch-and-price methods for routing with synchronization in general: We provide solutions to the pricing subproblem, which is an elementary shortest-path problem with time windows and with linear vertex costs, by solving, for the first time, its ng-path relaxation (Baldacci, Mingozzi, and Roberti 2011) by means of a bidirectional labeling algorithm. We actually apply a refined ng-path relaxation, taking into account partial requests (henceforth called *tasks*) and precedences between these tasks instead of individual vertices or complete requests. The pricing problem structure and the use of the ng-path relaxation, moreover, require a sophisticated merge step in the labeling algorithm. Computational experiments show that the proposed algorithm delivers improved bounds and solutions for the APVRP benchmark suite of Meisel and Kopfer (2014), solving instances with up to 76 tasks, four active, and eight passive vehicles to optimality within two hours of CPU time.

The paper is organized as follows. Related literature is reviewed in Section 2. In Section 3, we formally describe the APVRP. A corresponding extended set-partitioning formulation is provided in Section 4. The branch-and-price-and-cut algorithm is presented in Section 5, and the method is computationally evaluated in Section 6. Finally, Section 7 summarizes the paper and discusses potential avenues for further research.

2. Literature

The manifold real-world logistics applications for vehicle-routing problems with synchronization requirements have motivated several studies. Most papers consider applications in which active vehicles have to be synchronized. One example is the operations planning for cross-docks where different trucks deliver less-than-truckload shipments that are then merged to full-truckload shipments before being sent out to customers (see, e.g., Buijs, Vis, and Carlo 2014; Morais, Mateus, and Noronha 2014). Another active research field is found in the management of home care operations. Here, service operations have to be synchronized if a person requires the help of two caregivers at the same time (see Bredström and Rönnqvist 2008; Mankowska, Meisel, and Bierwirth 2013; Labadie, Prins, and Yang 2014; Afifi, Dang, and Moukrim 2016). Further research addresses forestry applications where trucks have to be served by forest loaders (Hachemi, Gendreau, and Rousseau 2013), intermodal transportation via ships, trains, and trucks for the supply of automotive factories (Mues and Pickl 2005), scheduling of cooperating technician teams at customer locations (Dohn, Kolind, and Clausen 2009), and ship routing where cargoes from different origins have to be delivered to one and the same client simultaneously (Andersson, Duesund, and Fagerholt 2011). Eventually, there are also arc routing problems that include synchronization of active vehicles, e.g., when planning snow plowing operations (see Laporte 2016) and road marking (see Salazar-Aguilar, Langevin, and Laporte 2013).

The mentioned applications all involve active vehicles that move autonomously to those locations where synchronized services are required. By contrast, in the APVRP, an active vehicle and a passive vehicle must both traverse route segments synchronously to perform a service. Typical applications are found in routing problems where trucks pull trailers or swap bodies (see Smilowitz 2006, Cheung et al. 2008, Drexl 2013). Obviously, trailers and swap bodies are passive vehicles that are immobile without a truck. Usually, they

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

play the role of optional capacity extensions of trucks rather than being mandatory for the execution of transport operations. An exceptional case is when an individual customer has such a large demand that it must be served jointly by a truck with a swap body (see Huber and Geiger 2014). A further application of such a synchronization requirement is found in the drayage operations of container terminals, where empty and loaded containers are moved between customer locations and transshipment points. In this field of logistics, the containers constitute the passive vehicles that are mandatory for the transport of goods (see Cheung et al. 2008; Xue et al. 2014; Zhang, Yun, and Kopfer 2010, 2013; Zhang, Lu, and Wang 2014). Movement synchronization en route is also found in VRPs where driver crews can be assigned flexibly to trucks (see Hollis, Forbes, and Douglas 2006; Drexl et al. 2013). In such problems, the exchange of crews enables a better utilization of the trucks in compliance with work regulations for truck drivers. Further applications of a synchronization of vehicles and crews are considered in the papers by Kim, Koo, and Park (2010), where technicians have to be carried to customer locations, and by Kergosien et al. (2011, 2013), where ambulances carry patients and accompanying physicians from one care unit to another.

Drexl (2007, 2014) studies the VRP with trailers and transshipments (VRPTT), a problem that also requires the synchronization of operations and movements of active and passive vehicles. Two branch-and-cut algorithms are presented, but only very small instances can be solved. For a deeper investigation of VRPs with synchronization of operations and vehicle movements, Meisel and Kopfer (2014) provide mixed-integer programming (MIP) formulations, a branch-and-cut algorithm, an adaptive large neighborhood search (ALNS) metaheuristic, and benchmark instances for the APVRP. To our knowledge, the branch-and-price algorithm by Smilowitz (2006) is the only other exact method for the APVRP. To facilitate understanding, the differences between the algorithm of Smilowitz (2006) and ours as well as the extensions and improvements our approach provides will be discussed in Section 5.2.1, when the difficulties arising from the synchronization requirements will have been thoroughly explained.

A few other papers provide exact methods (typically based on column generation techniques) for VRPs with synchronization requirements, but merely for problems that involve active vehicles only (see Mues and Pickl 2005; Dohn, Kolind, and Clausen 2009; Dohn, Rasmussen, and Larsen 2011; Andersson, Duesund, and Fagerholt 2011). However, the features of the VRPTT and the APVRP, which include simultaneous operations planning of active and passive vehicles together with the possibility to couple and decouple them flexibly on their routes, are not supported by any of these approaches.

3. Problem Description and Modeling

The problem considered in this paper can be formally described as follows. We are given a set of pickup-and-delivery requests R, a set A of classes of active vehicles, and a set P of passive vehicles. For each class $a \in A$ of active vehicles, K_a denotes the number of vehicles in the class. Each request $r \in R$ consists of transporting a loading unit from a pickup location ℓ_r^+ to a delivery location ℓ_r^-. To fulfill a request r, an active vehicle must carry a passive vehicle to ℓ_r^+ for loading. Each passive vehicle can load only one request at a time, and each active vehicle can transport only one passive vehicle at a time. Hence, the loaded passive vehicle must then be transported directly to ℓ_r^- for unloading. Afterward, the empty passive vehicle must be carried away from ℓ_r^-. From the point of view of an active vehicle, the fulfillment of a request comprises three transport tasks:

(i) providing an empty passive vehicle at the pickup location of a request,

(ii) direct transport of a loaded passive vehicle from the pickup to the delivery location,

(iii) carrying away the emptied passive vehicle from the delivery location.

One, two, or three different active vehicles may perform these tasks for a request.

There are compatibility relationships between the requests and the active and passive vehicles. P^r denotes the set of passive vehicles that can be used to perform request r. Likewise, R^p denotes the set of requests that can be performed with passive vehicle p. P^a indicates the set of passive vehicles that can be carried by an active vehicle from class a, and A^p is the set of classes of active vehicles compatible with passive vehicle p.

All active vehicles are initially based at the same start depot o and end their routes at the same end depot d. Each passive vehicle p has its own start and end positions: It is initially located at o_p and must be brought to d_p at the end, whether or not it is used to fulfill a request. Hence, there are two tasks associated with each passive vehicle p, namely, to pickup and deliver the vehicle at o_p and d_p, respectively. (Aggregating identical active vehicles into classes while at the same time considering individual passive vehicles is convenient when setting up the network on which our problem formulation is based, ensures that each network arc is traversed by at most one active and/or at most one passive vehicle in any feasible solution, and makes branching easier.)

Let s_r^+ indicate the time necessary to load an empty passive vehicle with request r and s_r^- indicate the time to unload r. Times for coupling and uncoupling of passive vehicles to or from active vehicles are assumed to be

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

zero (but could easily be incorporated into the model). Picking up a loaded request r at its pickup location can be finished no earlier than at time e_r. Waiting is allowed. Unloading a request r at its delivery location must be finished no later than l_r. The overall planning horizon is $[0, t^{\max}]$. Requests that cannot be fulfilled imply a penalty.

The objective is to minimize a weighted sum of the total distance traveled, the total completion time of the routes, and the number of unserved requests. The respective weights are α, β, and $\gamma \in \mathbb{R}_+$.

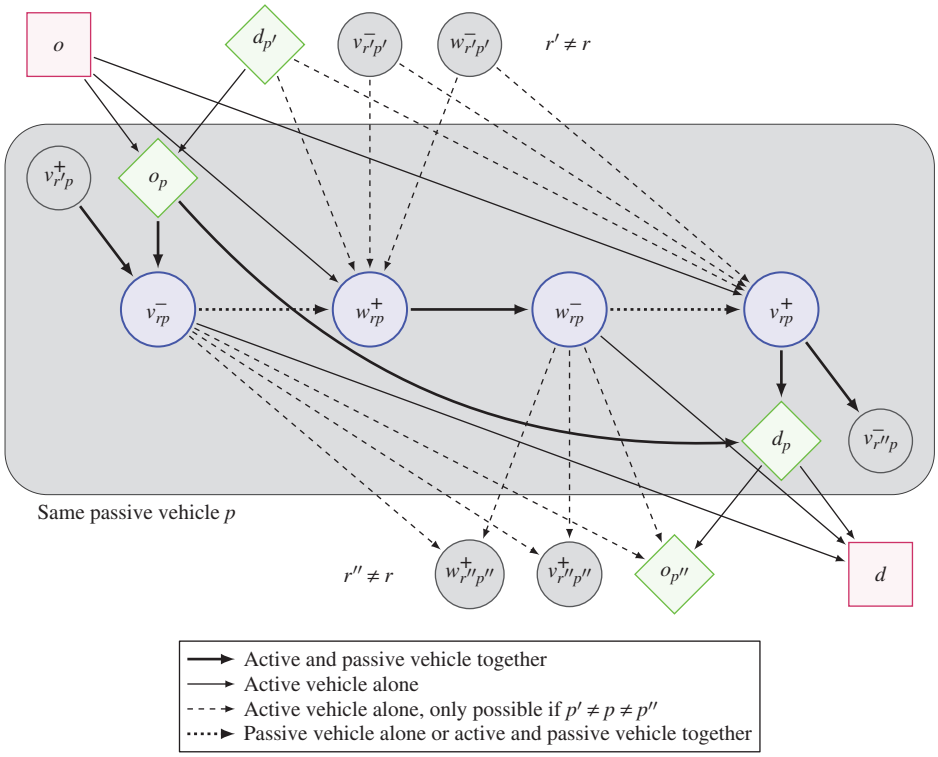
The APVRP as described above can be modeled as an optimization problem over a set of graphs $G^a = (V^a, E^a)$, one for each class a of active vehicles. Each set V^a of vertices contains o and d, i.e., the initial and final location of all class a active vehicles, o_p and d_p, i.e., the initial and final locations of all passive vehicles $p \in P^a$, and the set N^a containing the following four vertices for each passive vehicle $p \in P^a$ and each compatible request $r \in R^p$:

- v^-_{rp} delivery of an empty passive vehicle p into which request r is loaded;
- w^+_{rp} pickup of request r loaded in passive vehicle p;
- w^-_{rp} delivery of request r loaded in passive vehicle p;
- v^+_{rp} pickup of the empty passive vehicle p in which request r was transported.

Thus $N^a = \bigcup_{p \in P^a, r \in R^p} \{v^-_{rp}, w^+_{rp}, w^-_{rp}, v^+_{rp}\}$. For each request $r \in R$, we define $V^-_r = \{v^-_{rp}: p \in P^r\}$, $W^+_r = \{w^+_{rp}: p \in P^r\}$, $W^-_r = \{w^-_{rp}: p \in P^r\}$, and $V^+_r = \{v^+_{rp}: p \in P^r\}$. The vertices in $N^R = \bigcup_{r \in R} \left(V^-_r \cup W^+_r \cup W^-_r \cup V^+_r\right)$ are henceforth referred to as *request vertices*.

Arcs in E^a between each pair of vertex types are represented in Figure 1. In the figure, solid bold arcs (⟶) can be traversed by an active vehicle while traveling together with a passive vehicle attached. Dotted arcs (⋯➤) are service arcs and are traversed by the same passive vehicle p alone or by p together with an active vehicle, while solid and dashed arcs (⟶ and - - ➤) are traversed by the active vehicle alone. In particular, all dashed arcs (- - ➤) exist only if $p' \neq p \neq p''$. For the four request vertices in the shaded rectangle, all types of ingoing and outgoing arcs are depicted. It is easy to see that the graphs constructed in this way allow an active vehicle to perform any of the three transport tasks of any compatible request. For example, assume that active vehicle a_1 performs the first task of request r, and that active vehicle a_2 performs the second and third task of r, i.e., vehicle a_2 transports some passive vehicle p loaded with r from w^+_{rp} to w^-_{rp} and afterward moves away the empty passive vehicle. In this case, a_2 reaches w^+_{rp} without a passive vehicle, coming from either o or from a vertex $v^-_{r'p'}$, $w^-_{r'p'}$, or $d_{p'}$, with $r' \neq r$ and $p' \neq p$. (If $p' = p$, $d_{p'}$ is the final location of passive vehicle p, and an arc from $d_{p'}$ to w^+_{rp} for some $r \in R$ is impossible because of the uniqueness of the passive vehicles.) After that, a_2 visits vertices w^-_{rp} and v^+_{rp} and leaves v^+_{rp} heading toward either d_p or toward a vertex $v^-_{r''p}$ with $r \neq r' \neq r'' \neq r$.

