TRANSPORTATION SCIENCE
Vol. 44, No. 4, November 2010, pp. 455–473
ISSN 0041-1655 | EISSN 1526-5447 | 10 | 4404 | 0455

informs

DOI 10.1287/trsc.1100.0328

European Driver Rules in Vehicle Routing with Time Windows

Eric Prescott-Gagnon, Guy Desaulniers
École Polytechnique de Montréal, Montréal, Québec H3C 3A7, Canada, and GERAD, Montréal, Québec H3T 2A7, Canada {eric.prescott-gagnon@gerad.ca, guy.desaulniers@gerad.ca}

Michael Drexl
Fraunhofer-Centre for Applied Research on Supply Chain Services SCS, 90411 Nuremberg, Germany, michael.drexl@scs.fraunhofer.de

Louis-Martin Rousseau
École Polytechnique de Montréal and CIRRELT, Montréal, Québec H3C 3AT, Canada, louis-martin.rousseau@cirrelt.ca

As of April 2007, the European Union has new regulations concerning driver working hours. These rules force the placement of breaks and rests into vehicle routes when consecutive driving or working time exceeds certain limits. This paper proposes a large neighborhood search method for the vehicle routing problem with time windows and driver regulations. In this method, neighborhoods are explored using a column generation heuristic that relies on a tabu search algorithm for generating new columns (routes). Checking route feasibility after inserting a customer into a route in the tabu search algorithm is not an easy task. To do so, we model all feasibility rules as resource constraints, develop a label-setting algorithm to perform this check, and show how it can be used efficiently to validate multiple customer insertions into a given existing route. We test the overall solution method on modified Solomon instances and report computational results that clearly show the efficiency of our method compared to two other existing heuristics.

Key words: vehicle routing; time windows; driver rules; large neighborhood search; heuristic column generation; resource constraints
History: Received: October 2009; revision received: March 2010; accepted: May 2010. Published online in *Articles in Advance* August 19, 2010.

1. Introduction

Vehicle routing consists of determining a set of vehicle routes to service a set of customers at minimum cost. Applications of vehicle routing are numerous and so are the problem variants (see Toth and Vigo 2002; Golden, Raghavan, and Wasil 2008). One of the first variants to appear in the literature is the vehicle routing problem with time windows (VRPTW) that can be briefly defined as follows. Given a set of customers, each with a known demand and a time window in which service must begin, as well as an unlimited set of identical vehicles with a fixed capacity and all housed in a single depot, the VRPTW consists of finding vehicle routes such that (1) each customer is serviced within its time window, (2) each route starts and ends at the depot and respects vehicle capacity, (3) the number of vehicles used is minimized as a primary objective, and (4) the total distance traveled is minimized as a secondary objective. The VRPTW has been well-studied in the literature (see Kallehauge et al. 2005; Bräysy and Gendreau 2005a, b).

As of April 2007, new regulations for driving hours have been enforced in the European Union. These rules are defined in Regulation (EC) No. 561/2006 of the European Union (2006). Furthermore, European drivers are also subject to additional regulations concerning their working hours (which include driving and servicing hours) as stipulated in Directive 2002/15/EC of the European Union (2002). All of these rules strongly restrict the feasibility of vehicle routes that last several days and are assigned to a single driver. In this case, two approaches can be used to construct vehicle routes. In a two-phase approach, routes are first built without considering driver rules and are later modified to take these rules into account. In an integrated approach, driver rules are directly addressed while vehicle routes are constructed, which is usually more complex but provides lower-cost solutions. In this paper, we propose a hybrid heuristic combining local search and mathematical programming for solving the integrated problem of building vehicle routes that respect time windows and all driver rules dictated by Regulation (EC)

No. 561/2006 and Directive 2002/15/EC (European Union 2006, 2002) applicable to routes lasting less than one week. We call this problem the VRPTW with driver rules (VRPTWDR).