Figure 1. (Color online) Arc Set of Graph $G^a = (V^a, E^a)$



RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

Table 1. Vertex Time Windows

Vertex i	Earliest time e_i	Latest time l_i
v^-_{rp}	$t_{oo_p} + t_{o_p v^-_{rp}}$	$l_{w^+_{rp}} - (s^+_r + t_{w^+_{rp}w^-_{rp}} + s^-_r)$
w^+_{rp}	$\max\{e_{v^-_{rp}} + s^+_r, e_r\}$	$l_{w^-_{rp}} - (t_{w^+_{rp}w^-_{rp}} + s^-_r)$
w^-_{rp}	$e_{w^+_{rp}} + t_{w^+_{rp}w^-_{rp}}$	$\min\{l_{v^+_{rp}}, l_r\} - s^-_r$
v^+_{rp}	$e_{w^-_{rp}} + s^-_r$	$t^{\max} - (t_{v^+_{rp}d_p} + t_{d_p d})$
Other	0	$t^{\max}$

The travel distances between any pair $(i, j) \in E^a$ of vertices are denoted by c_{ij} and are given by the distance between the associated locations. The same holds for the travel and service time t_{ij} for $(i, j) \in E^a$ with t_{ij} equal to s^+_r and s^-_r for arcs (v^-_{rp}, w^+_{rp}) and (w^-_{rp}, v^+_{rp}), respectively. Following the specifications given by Meisel and Kopfer (2014), the time windows at the vertices are given in Table 1. The complete notation used throughout the paper is summarized in the appendix.

The request vertices take up an idea described by Drexl (2007, Section 4.3.3). They essentially correspond to pairs of operations and passive vehicles. This ensures that the itineraries of the passive vehicles are implicit, i.e., need not be determined explicitly, but can be unequivocally reconstructed from given routes for the active vehicles. Hence, a feasible solution to the APVRP is a set of scheduled routes for the active vehicles that fulfills the following:

- Each route starts at vertex o and terminates at vertex d.
- All visited vertices are visited within their prescribed time windows.
- For each $r \in R$, at most one vertex per set is visited in sets V^-_r, W^+_r, and V^+_r. This ensures the synchronization of task fulfillment of requests within and between active vehicles. Note that it is not necessary to impose that at most one vertex is visited in set W^-_r, because vertices in this set can be reached only via an arc coming from a vertex in W^+_r.
- For each $r \in R$, $p \in P^r$, vertex w^+_{rp} is visited if and only if vertex v^-_{rp} has been visited, and vertex v^+_{rp} is visited if and only if vertex w^+_{rp} has been visited. This requirement is necessary to preserve consistency with respect to the passive vehicle used to satisfy a request in its different stages. Moreover, given that the final locations d_p of any passive vehicle $p \in P$ can only be reached from the corresponding initial location o_p or from vertices in $\bigcup_{r \in R} V^+_r$, this requirement preserves the flow of the unique passive vehicle used to satisfy a request through vertices v^-_{rp}, w^+_{rp}, w^-_{rp}, and v^+_{rp}.
- $T_{v^-_r} + s^+_r \le T_{w^+_r}$ and $T_{w^-_r} + s^-_r \le T_{v^+_r}$ for each $r \in R$, where $T_{v^-_r}$, $T_{w^+_r}$, $T_{w^-_r}$, and $T_{v^+_r}$ are the service start times at the vertices possibly selected in sets V^-_r, W^+_r, W^-_r, and V^+_r, respectively. This ensures temporal synchronization of tasks within and between active vehicles.
- Each passive vehicle is picked up at its initial location and placed at its final location.
- Each active vehicle performs at most one feasible route (the trivial route from o to d does not exist), so that at most K_a routes are performed for each class a of active vehicles.

The first two requirements are intraroute constraints, the subsequent four represent both intraroute and interroute constraints, and the last one is an interroute constraint.

Note that, along a route, an active vehicle of class $a \in A$ can be associated with different passive vehicles $p \in P^a$. This is due to the structure of the graphs $G^a = (V^a, E^a)$. Moreover, a passive vehicle $p \in P$ can be associated with different routes traveled by different active vehicles.

4. An Extended Set-Partitioning Formulation

To solve the APVRP with branch-and-price-and-cut, we use an extended set-partitioning formulation. This extensive formulation can be derived from a compact APVRP model, e.g., the one by Meisel and Kopfer (2014), using a Dantzig–Wolfe reformulation for integer programs (see Desaulniers et al. 1998, Lübbecke and Desrosiers 2005). For the sake of brevity, though, we omit the formal derivation.

For each class $a \in A$ of active vehicles, let Ω^a be the set of all feasibly scheduled routes. By contrast to many other column-generation algorithms for vehicle-routing described in the literature, a column in the APVRP does not only represent a path, i.e., the sequence of visited vertices. A column in the APVRP provides a path (in the graph $G^a = (V^a, E^a)$) and a feasible schedule for this path. For simplicity, however, we will refer to *routes* in the following, but use the terms *path* and *schedule* to describe the routing and the scheduling components. The following attributes characterize a route q:

X^q_{ij} number of times arc (i, j) is traversed by route q;
T^q_i service start time at a vertex $i \in N^R \cup \{d\}$;
b^q_i number of times vertex i is visited on route q;
c^q total costs of route q.

The route costs are defined as by Meisel and Kopfer (2014) as $c^q = \alpha \sum_{(i,j) \in E^a} c_{ij} X^q_{ij} + \beta T^q_d$, i.e., they are a weighted sum of the length of the route and the arrival time at the destination. Note that service start times are well defined because a feasible route is elementary.

The formulation uses the following types of variables: continuous variables λ^{aq} measuring the flow of active vehicles of class $a \in A$ along route $q \in \Omega^a$, binary variables x^a_{ij} indicating whether or not arc $(i, j) \in E^a$ is traversed by an active vehicle of class $a \in A$, and binary variables u_r indicating whether or not request $r \in R$ remains unfulfilled. Recall that the penalty for

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

not performing a request is γ. Now, the extended set-partitioning formulation for the APVRP is as follows:

$$\min \sum_{a\in A}\sum_{q\in\Omega^a} c^q\lambda^{aq}+\gamma\sum_{r\in R}u_r \tag{1a}$$

$$\text{s.t.} \sum_{a\in A}\sum_{q\in\Omega^a}\sum_{p\in P^r\cap P^a} b^q_{v^-_{rp}}\lambda^{aq}+u_r=1, \quad r\in R, \tag{1b}$$

$$\sum_{a\in A^p}\sum_{q\in\Omega^a}(b^q_{v^-_{rp}}-b^q_{w^+_{rp}})\lambda^{aq}=0, \quad r\in R, p\in P^r, \tag{1c}$$

$$\sum_{a\in A^p}\sum_{q\in\Omega^a}(b^q_{w^-_{rp}}-b^q_{v^+_{rp}})\lambda^{aq}=0, \quad r\in R, p\in P^r, \tag{1d}$$

$$\sum_{a\in A}\sum_{q\in\Omega^a}\sum_{p\in P^r\cap P^a}(T^q_{w^+_{rp}}-T^q_{v^-_{rp}})\lambda^{aq}+s^+_r u_r\geq s^+_r, \quad r\in R, \tag{1e}$$

$$\sum_{a\in A}\sum_{q\in\Omega^a}\sum_{p\in P^r\cap P^a}(T^q_{v^+_{rp}}-T^q_{w^-_{rp}})\lambda^{aq}+s^-_r u_r\geq s^-_r, \quad r\in R, \tag{1f}$$

$$\sum_{a\in A^p}\sum_{q\in\Omega^a} b^q_{o_p}\lambda^{aq}=1, \quad p\in P, \tag{1g}$$

$$\sum_{q\in\Omega^a}\lambda^{aq}\leq K_a, \quad a\in A, \tag{1h}$$

$$x^a_{ij}=\sum_{q\in\Omega^a} X^q_{ij}\lambda^{aq}, \quad a\in A,(i,j)\in E^a, \tag{1i}$$

$$\lambda^{aq}\geq 0, \quad a\in A, q\in\Omega^a, \tag{1j}$$

$$x^a_{ij}\in\{0,1\}, \quad a\in A,(i,j)\in E^a, \tag{1k}$$

$$u_r\in\{0,1\}, \quad r\in R. \tag{1l}$$

(1a) is the objective function. (1b) are the set-partitioning constraints, which ensure that each request is either performed exactly once or the penalty γ is paid for leaving the request unfulfilled. (1c) and (1d) preserve consistency with respect to the passive vehicle used to satisfy a request in its different stages. (1e) and (1f) are time synchronization constraints ensuring that the different vertices corresponding to a request are visited at correct points in time, taking into account their precedence relationships. (1g) and (1h) are quantity constraints for passive vehicles and for classes of active vehicles, respectively. Constraints (1g) are valid only if it is not allowed to temporarily park a passive vehicle at its initial or final location. (1i) link the route and the arc variables. (1j)–(1l) indicate the domains of the variables.

Note that there are no binary restrictions on the λ^{aq} variables. The reason for this is that a solution may be fractional in terms of λ^{aq} variables for a class of active vehicles. The λ^{aq} variables may correspond to the same path with different schedules, i.e., to the same sequence of vertices visited at different points in time, so that the arc variables are integral and no branching is necessary. This issue is discussed in detail by Desaulniers et al. (1998) and Jans (2010).

In the following, the linear relaxation of formulation (1) is denoted as the *master program*. It does not contain the coupling constraints (1i) and the integer constraints (1k)–(1l). For solving the master program, a column generation algorithm (Desaulniers, Desrosiers, and Solomon 2005) is employed. Branching is required to finally ensure integer solutions of formulation (1).

5. A Branch-and-Price-and-Cut Algorithm

In this section, we present the branch-price-and-cut algorithm we devised for solving the APVRP. First, in Section 5.1, we give an MIP model of the pricing problems for generating new columns. In Section 5.2, we describe a labeling algorithm for actually solving these pricing problems. Then, in Section 5.3, we describe our branching strategy. Finally, valid cutting planes for the master problem are described in Section 5.4.

5.1. Pricing Problems

As is common for column-generation algorithms for vehicle-routing problems, the pricing problems for the APVRP are shortest-path problems with resource constraints (SPPRCs) on graphs with negative cost cycles (Irnich and Desaulniers 2005). For each class $a \in A$ of active vehicles, there is one such pricing problem. Its goal is to find at least one route with negative reduced costs or to prove that no such route exists. The dual variables of those constraints in which the λ^{aq} variables occur in the restricted master program, the resulting linear vertex costs, and the ensuing reduced costs of the arcs can be read from Table 2.

We use binary variables x_{ij} indicating whether or not arc $(i,j) \in E^a$ is traversed and continuous T_i variables indicating the point in time when the service at vertex $i \in V^a$ begins. The symbols $\delta^+(i)$ and $\delta^-(i)$ denote the forward and backward star of vertex i, respectively. The pricing problem for a class a of active vehicles can be specified as follows:

$$\min \sum_{(i,j)\in E^a}\tilde{c}_{ij}x_{ij}+\sum_{i\in N^a}\tilde{c}_i T_i+\beta T_d \tag{2a}$$

$$\text{s.t.} \sum_{(o,j)\in\delta^+(o)} x_{oj}=1, \tag{2b}$$

$$\sum_{(j,i)\in\delta^-(i)} x_{ji}-\sum_{(i,j)\in\delta^+(i)} x_{ij}=0, \quad i\in V^a\backslash\{o,d\}, \tag{2c}$$

$$\sum_{(i,d)\in\delta^-(d)} x_{id}=1, \tag{2d}$$

$$T_i+t_{ij}\leq T_j+t^{\max}(1-x_{ij}), \quad (i,j)\in E^a, \tag{2e}$$

$$e_i\leq T_i\leq l_i, \quad i\in\{o,d\}, \tag{2f}$$

$$e_i\sum_{(i,j)\in\delta^+(i)} x_{ij}\leq T_i\leq l_i\sum_{(i,j)\in\delta^+(i)} x_{ij}, \quad i\in V^a\backslash\{o,d\}, \tag{2g}$$

$$x_{ij}\in\{0,1\}, \quad (i,j)\in E^a. \tag{2h}$$

(2a) is the objective of minimizing the reduced costs of the route. (2b)–(2d) are path-flow constraints and (2e)–(2g) ensure time-window feasibility. All arc variables are binary, as imposed by (2h).

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

Table 2. Dual Variables and their Ranges (a); Linear Vertex Costs (b); Reduced Costs of Arcs (c)

(a)			(b)		(c)	
Constraint	Dual variable	Range	Vertex i	Linear vertex costs $\tilde{c}_i$	Arc (i,j)	Reduced costs $\tilde{c}_{ij}$
(1b)	π_r	$\mathbb{R}$	v^-_{rp}	τ^+_r	(i,o_p)	$\alpha c_{io_p} - \mu^P_p$
(1c)	ϕ^+_{rp}	$\mathbb{R}$	w^+_{rp}	$-\tau^+_r$	(i,d_p)	αc_{id_p}
(1d)	ϕ^-_{rp}	$\mathbb{R}$	w^-_{rp}	τ^-_r	(i,v^-_{rp})	$\alpha c_{iv^-_{rp}} - \pi_r - \phi^+_{rp}$
(1e)	τ^+_r	≥ 0	v^+_{rp}	$-\tau^-_r$	(i,w^+_{rp})	$\alpha c_{iw^+_{rp}} + \phi^+_{rp}$
(1f)	τ^-_r	≥ 0	d	β	(i,w^-_{rp})	$\alpha c_{iw^-_{rp}} - \phi^-_{rp}$
(1g)	μ^P_p	$\mathbb{R}$	Otherwise	0	(i,v^+_{rp})	$\alpha c_{iv^+_{rp}} + \phi^-_{rp}$
(1h)	μ^A_a	≤ 0			(i,d)	$\alpha c_{id} - \mu^A_a$

5.2. Dynamic Programming Labeling Algorithms

The usual solution approach for SPPRC pricing problems is a dynamic-programming-based labeling algorithm. As has been shown by Salani (2005), solving such pricing problems by bidirectional dynamic programming, i.e., by propagating labels forward from the start depot vertex and backward from the end depot vertex, allows considerable speedups compared to a unidirectional procedure. We therefore adopt this approach and adapt it to the APVRP pricing problem.

In what follows, we first point out the specific characteristics resulting from the linear vertex costs in the APVRP pricing problem and briefly review similar problems and solution approaches encountered in the literature. We then discuss issues related to (non-)elementarity of SPPRC solutions and present our adaptation of the ng-path relaxation. Afterward, we describe the forward label extension step and the dominance procedure we apply. Then, we elaborate on the backward label extension and the method for merging forward and backward labels, and, finally, we present some techniques for accelerating the pricing process.

5.2.1. Labeling Algorithms for SPPRCs with Linear Vertex Costs. Because of the linear vertex costs $\tilde{c}_i$ in model (2), which are not present in pricing problems for standard VRPs, there is a trade-off in the APVRP pricing problem: In a path in which the linear vertex costs are nonnegative at all vertices, it is best to visit all vertices as early as possible. Similarly, in a path in which the linear vertex costs are nonpositive at all vertices, it is best to visit all vertices as late as possible. However, paths in the APVRP may visit vertices with positive as well as vertices with negative vertex costs. Hence, determining a cost-optimal schedule even for a *given* path is a nontrivial optimization problem in itself. For a labeling algorithm, this means that the linear vertex costs and the resulting trade-off between costs and time create an infinite number of possible states that do not dominate one another. Such a situation was first studied by Ioachim et al. (1998) on an acyclic time-space network in the context of aircraft fleet routing and scheduling. Their approach of storing cost functions (with time as the independent variable) as labels and propagating these functions instead of scalar values forms the basis of our method, which is described in detail in Section 5.2.3.

Other authors that consider shortest-path pricing problems with linear vertex costs in column-generation algorithms for routing problems are Christiansen and Nygreen (1998) (for a ship routing and scheduling problem with inventory constraints), and Dohn, Kolind, and Clausen (2009) and Dohn, Rasmussen, and Larsen (2011) (for VRPs with precedences and temporal dependencies between visits to customers), Liberatore, Righini, and Salani (2011) (for a VRP with soft time windows), and Spliet and Gabor (2015) (for a VRP with time windows that have to be assigned before customer demand is known). The pricing problem of Christiansen and Nygreen (1998) is solved with the Ioachim et al. (1998) algorithm. Dohn, Kolind, and Clausen (2009) and Dohn, Rasmussen, and Larsen (2011) describe two alternative approaches for dealing with the linear vertex costs: They consider a so-called time-indexed formulation with an implicit representation of precedence constraints and the possibility of branching on time windows to handle temporal dependencies. Liberatore, Righini, and Salani (2011) and Spliet and Gabor (2015) also base their procedures on the Ioachim et al. (1998) approach.

The main differences between the problems and solution procedures presented by Ioachim et al. (1998), Liberatore, Righini, and Salani (2011), Spliet and Gabor (2015), and ours are as follows: Ioachim et al. (1998) consider an acyclic network, all others have networks with cycles. Because of the soft time windows, the Liberatore, Righini, and Salani (2011) problem is, on one hand, simpler with respect to time windows, as the complete planning horizon is feasible. On the other hand, it is more difficult with respect to the cost (penalty) function *at each single vertex*: this penalty function is decreasing until the beginning of the desired time window, then constant until the end of the desired time window, and then increasing. By contrast, the cost

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

functions in the papers by Ioachim et al. (1998) and Spliet and Gabor (2015) as well as in our paper are either constant or only increasing or only decreasing at each vertex.

Both the Liberatore, Righini, and Salani (2011) and the Spliet and Gabor (2015) pricing problem feature a capacity constraint, which is not present in the Ioachim et al. (1998) problem or in the APVRP. Liberatore, Righini, and Salani (2011) handle the capacity constraint in the dominance procedure by allowing a label L to dominate another label L' only if the load of L is less than or equal to that of L'. Spliet and Gabor (2015) construct an auxiliary acyclic graph in which the capacity constraint is taken into account implicitly at the cost of a weaker dominance.

The branch-and-price algorithm by Smilowitz (2006) mentioned in Section 2, although considering an APVRP use case, does not take into account the time synchronization aspect in an exact manner, contrary to our approach. Instead, the interdependencies between tasks are modeled away by assuming that a passive vehicle that is used to perform a task τ is available for performing other tasks at the earliest after the end of τ's time window plus the service time for τ. In this way, the underlying network does not contain arcs between tasks with overlapping time windows, no linear vertex costs occur, and standard labeling approaches can be used for solving the pricing problem.

5.2.2. Elementarity and Precedences. In an optimal solution to the set-partitioning formulation (1), no task will be performed more than once. This implies that all columns in an optimal solution correspond to elementary paths in the network. It is well known that the elementary shortest-path problem with resource constraints is NP-hard in the strong sense (Dror 1994). Hence, in column-generation algorithms for VRPs, many authors solve as pricing problems nonelementary SPPRCs (see the survey by Desaulniers, Madsen, and Ropke 2014). This can be done in pseudo-polynomial time (Irnich and Desaulniers 2005). Although this yields weaker lower bounds, and though routes with cycles must be removed in the branching process, solving only a relaxed pricing problem often pays off with respect to overall computation time. One such approach that has been successfully used for different types of VRPs is the ng-path relaxation introduced by Baldacci, Mingozzi, and Roberti (2011).

We adapt this approach to the APVRP as follows. Our ng-path relaxation is based on tasks instead of vertices, thus leading to a stronger relaxation on the network we use. For each vertex $i \in N^a \cup \{o_p, d_p : p \in P^a\}$, we use an ng-neighborhood $\mathcal{N}_i$ containing the task associated to i (see Table 3) and the ν closest tasks associated to vertices j for which a cycle $(j, \ldots, i, \ldots, j)$ would be feasible with respect to time windows and travel and service times. (We use different values of the parameter ν in our computational experiments.) In an ng-path in the APVRP, it is possible that a task τ is performed more than once if, between two visits to a vertex i associated with the task, at least one other vertex j is visited such that $\tau \notin \mathcal{N}_j$.

In addition, to foster elementarity of partial paths without weakening dominance, we use the following approach: Consider the second line of Table 3. As described in Section 3, we associate three tasks with each request, and two tasks with each passive vehicle, for pickup and delivery at the initial and final location. Obviously, these tasks must be performed in the chronological sequences τ_r^1, τ_r^2, τ_r^3 and τ_p^o, τ_p^d (but not necessarily by the same active vehicle). Now, at each request vertex i, we consider two subsets of these tasks, $\mathcal{T}_i^{\text{test}}$ and $\mathcal{T}_i^{\text{set}}$, as indicated in the third and fourth line in the table. Before extending a label L at vertex i to vertex j, we test the tasks in the associated set $\mathcal{T}_j^{\text{test}}$. If any of these tasks has already been fulfilled along the partial path represented by L, then L is not extended to j, because such a path is not feasible for the APVRP. Similarly, upon extending a label L at vertex i to vertex j, we mark the tasks in $\mathcal{T}_j^{\text{set}}$ as fulfilled, because visiting them would lead to an infeasible path regarding task precedences. Note that it is unnecessary to include the tasks in the sets $\mathcal{T}^{\text{test}}$ and $\mathcal{T}^{\text{set}}$ in the ng-neighborhood. This is because these tasks are irrelevant for dominance, as dominance checks occur only between labels resident at the same vertex, and as all tasks to test and all tasks to set are the same for all labels at the same vertex. With respect to the ng-path relaxation, on one hand, the tasks to set allow making labels more comparable from the dominance point of view, and, on the other hand, the tasks to test prevent the construction of infeasible paths. This improvement can always be applied whenever a subset of tasks associated with some vertices of the graph must be executed according to a given precedence relation.

5.2.3. Forward Label Extension. As mentioned, our approach is based on the one by Ioachim et al. (1998), who store cost functions as labels instead of scalars. Such a cost function c provides the minimal costs $c(T)$ incurred by a path when the service at the last vertex of the path starts at time T. Ioachim et al. (1998) prove that this function is piecewise linear, convex, and contains at most as many linear pieces as there are vertices in the path. Moreover, they show that pieces with positive slope can be replaced by a single piece with slope zero. Hence, for the APVRP, a label comprises the following components:

n	number of time-slope pairs (in the following called pieces);
$(t^p, s^p)_{p=1}^n$	the n pieces; $s^p < 0$ for $p = 1, \ldots, n-1$, and $s^n \le 0$;
c^1	(reduced) costs at start time of piece 1;

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

Table 3. Chain of Precedences for each Pair $(p, r) \in P \times R^p$

Vertex i	o_p	v^-_{rp}	w^+_{rp}	w^-_{rp}	v^+_{rp}	d_p
Associated task	τ^o_p	τ^1_r	τ^2_r	τ^2_r	τ^3_r	τ^d_p
Tasks to test $\mathcal{T}^{\text{test}}_i$	τ^o_p, τ^d_p	$\tau^1_r, \tau^2_r, \tau^3_r, \tau^d_p$	$\tau^2_r, \tau^3_r, \tau^d_p$	τ^3_r, τ^d_p	τ^3_r, τ^d_p	τ^d_p
Tasks to set $\mathcal{T}^{\text{set}}_i$	τ^o_p	τ^o_p, τ^1_r	τ^o_p, τ^1_r	$\tau^o_p, \tau^1_r, \tau^2_r$	$\tau^o_p, \tau^1_r, \tau^2_r, \tau^3_r$	τ^o_p, τ^d_p

t^{n+1} end time of last piece n;
S tasks that have already been performed or are unreachable along the route and that are taken into account for checking elementarity in the label extension step according to the ng-path relaxation;
k number of intervals on which the label is dominated;
$(I^d)_{d=1}^k$ the k intervals on which the label is dominated.

The following information can then be derived:

$c^* =$ $c^1 + \sum_{p=1}^{n} s^p(t^{p+1} - t^p)$; the optimal (reduced) costs;
$t^* =$ t^n for $s^n = 0$, and t^{n+1} otherwise; the earliest time to obtain costs c^*;
$c(T) =$ $c^1 + \sum_{p=1}^{q-1} s^p(t^{p+1} - t^p) + s^q(T - t^q)$ for $t^q \le T \le t^{q+1}$; the trade-off curve;
$s^{n+1} =$ 0 defined so for convenience.

Figure 2 depicts two typical situations. In Figure 2(a), the slope of the function is strictly negative over its complete range, so that the minimal (reduced) costs are achieved when starting the service at the vertex as late as possible. In Figure 2(b), the slope of the function is zero on the positive-length interval from t^2 to t^3, and any time value in this interval yields the minimal (reduced) costs c^*. Note that the figures imply that the paths whose cost functions are depicted in Figures 2(a) and 2(b) contain at least three and two vertices, respectively.