The literature on vehicle routing that includes driver regulations is rather scant. Savelsbergh and Sol (1998) proposed a branch-and-price algorithm for a pick-up and delivery problem where breaks for drivers must be taken within a time interval around noon. Xu et al. (2003) tackled a pick-up and delivery problem with several complicating side constraints, including driving time restrictions that force night rests. To solve this problem, Xu et al. (2003) developed a column generation method that relies on a heuristic column generator. Bartodziej et al. (2009) designed a column generation method and three metaheuristics for solving a combined vehicle and crew scheduling problem with time windows and rest regulations. Goel and Gruhn (2006) and Goel (2009) proposed a large neighborhood search framework for solving a variant of the VRPTWDR that considers only a subset of the European regulations. The chosen rules are strict enough to ensure feasibility of the routes when all regulations are taken into account. Zäpfel and Bögl (2008) developed a two-phase algorithm for a complex VRPTW with certain driver rules. In the first phase, a vehicle routing problem is solved using a metaheuristic while, in the second phase, a personnel assignment problem is solved using a simple heuristic. Daily constraints (breaks and maximum daily driving time) are tackled by the routing algorithm. Weekly constraints (rests and maximum weekly driving time) are treated in the personnel assignment problem. In a very recent working paper, Kok et al. (2009) presented an heuristic dynamic programming algorithm that can solve the VRPTWDR (with all European regulations). Kok et al. (2009) obtained much better solutions than Goel (2009) for the same instances (considering only a subset of the driver rules) and also showed that considering complex regulations that provide additional flexibility for positioning breaks and rests can yield substantial cost savings.

This paper introduces a large neighborhood search algorithm for the VRPTWDR. The algorithm uses a column generation heuristic for exploring a neighborhood (that is, reoptimizing the current solution). In this heuristic, columns are generated by a tabu search that checks the feasibility of the driver rules using resource constraints. Computational results obtained on the instances used by Goel (2009) and Kok et al. (2009) show that our algorithm clearly outperforms their algorithms, which are the only ones proposed so far in the literature for the VRPTWDR.

This paper is structured as follows. Section 2 defines the VRPTWDR. Section 3 describes the proposed large neighborhood search algorithm while §4 details the procedure used to check the feasibility of a route. Section 5 reports the results of our computational experiments. Finally, we draw conclusions in §6.

2. Problem Statement

Given a set of identical vehicles assigned to a single depot, the VRPTW consists of computing a set of feasible routes (one per vehicle used) to deliver goods to a set $\mathcal{N}$ of customers while minimizing (1) the total number of vehicles used and (2) the total distance traveled. Each customer $i \in \mathcal{N}$ must be visited by exactly one vehicle to receive a quantity q_i of goods, and its service must begin within a prescribed time window $[a_i, b_i]$. A route starts and ends at the depot and is feasible if it respects the time windows of the visited customers and if the total quantity delivered does not exceed the vehicle capacity Q.

Denote the depot by 0 and let $\mathcal{N}_0 = \mathcal{N} \cup \{0\}$. Without loss of generality, we associate an unrestrictive time window $[a_0, b_0] = [0, H]$ with the depot, where H is the length of the planning horizon. Let t_{ij}, $i, j \in \mathcal{N}_0$, be the driving time between locations i and j and let s_i be the service time at i, $i \in \mathcal{N}_0$ ($s_0 = 0$). Furthermore, let c_{ij} be the travel distance between i and j (which is usually proportional to t_{ij}). We assume that the driving times and the travel distances satisfy the triangle inequality. In this case, customer j can be visited immediately after customer i only if $a_i + s_i + t_{ij} \le b_j$. Note that although the service at customer j must start within time window $[a_j, b_j]$, the vehicle servicing this customer can arrive earlier than a_j and wait until a_j before starting service.

The VRPTWDR extends the VRPTW with the additional constraint that every selected route must satisfy the driver regulations defined below. These regulations impose scheduling breaks and rests for the drivers along the vehicle routes. However, these breaks and rests cannot interrupt the service at a customer (customer service is not preemptive).