The initial label at vertex o has a single piece $(t^1_o, s^1_o) = (e_o, 0)$, initial reduced costs $c^1_o = 0$, task set $S_o = \emptyset$, and it is undominated, i.e., $k_o = 0$. Note that by initializing the first piece as $(t^1_o, s^1_o) = (e_o, -\beta)$ would allow for changing the time-related part of the objective to route duration minimization; see also Tilk and Irnich (2017).

Now we describe the label extension step. (We present the extension step here because the paper by Ioachim et al. 1998 omits several details.) Let $L_i = ((t^p_i, s^p_i)_{p=1}^{n_i+1}, c^1_i, S_i, (I^d_i)_{d=1}^{k_i})$ be a label at vertex i. The extension of L_i along the arc (i, j) is feasible if $t^1_i + t_{ij} \le l_j$ and $S_i \cap \mathcal{T}^{\text{test}}_j = \emptyset$. In this case, a new label L_j at vertex j is created. The attributes of L_j are computed in the following way.

First, extending some of the existing pieces may be obsolete (cf. Ioachim et al. 1998, p. 200), either because the extended pieces arrive too early or too late at j or because several resulting new slopes are zero. We compute the indices f and g of the first and last new piece to be kept

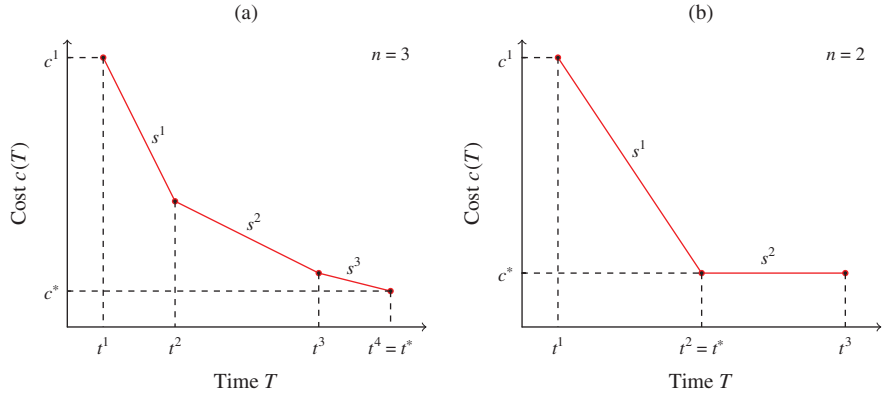
$$f := \max\{1;\ \max\{p \in \{1, \dots, n_i + 1\}: t^p_i + t_{ij} \le e_j\}\},$$

and

$$g := \min\{n_i;\ \min\{p \in \{0, \dots, n_i\}: s^p_i + \tilde{c}_j \ge 0 \text{ or } t^{p+1}_i + t_{ij} \ge l_j\}\}$$

(defining $s^0_i = -\infty$ for the case $p = 0$). It may also happen that a new piece must be created (cf. Ioachim et al. 1998, p. 200). There are three cases: (i) The last piece of L_i has a negative slope, will be extended to j, and when

Figure 2. (Color online) Examples of Labels: (a) A Label with $n = 3$ Linear Pieces and Final Slope $s^n < 0$; (b) A Label with $n = 2$ Linear Pieces and Final Slope $s^n = 0$



RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

starting from i at the optimal time t_i^* vertex j can be reached in its time window, (ii) j is reached before the start of its time window for all $T \in [t_i^1, l_i]$, and (iii) only a single point in time is propagated to j and this point is a breakpoint of $c_i(T)$. Thus, we define the new-piece indicator

$$\delta := \begin{cases} 1 & \text{if } (g = n_i, t_i^* = t_i^{n+1}, \text{ and } t_i^* + t_{ij} < l_j) \\ & \quad \text{or } (f = n_i + 1) \text{ or } (l_j = t_i^f + t_{ij}), \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

so that the new label L_j comprises the

$$n_j := \max\{0, g - f + 1\} + \delta \tag{4}$$

new pieces.

Second, nonobsolete pieces (t, s) are extended using the function $f_{ij}(t, s) := (\max\{e_j, t + t_{ij}\}, \min\{0, s + \tilde{c}_j\})$, so that the new pieces are

$$(t_j^p, s_j^p) := f_{ij}(t_i^{f+p-1}, s_i^{f+p-1}), \quad \text{for all } p = 1, \ldots, n_j, \tag{5a}$$

and the new end time of the pieces associated with slope 0 is

$$(t_j^{n_j+1}, s_j^{n_j+1}) := (l_j, 0). \tag{5b}$$

Third, with the help of the trade-off curve $c_i(T)$ of L_i and the already computed attributes, the costs at the start time t_j^1 of the new pieces can be expressed as

$$c_j^1 := c_i(\min\{t_i^*, t_j^1 - t_{ij}\}) + \tilde{c}_{ij} + \tilde{c}_j t_j^1. \tag{5c}$$

Note that the formula for the cost update given by Ioachim et al. (1998, p. 202) is not always correct.

Fourth, the remaining attributes are

$$S_j := (S_i \cap \mathcal{N}_j) \cup \mathcal{T}_j^{\text{set}} \quad \text{and} \quad k_j := 0. \tag{5d}$$

The latter definition means that at the time of its creation, the new label is undominated. This completes the description of the new label $L_j = ((t_j^p, s_j^p)_{p=1}^{n_j+1}, c_j^1, S_j, (I_j^d)_{d=1}^{k_j})$.

From a label L_d at the end depot d, the corresponding o-d-path is reconstructed, as usual, by iterating backward through the predecessor labels. Let $(d = i + 1, i, \ldots, 0 = o)$ be the backtracked path from a label L_d. The optimal schedule $(T_{i+1}, \ldots, T_0)$ is then given by the following recursion:

$$T_{i+1} = t_d^*, \tag{6a}$$

$$T_i = \min\{t_i^*, T_{i+1} - t_{i,i+1}\}. \tag{6b}$$

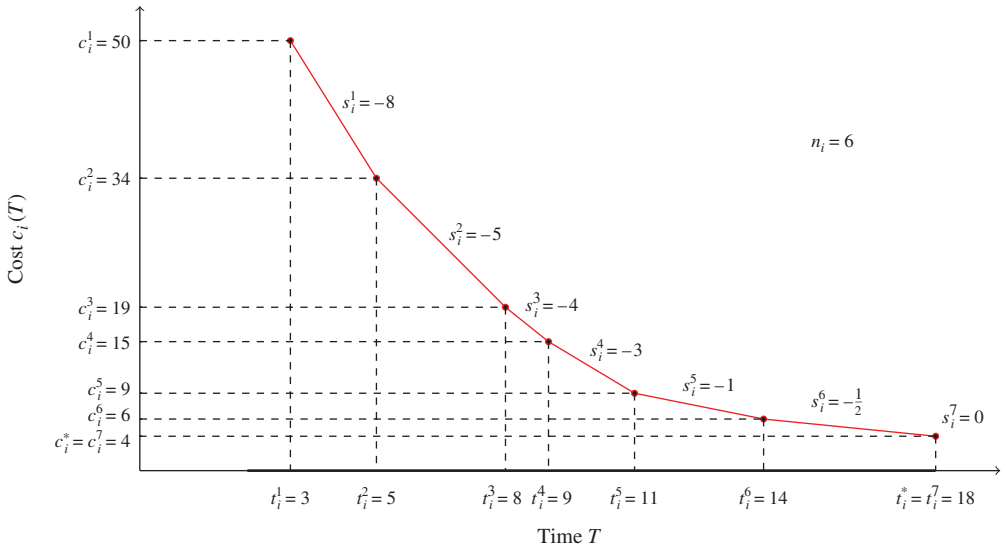
Example 1. We give a concrete numerical example for the extension step:

A label L_i at vertex i has $n_i = 6$ pieces defined as in Figure 3. The time window at vertex i is $[e_i, l_i] = [2, 18]$, while the domain of $c_i(T)$ is $[3, 18]$. As the last piece has a negative slope $s_i^6 = -\frac{1}{2}$, we have $t_i^* = l_i = 18$ with a minimum cost $c_i^* = 4$. We extend L_i along the arc (i, j) having a reduced cost $\tilde{c}_{ij} = 15$ and travel time $t_{ij} = 3$. The linear vertex cost of the head vertex j is assumed to be $\tilde{c}_j = -1$.

We will show how the new trade-off curve $c_j(T)$ at vertex j is computed. For the sake of explanation, we vary the time window $[e_j, l_j]$ of vertex j. The following table shows, depending on e_j and l_j (assuming $e_j \le l_j$), the resulting auxiliary values (f, g, δ) and the number n_j of pieces of the new trade-off curve $c_j(T)$.

In a first example, we choose $[e_j, l_j]$ so wide that all six pieces of the original curve $c_i(T)$ are transferred into the inner part of the time window, and an additional seventh piece must be created. This happens, for example, if $[e_j, l_j] = [0, 25]$. The corresponding values are $(f, g, \delta) = (1, 6, 1)$ giving us $n_j = 7$. Using Equation (5a) with

Figure 3. (Color online) Detailed Numerical Example of Extension Step



RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

Detailed Numerical Example of Extension Step

$e_j \le l_j$ required	$l_j = 6$	$6 < l_j \le 8$	$8 < l_j \le 11$	$11 < l_j \le 12$	$12 < l_j \le 14$	$14 < l_j \le 17$	$17 < l_j \le 21$	$l_j > 21$
$e_j < 8$	$(1,0,1)$ $n_j = 1$	$(1,1,0)$ $n_j = 1$ $(2,1,1)$ $n_j = 1$	$(1,2,0)$ $n_j = 2$ $(2,2,0)$ $n_j = 1$ $(3,2,1)$ $n_j = 1$	$(1,3,0)$ $n_j = 3$ $(2,3,0)$ $n_j = 2$ $(3,3,0)$ $n_j = 1$ $(4,3,1)$ $n_j = 1$	$(1,4,0)$ $n_j = 4$ $(2,4,0)$ $n_j = 3$ $(3,4,0)$ $n_j = 2$ $(4,4,0)$ $n_j = 1$ $(5,4,1)$ $n_j = 1$	$(1,5,0)$ $n_j = 5$ $(2,5,0)$ $n_j = 4$ $(3,5,0)$ $n_j = 3$ $(4,5,0)$ $n_j = 2$ $(5,5,0)$ $n_j = 1$ $(6,5,1)$ $n_j = 1$	$(1,6,0)$ $n_j = 6$ $(2,6,0)$ $n_j = 5$ $(3,6,0)$ $n_j = 4$ $(4,6,0)$ $n_j = 3$ $(5,6,0)$ $n_j = 2$ $(6,6,0)$ $n_j = 1$ $(7,6,1)$ $n_j = 1$	$(1,6,1)$ $n_j = 7$ $(2,6,1)$ $n_j = 6$ $(3,6,1)$ $n_j = 5$ $(4,6,1)$ $n_j = 4$ $(5,6,1)$ $n_j = 3$ $(6,6,1)$ $n_j = 2$ $(7,6,1)$ $n_j = 1$
$8 \le e_j < 11$								
$11 \le e_j < 12$								
$12 \le e_j < 14$								
$14 \le e_j < 17$								
$17 \le e_j < 21$								
$e_j \ge 21$								

$f_{ij}(t,s) = (\max\{e_j, t+t_{ij}\}, \min\{0, s-\tilde{c}_j\}) = (\max\{0, t+3\}, \min\{0, s-1\})$, we get $(t_j^1, s_j^1) = f_{ij}(t_i^1, s_i^1) = f_{ij}(3,-8) = (6,-9)$, $(t_j^2, s_j^2) = f_{ij}(t_i^2, s_i^2) = f_{ij}(5,-5) = (8,-6)$, $(t_j^3, s_j^3) = f_{ij}(t_i^3, s_i^3) = f_{ij}(8,-4) = (11,-5)$, $(t_j^4, s_j^4) = f_{ij}(t_i^4, s_i^4) = f_{ij}(9,-3) = (12,-4)$, $(t_j^5, s_j^5) = f_{ij}(t_i^5, s_i^5) = f_{ij}(11,-1) = (14,-2)$, and $(t_j^6, s_j^6) = f_{ij}(t_i^6, s_i^6) = f_{ij}(14,-\frac{1}{2}) = (17,-1.5)$. The seventh piece results from the zero slope piece $(t_i^7, s_i^7) = (18,0)$ at vertex i and is $(t_j^7, s_j^7) = f_{ij}(18,0) = (21,-1)$. The reduced cost at the start time $t_j^1 = 6$ is then computed with the help of Equation (5c). We obtain $c_i(\min\{t_i^*, t_j^1 - t_{ij}\}) = c_i(\min\{18, 6-3\}) = c_i(3) = 50$ resulting in $c_j^1 = 50 + \tilde{c}_{ij} + \tilde{c}_j t_j^1 = 50 + 15 + (-1)6 = 59$.

As a second set of examples, we show the extreme case that $c_j(T)$ has a single piece only, i.e., $n_j = 1$. This results if the time window $[e_j, l_j]$ is so small that only one of the pieces of $c_i(T)$ is relevant and no new piece is created as a last piece. This is true for $f = g$ and $\delta = 0$ (values above the diagonal in the table). For example, choosing $[e_j, l_j] = [13,14]$ gives $(f,g,\delta) = (4,4,0)$. With $f_{ij}(t,s) = (\max\{e_j, t + t_{ij}\}, \min\{0, s+\tilde{c}_j\}) = (\max\{13, t+3\}, \min\{0, s-1\})$, we get $(t_j^1, s_j^1) = f_{ij}(t_i^4, s_i^4) = f_{ij}(9,-3) = (13,-4)$. A single piece can also result if the domain of $c_j(T)$ is a single point in time $(t_j^1 = l_j)$ and a breakpoint of $c_i(T)$ is mapped into this degenerated domain. The latter condition is true if $l_j = t_i^f + t_{ij}$, which imposes $\delta = 1$ because of the last term in Equation (3). For example, the time window $[e_j, l_j] = [11,11]$ gives $(f,g,\delta) = (3,2,1)$ and thus $n_j = 1$ (values on diagonal of the above table). For the new extra piece, we get $(t_j^1, s_j^1) = f_{ij}(t_i^3, s_i^3) = f_{ij}(8,-4) = (11,-5)$ with $f_{ij}(t,s) = (\max\{e_j, t+t_{ij}\}, \min\{0, s+\tilde{c}_j\}) = (\max\{11, t+3\}, \min\{0, s-1\})$ according to Equation (5a).

Finally, we give a last example of a resulting curve with $n_j = 3$ pieces. We choose $[e_j, l_j] = [9,14]$ resulting in $(f,g,\delta) = (2,4,0)$. Again, the three resulting pieces of $c_j(T)$ are computed using Equation (5a) with $f_{ij}(t,s) = (\max\{9, t+3\}, \min\{0, s-1\})$. We get $(t_j^1, s_j^1) = f_{ij}(t_i^2, s_i^2) = f_{ij}(5,-5) = (9,-6)$, $(t_j^2, s_j^2) = f_{ij}(t_i^3, s_i^3) = f_{ij}(8,-4) = (11,-5)$, and $(t_j^3, s_j^3) = f_{ij}(t_i^4, s_i^4) = f_{ij}(9,-3) = (12,-4)$. The initial cost is $c_j^1 = 29 + 15 + (-1)9 = 35$, where $29 = c_1(\min\{t_i^*, t_j^1 - t_{ij}\}) = c_1(\min\{18, 9-3\}) = c_1(6)$.

We now proceed to the discussion of the dominance procedure and details of its implementation.

5.2.4. Dominance Between Labels. When checking a label L for dominance, it is compared one by one with all other undominated labels residing at the same vertex. During each such comparison between L and another label L', first, the sets S_L and $S_{L'}$ are compared. If $S_{L'} \subseteq S_L$, a pointwise dominance is performed with respect to each point on the trade-off curve of L, and the intervals on which L is dominated, i.e., where $c_{L'}(T) \le c_L(T)$ holds, are tentatively stored. We compute these domination intervals efficiently by determining the intersection points of $c_L(T)$ and $c_{L'}(T)$. Ioachim et al. (1998) have shown that even if the intersection points may be noninteger real numbers, it suffices to describe the intervals with integer bounds. Also Liberatore, Righini, and Salani (2011) exploit domination intervals, however, they delete the dominated intervals from the trade-off curves so that they can become discontinuous.

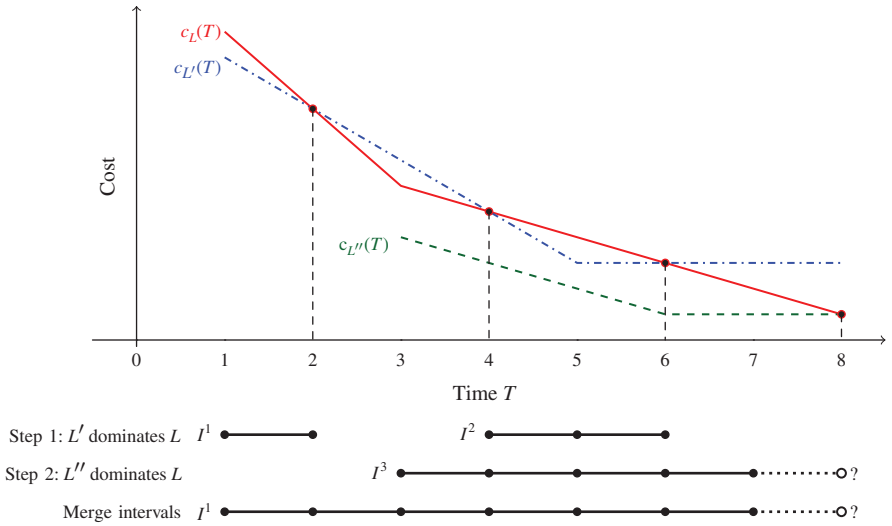
After having compared L with all other labels in this way, these intervals are used to update $(I^d)_{d=1}^k$, the intervals on which the trade-off curve of L is dominated, and k, the number of intervals on which L is dominated. If the complete trade-off curve of a label is dominated, the label itself is dominated and can be discarded. Note that this means that the decision on whether or not a label is obsolete will regularly be based on comparisons with more than one other label. Put differently, in most cases, only several other labels together will make one label obsolete. This is in contrast to dominance procedures for classical VRPs, where a pairwise dominance is applicable and one label dominates another one or not.

To avoid that two labels L and L' dominate each other on an interval I^d when $S_L = S_{L'}$ and $c_L(T) = c_{L'}(T)$ for all

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

Figure 4. (Color online) Detailed Example of Dominance



$T \in I^d$, we use a tie-breaking rule, similar to standard VRPs where it must be avoided that two identical labels eliminate each other.

Example 2. We explain the dominance mechanism with the help of another concrete example depicted in Figure 4.

We assume that a label L is given with trade-off curve $c_L(T)$. Initially, L is not dominated, i.e., $k = 0$. In a first step, another label L' with trade-off curve $c_{L'}(T)$ is generated at the same vertex. Assuming $S_{L'} \subsetneq S_L$, this second label dominates L on the intervals $[1,2]$ and $[4,6]$. This information is stored within L setting $k = 2$ and $(I^1, I^2) = ([1,2],[4,6])$.

In the second step, a third label L'' with trade-off curve $c_{L''}(T)$ is generated. Now we assume that $S_{L''} = S_L$ holds. Clearly, L'' dominates L on the half-open interval $[3,8)$ because $c_{L''}(T) < c_L(T)$ for all $T \in [3,8)$. Since $c_{L''}(8) = c_L(8)$, dominance at time $T = 8$ depends on the tie-breaking rule. Therefore, the dominance interval I^3 is either $[3,7]$ or $[3,8]$ and in Figure 4 the right end of the interval is dotted.

Merging the dominance intervals I^1, I^2, and I^3 leads to a single interval, even if neither $c_{L'}(T)$ nor $c_{L'}(T)$ is below $c_L(T)$ on the open interval $(2,3)$. As stressed previously, it suffices that dominance reflects the properties of schedules at integer points in time. As a result, the merging results in $k = 1$ and, depending on the tie-breaking rule, the new dominance interval is $I^1 = [1,7]$ or $I^1 = [1,8]$. Only in the latter case, I^1 comprises the complete domain of $c_L(T)$ so that label L can be discarded.

Refinements of the dominance procedure that speed up the pricing process are the elimination of dominated pieces at the beginning or the end of the range of the trade-off curve and the replacement of consecutive dominated pieces by one aggregated piece.

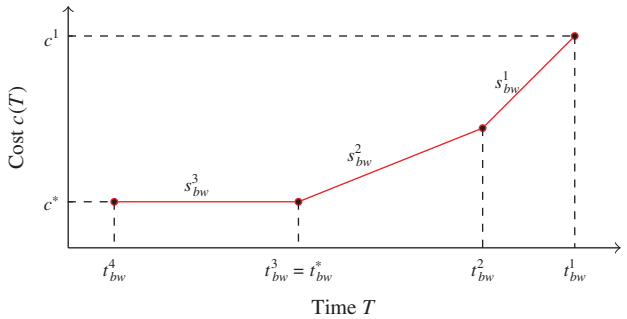
5.2.5. Bidirectional Labeling.

We start by briefly describing the backward label extension before we detail the bidirectional labeling approach.

Given the forward label extension process, the backward label extension process is rather simple: it is sufficient to invert all time windows and to invert the linear vertex costs. To be precise, a time window $[e_r, l_r]$ for a request r is replaced by $[t^{\max} - l_r, t^{\max} - e_r]$, $\tilde{c}$ is replaced by $-\tilde{c}$, and then the same algorithm as in the forward labeling is applied. Note that it suffices to compute $t^{\max} - T$ to recalculate the real time for a given point in time T on the trade-off curve of the backward label. For simplicity, when a point in time on the trade-off curve of the backward label is mentioned in the following, we refer to real time. Hence, the pieces are numbered from 1 to n with decreasing, nonnegative slopes, and t^1 is the latest feasible time for a backward label. Figure 5 summarizes the notation for backward labels.

In bidirectional labeling algorithms, forward labels are not necessarily propagated until the end depot, and backward labels are not necessarily propagated until the start depot. Instead, labels are propagated only up

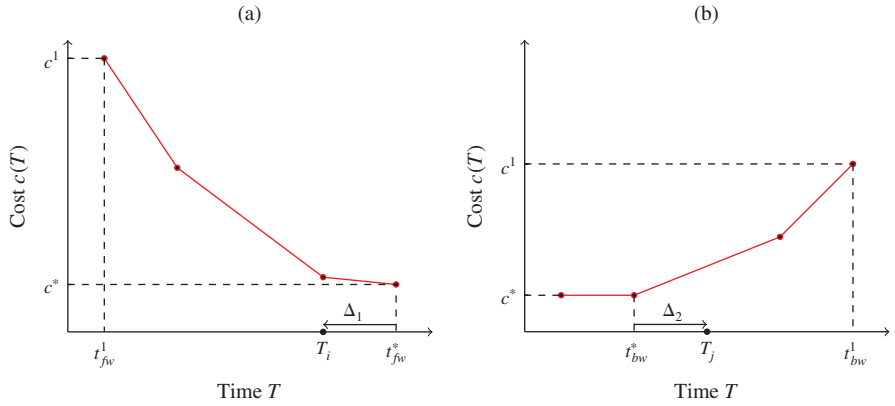
Figure 5. (Color online) Example of a Backward Label with $n = 3$ Pieces



RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

Figure 6. (Color online) Example of a Merge Step: Forward and Backward Label with Optimal Times T_i and T_j for their Concatenation



to a so-called halfway point, thus limiting the overall number of created labels. Suitable forward and backward labels must then be merged to obtain complete o-d-paths. As described by Salani (2005, Section 4.6.4), this is done using a halfway point test to avoid creating the same path from different pairs of forward and backward labels. Setting $t^{\max}/2$ as the halfway point, we propagate forward labels at a vertex i only if $t^1_i \le t^{\max}/2$, and backward labels at a vertex j only if $t^1_j > t^{\max}/2$.

We then merge on vertices, i.e., we consider forward and backward labels at the same vertex i. A forward label at a vertex i qualifies for merging if $i = d$ or its earliest service start time is $t^1_{fw} > t^{\max}/2$. This condition prevents the creation of identical paths from different pairs of forward and backward labels. We check all backward labels at vertex i for whether or not they can be merged with the chosen forward label. The procedure becomes simpler if instead of the backward label itself, its predecessor label is considered. Let this predecessor be resident at vertex j. Note that this is a kind of merging along the resulting arc (i, j). However, with the merge on vertices we benefit from the fact that both the forward and backward label are Pareto optimal. This is in contrast to the situation when merging is directly performed over arcs, where neither the forward extension of the forward label along (i, j) nor the backward extension of the backward label along (i, j) is guaranteed to be Pareto optimal, leading to considerably more pairs of labels to be checked.

We consider forward labels at a vertex i for a merge with predecessor backward labels at a vertex j if the following conditions are fulfilled to obtain a feasible path:

$$S_{fw} \cap S_{bw} = \varnothing \quad \text{and} \quad t^1_{fw} + t_{ij} \le t^1_{bw}. \tag{7}$$

The first condition is necessary to avoid paths violating the partial elementarity required by the ng-path relaxation, the second is needed to ensure overall temporal feasibility. Now assume that these conditions are fulfilled for a forward label at a vertex i and a backward label at a vertex j, and consider Figure 6. On the left-hand side of Figure 6, an exemplary forward cost function at i is depicted. The predecessor backward label with an exemplary backward linear cost function at j is depicted on the right-hand side. We have to compute the optimal service start times T_i and T_j at the vertices i and j, respectively, so that all other service start times at the vertices in the path can be derived from these values; see Equations (6). There are two cases:

(i) $t^*_{fw} + t_{ij} \le t^*_{bw}$. This is unproblematic: $T_i = t^*_{fw}$ and $T_j = t^*_{bw}$ are the optimal service start times for vertices i and j.

(ii) $\Delta = t^*_{fw} + t_{ij} - t^*_{bw} > 0$. In this case, T_i must be set to an earlier point in time than t^*_{fw}, and/or T_j must be set to a later point in time than t^*_{bw}. To ensure optimality, we compute the optimal times T_i and T_j by distributing the total required time shift Δ between the forward and the backward label. To distribute Δ, we initialize $T_i = t^*_{fw}$ and $T_j = t^*_{bw}$ and update them in the following manner. We consider the slopes of the cost function pieces. If, for example, $t^*_{fw} = t^{n+1}_{fw}$, $t^*_{bw} = t^n_{bw}$, and $|s^n_{fw}| \le |s^{n-1}_{bw}|$, as is the case in Figure 6, setting T_i to an earlier time than t^*_{fw} will increase the costs of the merged path by at most as much as pushing T_j to a later point in time than t^*_{bw}. Hence, we shift T_i backward by $\partial_1 := \min\{T_i - t^n_{fw}, \Delta\}$ units, i.e., $T_i := T_i - \partial_1$. The remaining time shift is $\Delta := \Delta - \partial_1$. Now, if $\Delta = 0$, we are done. Otherwise, we compare the slope of the preceding forward piece, here s^{n-1}_{fw}, with s^{n-1}_{bw} and select the piece with the smaller absolute slope. If, as depicted in the figure, $|s^{n-1}_{bw}| < |s^{n-1}_{fw}|$, we shift T_j forward by $\partial_2 := \min\{t^{n-1}_{bw} - T_j, \Delta\}$ units, i.e., $T_j := T_j + \partial_2$. Again, the remaining time shift is $\Delta := \Delta - \partial_2$. We iterate until $\Delta = 0$, and this is guaranteed to happen because of condition (7). The overall algorithm is depicted in Algorithm 1. Note that if the next forward piece to consider does not exist, i.e., if the merge point T_i is already equal to the start time t^1_{fw} of the forward trade-off curve, we take the backward

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

one, and vice versa. This can be achieved by defining $s^0_{fw} := -\infty$ and $s^0_{bw} := +\infty$.

In the merge procedure just described, the cost functions of backward labels are relevant only in the interval $[t^{\max}/2, t^{\max}]$ from the halfway point to the end of the planning horizon. Therefore, it is sufficient to construct the trade-off curve of a backward label only up to $t^{\max}/2$. This saves some computation time when constructing the linear cost function while maintaining optimality of the procedure.

The reduced cost of a merged path can be obtained by $c^*_m := c_{fw}(T_i) + c_{bw}(T_j) + \tilde{c}_{ij}$. After the merge, we perform a final dominance test as described in Section 5.2.4 for all routes resulting from merged labels. Finally, we add all undominated negative reduced-cost routes to the master problem.

Algorithm 1 Computation of optimal service start times T_i and T_j when merging

$\Delta := t^*_{fw} + t_{ij} - t^*_{bw}$, $T_i := t^*_{fw}$, $T_j := t^*_{bw}$
if $(s^n_{fw} < 0)$ **then** $p_{fw} := n$ **else** $p_{fw} := n-1$
if $(s^n_{bw} > 0)$ **then** $p_{bw} := n$ **else** $p_{bw} := n-1$
while $(\Delta > 0)$ **do**
  if $(|s^{p_{fw}}_{fw}| < |s^{p_{bw}}_{bw}|)$ **then**
    $\partial := \min\{T_i - t^{p_{fw}}_{fw}, \Delta\}$
    $T_i := T_i - \partial$, $p_{fw} := p_{fw} - 1$
  else
    $\partial := \min\{t^{p_{bw}}_{bw} - T_j, \Delta\}$
    $T_j := T_j + \partial$, $p_{bw} := p_{bw} - 1$
  $\Delta := \Delta - \partial$

Result: Optimal service start times T_i and T_j

5.2.6. Acceleration Techniques. To accelerate the pricing process, we use a heuristic pricing procedure, the so-called *limited discrepancy search* (LDS) introduced by Feillet, Gendreau, and Rousseau (2007), and a heuristic dominance.

LDS works as follows. In each pricing iteration, the outgoing arcs of each vertex are partitioned into "good" and "bad" arcs. The arcs with the lowest current modified arc costs and all arcs incident to the start or the end depot are considered good arcs, the others are regarded as bad arcs. The labels have an additional attribute storing the number of bad arcs used in the associated path. The value of this attribute is incremented by one each time a label is extended along a bad arc. Only those labels are considered feasible for which the number of bad arcs used is below a specified upper bound, the discrepancy limit. This limit is set to a fixed value during the complete solution process.

As for heuristic dominance, we use a pairwise comparison of two labels L and L', and if $c^*_L < c^*_{L'}$ and $t^1_L < t^1_{L'}$, we discard L'. The limited discrepancy search, as well as the heuristic dominance, are applied in each column generation iteration. Only if they fail to produce a negative reduced cost column, the exact pricing and dominance algorithms are applied.

5.3. Branching Strategy

Let $\tilde{\lambda}^{aq}$, $\tilde{u}_r$, and $\tilde{x}^a_{ij} = \sum_{q\in\Omega^a} X^q_{ij}\tilde{\lambda}^{aq}$ be the values of the corresponding decision variables λ^{aq}, u_r, and x^a_{ij}. We apply the following five-stage hierarchical branching scheme: First, if the overall number of unserved requests, i.e., $u_\Sigma = \sum_{r\in R}\tilde{u}_r$, is fractional, we create the two branches $\sum_{r\in R} u_r \le \lfloor u_\Sigma \rfloor$ and $\sum_{r\in R} u_r \ge \lceil u_\Sigma \rceil$. If the objective (1a) prioritizes request fulfillment, i.e., $\gamma \gg \alpha, \beta$, the latter branch is obsolete. Second, we branch on the individual u_r variables, where one request r^* with $\tilde{u}_{r^*}$ closest to $1/2$ is selected, and the two branches $u_{r^*} = 0$ and $u_{r^*} = 1$ are created. Third, we branch on the overall number of active vehicles in use: if $a_\Sigma = \sum_{a\in A}\sum_{q\in\Omega^a}\tilde{\lambda}^{aq}$ is fractional, the resulting branches are given by $\sum_{a\in A}\sum_{q\in\Omega^a}\lambda^{aq} \le \lfloor a_\Sigma \rfloor$ and $\sum_{a\in A}\sum_{q\in\Omega^a}\lambda^{aq} \ge \lceil a_\Sigma \rceil$. All these branching rules are put into effect by adding a constraint to the master program (1). In addition, when the second branching rule is applied and u_{r^*} is set to 1, all of the subgraphs induced by vertices in $V^-_{r^*} \cup W^+_{r^*} \cup W^-_{r^*} \cup V^+_{r^*}$ can be removed from all networks G^a, $a \in A$.

Fourth, note that for arcs $(i_p, j_{p'}) \in E$ that have both end points in N^R, at most one of the arcs $\bigcup_{q,q'\in P}\{(i_q, j_{q'})\}$ of one of the networks G^a, $a \in A$, can be present in a solution. Hence, if $\Sigma_{ij} = \sum_{a\in A}\sum_{q,q'\in P^a}\tilde{x}^a_{i_q,j_{q'}}$ is fractional, we create two branches by setting $\sum_{a\in A}\sum_{q,q'\in P^a} x^a_{i_q,j_{q'}}$ to zero and one, respectively. If only one of the end points is in N^R, we can apply a similar branching rule. If several Σ_{ij} are fractional, we choose a pair (i, j) with a value closest to $1/2$. The zero-branch is implemented by eliminating the associated arcs from all networks G^a, $a \in A$. The one-branch first fixes $u_r = 0$ for the corresponding request(s) r associated with i_p and $j_{p'}$ and eliminates incompatible arcs from all networks G^a, $a \in A$.

The fifth and last rule is branching on individual arcs, which finally ensures integrality of the x^a_{ij} and u_r variables. (Recall that model (1) poses no integer requirement on the λ^{aq} variables.) If $\tilde{x}^a_{ij}$ for an arc $(i, j) \in E^a$ is fractional, then one can branch on $x^a_{ij} = 0$ and $x^a_{ij} = 1$, where we select the combination (a, i, j) for which $\tilde{x}^a_{ij}$ is closest to $1/2$. Branching on individual arcs $(i, j) \in E^a$ is implemented by eliminating (i, j) from the underlying network G^a for $x^a_{ij} = 0$, while for the branch $x^a_{ij} = 1$, all ingoing arcs $\delta^-(i)$ are eliminated for networks $G^{a'}$, $a' \ne a$, and all arcs $\delta^+(i)\backslash\{(i, j)\}$ are eliminated from the network G^a.

As the branch-and-bound node-selection rule, we apply a best-bound-first strategy, because our primary goal is to improve the dual bound.

5.4. Cutting Planes

To strengthen the formulation, we extend formulation (1) with subset-row inequalities (Jepsen et al. 2008). For the APVRP, a valid inequality is defined on

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

a subset of tasks instead of vertices as done for defining ng-neighborhoods; see Section 5.2.2. We restrict ourselves to those inequalities defined on three tasks as proposed by Jepsen et al. (2008) because they can be separated by straightforward enumeration. The inequality for a task set U_k, in the following denoted by $SR(U_k)$, is given by $\sum_{a\in A}\sum_{q\in\Omega}\lfloor h^q/2\rfloor\lambda^{aq} \le 1$, where h^q is the number of times route q serves a task in U_k.

The addition of subset-row inequalities in the master problem requires the following adjustments to our pricing problems: Let $\eta_k \le 0$ be the dual price of the subset-row inequality $SR(U_k)$. The value η_k must be subtracted from the reduced costs for every second service to tasks in U_k. Therefore, an additional binary resource sr^k, one for each inequality $SR(U_k)$, is necessary in the labeling algorithm for indicating the parity of the number of times a task in U_k is served. Note that the same task may be served more than once in the ng-path relaxation.

Jepsen et al. (2008) proposed a tailored dominance rule that avoids a pointwise comparison of all resources sr^k and thereby reduces the number of incomparable labels significantly. In the APVRP case, the dominance rule changes as follows: A precondition for a label L' to dominate another label L is $S_{L'} \subseteq S_L$. Moreover, let $H = \{k: sr^k_{L'} = 1, sr^k_L = 0\}$ (the index set of new resources on which L' is inferior compared to L). Then, dominance is given at those times T, where $c'_L(T) - \sum_{k\in H}\eta_k \le c_L(T)$.

In addition, the merge procedure of the bidirectional labeling algorithm slightly changes. If both a forward and a backward path have served an odd number of tasks in U_k, then the dual price η_k of the subset-row inequality $SR(U_k)$ has to be subtracted from the reduced costs.

6. Experimental Results

The results reported in this section were obtained using a standard PC with an Intel Core i7-2600 3.4 GHz processor and 16 GB RAM. The algorithms were coded in C++ with MS-Visual Studio 2010. The callable library of CPLEX 12.5 was used for solving the linear relaxations of the restricted master program in the column-generation algorithm.

6.1. Test Instances

Meisel and Kopfer (2014) created a set of 180 APVRP test instances (henceforth referred to as MK instances) with the following characteristics: The instances range in size from 38–1,600 tasks, 2–25 active vehicles, 4–50 passive vehicles, and a planning horizon $t^{\max}$ of 2,500–5,000 time units. Locations are randomly placed in a 100×100 area, distances and travel times are set to the Euclidean distance, time windows are of width 1,000 with random start times in $[0, t^{\max}-1{,}000]$, and service times for loading and unloading range between 50 and 100 time units. Each passive vehicle is compatible with two active vehicles; each request is compatible with three passive vehicles. Because of their huge planning horizon $t^{\max}$ where time units can model minutes of a working week, these instances are very hard to solve with an exact approach.

Hence, we additionally created a new set of instances, in which a time unit can model 10 minutes of a working week. It is clear that our new instances are on average easier to solve than the MK instances, however, the chosen time discretization is fine enough for many practical situations. The new set contains 20 instances with 38 tasks, two active, and four passive vehicles (leading to extended networks with approximately 170 vertices and 3,500 arcs), and 20 instances with 76 tasks, four active, and eight passive vehicles (resulting in extended networks with ca. 650 vertices and 4,000 arcs). The instances have a planning horizon of 1,000 time units, and service times s^+_r and s^-_r vary between 25 and 50. For each of the 40 instances, we assigned time windows that allow different levels of flexibility: For each request $r \in R$, we first compute the travel time between its pickup and its delivery location. Initially, each request time window $[e_r, l_r]$ (refer to Table 1 for the definition of e_r and l_r) is set so that the time window width $l_r - e_r$ is equal to the computed travel time plus the service times s^+_r and s^-_r. By adding a *time window flexibility* of $F = 25$, 50, 100, and 200, we create 160 new instances. A time window flexibility of F means that e_r and l_r are further shifted apart by exactly F time units (it is ensured that the new $[e_r, l_r]$ is in the overall planning horizon $[0, t^{\max}]$). All other characteristics of the new instances are the same as in the MK instances.

6.2. Algorithmic Setup

For all experiments, we used objective function weights of $\alpha = 10$, $\beta = 1$, and $\gamma = 10{,}000$, i.e., a hierarchical objective of first fulfilling as many requests as possible, then minimizing traveled distance, and then minimizing route completion time. Feasibility of the restricted master program is ensured by initializing it with dummy routes each bringing a passive vehicle from its origin to its destination as well as with columns for the u_r variables. We set a CPU time limit of two hours. Preliminary tests showed that bidirectional labeling was superior to the unidirectional variant, that the best results were obtained with an ng-neighborhood size of 15, applying limited discrepancy search and heuristic dominance during the pricing as described in Section 5.2.6, and stopping each labeling procedure as soon as 100 or more negative reduced cost routes have been found.

6.3. Algorithmic Performance

Here, we present the experimental results for different setups. First, we report the results of the branch-and-price algorithm obtained without using subset-row inequalities. After that, we describe the results

RIGHTSLINK

Table 4. Results on 38-Task Instances without Cuts

TW flexibility	No. solved	Time (sec)			Gap at root (%)			Gap closed (%)			No. of nodes solved		
		Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
25	20/20	36	191	931	0.00	0.43	1.65	100.00	100.00	100.00	1	5.40	23
50	20/20	43	362	2,485	0.00	0.55	2.11	100.00	100.00	100.00	1	8.30	47
100	19/20	38	1,076	7,200	0.00	0.99	3.00	64.29	98.21	100.00	1	16.95	56
200	14/20	109	3,466	7,200	0.00	2.67	9.89	8.94	79.94	100.00	1	26.20	106
All	73/80		1,274			1.16			94.54			14.21	

of the full branch-and-price-and-cut algorithm. Finally, we discuss the impact of time window flexibility on solution quality.

6.3.1. Branch-and-Price Results. In a first series of experiments, we applied our algorithm without adding cutting planes. As for the MK instances, Meisel and Kopfer (2014) could not solve any of these to optimality with the branch-and-cut algorithm presented in their paper. Our algorithm was able to solve to optimality one 38-task MK instance within the two-hour time limit. For this solved instance the previous upper bound was improved by 10.05%. The average gap between the upper and lower bound after two hours was 6.79% for the MK instances, with a minimum nonzero gap of 3.47% and a maximum gap of 13.90%. The average increase in the lower bound at the root node of the branch-and-bound tree compared to the branch-and-cut by Meisel and Kopfer (2014) was significant (18.35%).

Tables 4 and 5 report the results on the test instances we created. Both tables have the same columns. The first column indicates the time window flexibility F of the instances considered in each row, the second specifies the number of instances solved to optimality. The next columns indicate minimal, average, and maximal values of (i) the CPU times; (ii) the gap between the value of the LP relaxation at the root node and the best known upper bound (also using upper bounds for instances with smaller F as bounds for instances with a bigger F); (iii) the percentage of the root gap closed at the end of the optimization (100.0% for instances that were solved to optimality); and (iv) the number of branch-and-bound nodes solved. It can be seen from the tables that we are able to consistently solve instances with 38 tasks and a time window flexibility of up to 100. In general, the instances become more difficult with increasing time window flexibility, and timeout is reached for most 76-task instances, with significant gaps remaining. For the 76-task instances with a time window flexibility of 100 and 200, the number of solved branch-and-bound nodes even decreases, on average, compared to the instances with shorter time windows. This means that the time needed to solve a node increases considerably. Analyses showed that this is mostly caused by the slow convergence of the master program as well as the increasing solution times for the pricing problems, which, in turn, are due to the growing number of labels created because of the greater flexibility offered by longer time windows.

6.3.2. Branch-and-Price-and-Cut Results. The dynamic programming algorithm can become quite slow when too many subset-row inequalities are present in the restricted master problem. To overcome this, we use the following two parameters to define the strategy used for separating inequalities: $Cut_{\max}$ gives the maximum number of inequalities that can be added to the restricted master problem. When more than $Cut_{\max}$ inequalities are violated, we choose the most violated ones. Cut_{task} gives the maximum number of inequalities that a single task can be involved in. Preliminary tests have shown that the best results are obtained by choosing $Cut_{\max} = 5$ and $Cut_{\text{task}}=2$.

The results for the MK instances are similar to those without using subset-row inequalities: One instance was solved to optimality, the average gap between the upper and lower bound after two hours was 7.30%, with a minimum nonzero gap of 3.16% and a maximum gap of 13.90%. Tables EC.1 and EC.2 in the online appendix report the results on the test instances we created. The columns contain the same information as in the previous paragraph, but the gap at the root lower bound is missing because it is the same as in the tables before. In addition, we have the minimum, average, and maximum percentage that the root gap was closed only by the subset-row inequalities. Also, here the results do not differ significantly. We can solve as many 38-task instances as before, but the average computation time increases by nearly 200 seconds and the average percentage the root gap was closed slightly decreases. The detailed results in the online appendix show that one of the previously solved instances was not solved, but another one was solved for the first time. For the 76-task instances, we can solve one less instance, but the average computation time decreases by more than 300 seconds and the average percentage the root gap was closed increases by more than 2%. For both instance sets, the number of solved branch-and-bound nodes decreases by around 3.5, implying that the time for solving one

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

Table 5. Results on 76-Task Instances without Cuts

TW flexibility	No. solved	Time (sec)			Gap at root (%)			Gap closed (%)			No. of nodes solved		
		Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
25	17/20	131	3,594	7,200	0.00	0.80	2.39	46.52	90.54	100.00	1	22.70	49
50	8/20	282	5,419	7,200	0.28	1.55	4.30	21.26	67.96	100.00	3	25.90	46
100	2/20	2,814	6,882	7,200	0.47	3.53	6.94	10.70	34.11	100.00	11	25.50	45
200	0/20	7,200	7,200	7,200	6.10	8.48	13.47	1.79	6.27	15.35	5	11.60	19
All	27/80		5,775			3.59			49.72			21.43	

Table 6. Impact of Time Window Flexibility

TW flexibility increase	Objective improvement (%)	
	38 tasks	76 tasks
25 → 50	1.20	1.54
50 → 100	1.94	1.56
100 → 200	5.84	N/A

branch-and-bound-node increases significantly. This is because of the fact that the exact dominance becomes quite weak, even with a small number of subset-row inequalities. Detailed results have shown that nearly three times more labels are generated in the dynamic programming algorithm when subset-row inequalities are present in the restricted master problem. This is because of the fact that we need several labels to dominate one other label.

6.3.3. Impact of Time Window Flexibility. Table 6 shows that there is a positive impact of increasing time window flexibility on possible solution quality. The values in the table reflect the objective function improvement obtained with increasing time window flexibility. For example, for the 38-task instances, the average objective function value improves by 1.20% if the time window flexibility is increased from 25 to 50.

7. Conclusion

This paper has investigated the exact solution of the APVRP by means of a branch-and-price-and-cut method. The problem includes synchronization constraints for the operations and the movement of active and passive vehicles. It supports a flexible coupling of these transport resources to achieve an efficient resource utilization and high-quality transport solutions. The synchronization constraints introduce linear vertex costs in the ESPPRC pricing problem, thus significantly complicating its solution. To solve the pricing problems, we use a bidirectional labeling algorithm and apply a refinement of the ng-path relaxation approach. What is more, we propose a sophisticated procedure for merging forward and backward labels. Computational experiments show that our method delivers improved results for the APVRP benchmark instances introduced by Meisel and Kopfer (2014). For a newly created benchmark set, it is capable of solving instances with 76 tasks, four active, and eight passive vehicles to optimality.

In its current form, the APVRP already covers a number of relevant applications from the areas of distribution logistics, healthcare management, the security industry, and others. Nevertheless, there are several further features of real-world problems that are not yet supported by the presented model. This includes, for example, the possibility of a temporary drop-off of passive vehicles at intermediate locations (e.g., public parking lots) that neither belong to the set of initial or final locations nor to the set of pickup and delivery locations. Such drop-off locations could reduce the detours needed by active vehicles for the exchange of passive vehicles. Furthermore, a rendezvous of active vehicles is required in some applications where passive vehicles cannot be left behind but must be handed over directly from one active vehicle to another. This is the case if only one of the active vehicles is equipped with a certain loading equipment, if the passive vehicle cannot be left unattended for security reasons, or if cargo documents need to be exchanged among the active vehicles. Moreover, in many or even most practical applications, vehicles with a capacity of more than one are used, and performing load transshipments between such vehicles is quite common in practice. These features and their corresponding synchronization requirements represent practically relevant and mathematically nontrivial extensions of the APVRP and will be a subject of our future research.

Appendix. Notation

Sets

- R Set of all requests;
- r A request;
- A Set of classes of active vehicles;
- a A class of active vehicles;
- K_a Number of active vehicles of class a;
- P Set of passive vehicles;
- p A passive vehicle;
- R^p Set of requests compatible with passive vehicle p;
- P^r Set of passive vehicles compatible with request r;
- P^a Set of passive vehicles compatible with active vehicle a;
- A^p Set of classes of active vehicles compatible with passive vehicle p.

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

Locations

ℓ_r^+ The pickup location of request r;
ℓ_r^- The delivery location of request r;
o Initial location of all active vehicles;
d Final location of all active vehicles;
o_p Initial location of passive vehicle p;
d_p Final location of passive vehicle p.

Cost parameters

α Weight for travel distance;
β Weight for arrival time at the final depot d;
γ Penalty for not fulfilling a request;
c_{ij} Travel distance from vertex i to vertex j.

Time parameters

t_{ij} Travel and service time from vertex i to vertex j;
$[e_r, l_r]$ Time window for fulfilling request $r \in R$;
s_r^+ The service duration to process an empty passive vehicle at request r;
s_r^- The service duration to process a loaded passive vehicle at request r;
$[0, t^{\max}]$ Planning horizon.

Network parameters

$G^a = (V^a, E^a)$ Extended graph for active vehicle class $a \in A$ with vertex set V^a and arc set E^a;
$\delta^+(i)$ $\{j \in V^a : (i,j) \in E^a\}$; forward star of vertex $i \in V^a$;
$\delta^-(i)$ $\{j \in V^a : (j,i) \in E^a\}$; backward star of vertex $i \in V^a$;
N^R $\bigcup_{r \in R}(V_r^- \cup W_r^+ \cup W_r^- \cup V_r^+)$; set of request vertices.

For each passive vehicle $p \in P^a$ and each compatible request $r \in R^p$:

v_{rp}^- Vertex where empty passive vehicle is delivered;
w_{rp}^+ Vertex where loaded passive vehicle is picked up;
w_{rp}^- Vertex where loaded passive vehicle is delivered;
v_{rp}^+ Vertex where empty passive vehicle is picked up;
N^a $\bigcup_{p \in P^a, r \in R^p}\{v_{rp}^-, w_{rp}^+, w_{rp}^-, v_{rp}^+\}$; set of request vertices for each passive vehicle $p \in P^a$ and each compatible request $r \in R^p$.

For each request $r \in R$:

V_r^- $\{v_{rp}^- : p \in P^r\}$; set of vertices where empty passive vehicle may be delivered for pickup of r;
W_r^+ $\{w_{rp}^+ : p \in P^r\}$; set of vertices where passive vehicle loaded with r may be picked up;
W_r^- $\{w_{rp}^- : p \in P^r\}$; set of vertices to where passive vehicle loaded with r may be delivered;
V_r^+ $\{v_{rp}^+ : p \in P^r\}$; set of vertices where empty passive vehicle may be picked up after delivery of r.

Set-partitioning formulation parameters

Ω^a Set of all routes that can be defined in graph $G^a = (V^a, E^a)$, $a \in A$;
q A route in $\bigcup_{a \in A} \Omega^a$;
X_{ij}^q Number of times arc (i,j) is traversed on route q;
T_i^q Service start time at request vertex i on route q;
b_i^q Number of times vertex i is visited on route q;
c^q Cost of route q; weighted sum of arrival time at the destination d and length of the route;
λ^{aq} Continuous variables measuring the flow of active vehicles of class $a \in A$ along route $q \in \Omega^a$;
x_{ij}^a Binary variables indicating the number of times arc $(i,j) \in E^a$ is traversed by an active vehicle of class $a \in A$;
u_r Binary variable indicating whether or not request $r \in R$ remains unfulfilled.

Pricing problem parameters

π_r Unrestricted dual variable of constraint (1b);
ϕ_{rp}^+ Unrestricted dual variable of constraint (1c);
ϕ_{rp}^- Unrestricted dual variable of constraint (1d);
τ_r^+ Nonnegative dual variable of constraint (1e);
τ_r^- Nonnegative dual variable of constraint (1f);
μ_p^P Unrestricted dual variable of constraint (1g);
μ_a^A Nonpositive dual variable of constraint (1h);
$\tilde{c}_{ij}$ Reduced costs of arc (i,j);
$\tilde{c}_i$ Linear vertex costs of vertex i;
x_{ij} Binary variable indicating whether or not arc $(i,j) \in E^a$ is traversed;
T_i Continuous variable indicating the point in time when the service at vertex $i \in V^a$ begins.

Labeling algorithm parameters

τ A task;
$\mathcal{N}_i$ ng-neighborhood at vertex i; $\mathcal{N}_i \subset N^R$; $\mathcal{N}_i$ contains i and the ν closest request vertices j for which a cycle $(j, \ldots, i, \ldots, j)$ would be feasible when taking time windows and travel and service times into account;
ν Size of ng-neighborhood;
τ_p^o Task of picking up the passive vehicle $p \in P$ at its initial location;
τ_r^1 Task of delivering an empty passive vehicle for request $r \in R$;
τ_r^2 Task of picking up and delivering a passive vehicle loaded with request $r \in R$;
τ_r^3 Task of picking up an empty passive vehicle after delivering request $r \in R$;
τ_p^d Task of delivering the passive vehicle $p \in P$ at its final location;
L $L = ((t^p, s^p)_{p=1}^{n+1}, c^1, S, (I^d)_{d=1}^k)$; a label;
n Number of time-slope pairs/pieces in trade-off curve;
$(t^p, s^p)_{p=1}^n$ The n pieces of the trade-off curve; $s^p < 0$, $p = 1, \ldots, n-1$ and $s^n \le 0$;
t^{n+1} End time of last piece n in trade-off curve;
$c(T) = c^1 + \sum_{p=0}^{q-1} s^p(t^{p+1} - t^p) + s^q(T - t^q)$ for $t^q \le T \le t^{q+1}$; the piecewise linear cost function/trade-off curve of a label, i.e., a label's associated partial path; provides the minimal costs $c(t)$ incurred by the path when the service at the last vertex of the path starts at time t;
c^1 Reduced costs at start time of first piece in trade-off curve;
S Tasks that have already been performed or are unreachable along the route and which are taken into account for checking elementarity in the label extension step according to the ng-path relaxation;
k Number of intervals on which the associated label is dominated;
$(I^d)_{d=1}^k$ The k intervals on which the associated label is dominated;
$c^* = c^1 - \sum_{p=1}^n s^p(t^{p+1} - t^p)$; optimal (reduced) costs;
$t^* = t^n$ for $s^n = 0$, t^{n+1} otherwise; earliest time to obtain costs c^*;

RIGHTSLINK

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

$s^{n+1} = 0$; slope of $(n+1)$th piece; defined for convenience;
$\mathcal{T}_j^{\text{test}}$ Before extending a label L at vertex i to vertex j, tasks in $\mathcal{T}_j^{\text{test}}$ are tested and L is extended only if none of these tasks has already been fulfilled along the partial path represented by L;
$\mathcal{T}_j^{\text{set}}$ Upon extending a label L at vertex i to vertex j, tasks in $\mathcal{T}_j^{\text{set}}$ are marked as fulfilled;
f Index of first linear slope piece to be kept when extending a label;
g Index of last linear slope piece to be kept when extending a label;
δ Piece indicator; one if a new linear slope piece must be created upon extension of a label, zero otherwise;
$f_{ij}(t,s) = (\{\max\{e_j, t + t_{ij}\}\}, \min\{0, s + \tilde{c}_j\})$; function for extending pieces during label extension.

References

Afifi S, Dang DC, Moukrim A (2016) Heuristic solutions for the vehicle routing problem with time windows and synchronized visits. *Optim. Lett.* 10(3):511–525.

Andersson H, Duesund JM, Fagerholt K (2011) Ship routing and scheduling with cargo coupling and synchronization constraints. *Comput. Indust. Engrg.* 61(4):1107–1116.

Baldacci R, Mingozzi A, Roberti R (2011) New route relaxation and pricing strategies for the vehicle routing problem. *Oper. Res.* 59(5):1269–1283.

Bredström D, Rönnqvist M (2008) Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *Eur. J. Oper. Res.* 191(1):19–31.

Buijs P, Vis IF, Carlo HJ (2014) Synchronization in cross-docking networks: A research classification and framework. *Eur. J. Oper. Res.* 239(3):593–608.

Cheung RK, Shi N, Powell WB, Simao HP (2008) An attribute-decision model for cross-border drayage problem. *Transportation Res. Part E: Logist. Transportation Rev.* 44(2):217–234.

Christiansen M, Nygreen B (1998) A method for solving ship routing problems with inventory constraints. *Ann. Oper. Res.* 81:357–378.

Desaulniers G, Desrosiers J, Solomon M, eds. (2005) *Column Generation* (Springer, New York).

Desaulniers G, Madsen O, Ropke S (2014) The vehicle routing problem with time windows. Toth P, Vigo D, eds. *Vehicle Routing: Problems, Methods, and Applications*, 2nd ed. MOS-SIAM Series Optim. (SIAM, Philadelphia), 119–159.

Desaulniers G, Desrosiers J, Ioachim I, Solomon M, Soumis F, Villeneuve D (1998) A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. Crainic TG, Laporte G, eds. *Fleet Management and Logistics* (Kluwer, Boston), 57–93.

Dohn A, Kolind E, Clausen J (2009) The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Comput. Oper. Res.* 36(4):1145–1157.

Dohn A, Rasmussen M, Larsen J (2011) The vehicle routing problem with time windows and temporal dependencies. *Networks* 58:273–289.

Drexl M (2007) On some generalized routing problems. Unpublished doctoral thesis, Faculty of Business and Economics, RWTH Aachen University, Aachen, Germany.

Drexl M (2012) Synchronization in vehicle routing—A survey of VRPs with multiple synchronization constraints. *Transportation Sci.* 46(3):297–316.

Drexl M (2013) Applications of the vehicle routing problem with trailers and transshipments. *Eur. J. Oper. Res.* 227(2):275–283.

Drexl M (2014) Branch-and-cut algorithms for the vehicle routing problem with trailers and transshipments. *Networks* 63(1):119–133.

Drexl M, Rieck J, Sigl T, Press B (2013) Simultaneous vehicle and crew routing and scheduling for partial- and full-load long-distance road transport. *Bus. Res.* 6(2):242–264.

Dror M (1994) Note on the complexity of the shortest path models for column generation in VRPTW. *Oper. Res.* 42(5):977–978.

Feillet D, Gendreau M, Rousseau LM (2007) New refinements for the solution of vehicle routing problems with branch and price. *INFOR* 45(4):239–256.

Hachemi NE, Gendreau M, Rousseau LM (2013) A heuristic to solve the synchronized log-truck scheduling problem. *Comput. Oper. Res.* 40(3):666–673.

Hollis B, Forbes M, Douglas B (2006) Vehicle routing and crew scheduling for metropolitan mail distribution at Australia Post. *Eur. J. Oper. Res.* 173(1):133–150.

Huber S, Geiger M (2014) Swap body vehicle routing problem: A heuristic solution approach. Gonzalez-Ramirez R, Schulte F, Voß S, Ceroni Daz J, eds. *Computational Logistics*, Lecture Notes Comput. Sci., Vol. 8760 (Springer International Publishing, Cham, Switzerland), 16–30.

Ioachim I, Gélinas S, Soumis F, Desrosiers J (1998) A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks* 31(3):193–204.

Irnich S, Desaulniers G (2005) Shortest path problems with resource constraints. Desaulniers G, Desrosiers J, Solomon M, eds. *Column Generation* (Springer, New York), 33–65.

Irnich S, Toth P, Vigo D (2014) The family of vehicle routing problems. Vigo D, Toth P, eds. *Vehicle Routing* (SIAM, Philadelphia), 1–33.

Jans R (2010) Classification of Dantzig–Wolfe reformulations for binary mixed integer programming problems. *Eur. J. Oper. Res.* 204(2):251–254.

Jepsen M, Petersen B, Spoorendonk S, Pisinger D (2008) Subset-row inequalities applied to the vehicle-routing problem with time windows. *Oper. Res.* 56(2):497–511.

Kergosien Y, Lenté C, Billaut JC, Perrin S (2013) Metaheuristic algorithms for solving two interconnected vehicle routing problems in a hospital complex. *Comput. Oper. Res.* 40(10):2508–2518.

Kergosien Y, Lenté C, Piton D, Billaut JC (2011) A tabu search heuristic for the dynamic transportation of patients between care units. *Eur. J. Oper. Res.* 214(2):442–452.

Kim BI, Koo J, Park J (2010) The combined manpower-vehicle routing problem for multi-staged services. *Expert Systems Appl.* 37(12):8424–8431.

Labadie N, Prins C, Yang Y (2014) Iterated local search for a vehicle routing problem with synchronization constraints. Vitoriano B, Pinson E, Valente F, eds. *ICORES* 2014—*Proc. 3rd Internat. Conf. Oper. Res. Enterprise Systems* (SCITEPRESS, Setúbal, Portugal), 257–263.

Lahyani R, Khemakhem M, Semet F (2015) Rich vehicle routing problems: From a taxonomy to a definition. *Eur. J. Oper. Res.* 241(1):1–14.

Laporte G (2016) Scheduling issues in vehicle routing. *Ann. Oper. Res.* 236(2):463–474.

Liberatore F, Righini G, Salani M (2011) A column generation algorithm for the vehicle routing problem with soft time windows. *4OR* 9(1):49–82.

Lübbecke M, Desrosiers J (2005) Selected topics in column generation. *Oper. Res.* 53(6):1007–1023.

Mankowska DS, Meisel F, Bierwirth C (2013) The home health care routing and scheduling problem with interdependent services. *Health Care Management Sci.* 17(1):15–30.

Meisel F, Kopfer H (2014) Synchronized routing of active and passive means of transport. *OR Spectrum* 36(2):297–322.

Morais VW, Mateus GR, Noronha TF (2014) Iterated local search heuristics for the vehicle routing problem with cross-docking. *Expert Systems Appl.* 41(16):7495–7506.

Mues C, Pickl S (2005) Transshipment and time windows in vehicle routing. *Proc. 8th Internat. Sympos. Parallel Architectures, Algorithms Networks* (IEEE, Piscataway, NJ), 113–119.

Salani M (2005) Branch-and-price algorithms for vehicle routing problems. Unpublished doctoral thesis, Faculty of Mathematical, Physical and Natural Sciences, University of Milan, Milan.

Salazar-Aguilar MA, Langevin A, Laporte G (2013) The synchronized arc and node routing problem: Application to road marking. *Comput. Oper. Res.* 40(7):1708–1715.

Smilowitz K (2006) Multi-resource routing with flexible tasks: An application in drayage operations. *IIE Trans.* 38(7):555–568.

RIGHTSLINK

Spliet R, Gabor A (2015) The time window assignment vehicle routing problem. *Transportation Sci.* 49(4):721–731 .

Tilk C, Irnich S (2017) Dynamic programming for the minimum tour duration problem. *Transportation Sci.* 51(2):549–565.

Xue Z, Zhang C, Lin WH, Miao L, Yang P (2014) A tabu search heuristic for the local container drayage problem under a new operation mode. *Transportation Res. Part E: Logist. Transportation Rev.* 62:136–150.

Zhang R, Lu JC, Wang D (2014) Container drayage problem with flexible orders and its near real-time solution strategies. *Transportation Res. Part E: Logist. Transportation Rev.* 61:235–251.

Zhang R, Yun WY, Kopfer H (2010) Heuristic-based truck scheduling for inland container transportation. *OR Spectrum* 32(3):787–808.

Zhang R, Yun WY, Kopfer H (2013) Multi-size container transportation by truck: modeling and optimization. *Flexible Services Manufacturing J.* 27(2–3):403–430.

Downloaded from informs.org by [130.64.11.153] on 21 June 2017, at 19:02 . For personal use only, all rights reserved.

RIGHTSLINK