This problem definition relies on the assumption that every vehicle is assigned to a single driver for the whole six-day horizon, as is often the case in national and international road transportation (long distances between the customers). This allows us to directly impose the driver regulations on the vehicle routes. Furthermore, as in Goel (2009) and Kok et al. (2009), we assume that the VRPTWDR is defined over a 6-day horizon (from Monday 00:00 to Saturday 24:00), that is, $H = 8{,}640$ minutes. Indeed, in Europe, drivers are not allowed to work on Sundays, and their weekly rests (to be defined later) usually contain Sundays.

2.1. Definitions for Driver Regulations

The following definitions are needed to describe the driver regulations. Each period of time in the following statements shall be uninterrupted.

• A calendar *week* is defined by the period of time between 00:00 on Monday through 24:00 on Sunday.

• A *break* is any period of time of at least 15 minutes but less than 3 hours where a driver can dispose freely of his time.

—A *short break* is a break of at least 15 minutes but less than 45 minutes.

—A *long break* is a break of at least 45 minutes or a break of at least 30 minutes if a short break was taken since the last long break or rest.

• A *rest* is any period of time of at least three hours where a driver can dispose freely of his time.

—A *short rest* is a rest of at least three hours but less than nine hours.

—A *long rest* is a rest of at least 9 hours but less than 24 hours.

—A *regular daily rest* is either a long rest of at least 11 hours or a combination of a short rest and a long rest.

—A *reduced daily rest* is a long rest of less than 11 hours not preceded by a short rest.

—A *weekly rest* is a rest of at least 24 hours. In our case, we assume that a weekly rest is taken before the start and after the end of every route.

• The *interval driving time* is the total cumulated driving time between two long breaks or rests.

• The *daily driving time* is the total cumulated driving time between two regular or reduced rests (daily or weekly).

• The *weekly driving time* is the total cumulated driving time in a calendar week.

• The *interval working time* is the total cumulated working time between two long breaks or rests. In our case, working time is cumulated when driving or servicing a customer.

• The *weekly working time* is the total cumulated working time in a calendar week.

2.2. Regulations on Driving Time and Working Time

There are five different driving or working time periods, each having its own restrictions.

• The interval driving time must not exceed 4.5 hours.

• The daily driving time must not exceed 9 hours. It is possible to increase the daily driving time to a maximum of 10 hours but not more than twice per calendar week.

• The weekly driving time must not exceed 56 hours.

• The interval working time must not exceed six hours.

• The weekly working time must not exceed 60 hours.

These last two regulations are the only ones imposed by Directive 2002/15/EC of the European Union (2002). All of the other rules, including those stated in the next subsection, are enforced by Regulation (EC) No. 561/2006 of the European Union (2006).

Note that for the interval working time rule, Directive 2002/15/EC (European Union 2002) is less constraining than what we stated above. In fact, a working interval can be interrupted by a 30-minute break (not necessarily a long break) if the daily working time does not exceed 9 hours; otherwise, a 45-minute (long) break or a rest is needed. In any case, such a break may be subdivided into periods of at least 15 minutes each. In a context where the distances between customer locations are rather long (such as the one we consider in our experiments), the interval working time rule is often redundant with the interval driving time rule. This justifies our choice to consider a more restrictive rule on interval working time.

2.3. Regulations on Rest Periods

The following rules apply to the daily rests.

• Within 24 hours after the end of the previous daily rest or weekly rest, a driver must have taken a new daily rest.

• If the portion of the daily rest period that falls within that 24-hour period is at least 9 hours but less than 11 hours, then the daily rest in question shall be regarded as a reduced daily rest.

• The daily rest must not necessarily be over at the end of the 24-hour period as long as more than 11 (or 9) hours of daily rest fall within the 24-hour period. In other words, a daily rest must start no later than 13 hours after the end of the last one in the case of a regular daily rest or 15 hours in the case of a reduced daily rest. These values (13 and 15 hours) are referred to as the maximum daily duration and the extended maximum daily duration.

• A maximum of three reduced daily rests can be taken between two weekly rests.

The following rule applies to the weekly rests.

• There must be no more than six 24-hour periods (144 hours) between two weekly rests. This rule is, thus, directly taken into account by considering a 6-day horizon and weekly rests before and after each route.

3. Solution Method

For solving the VRPTW, Prescott-Gagnon, Desaulniers, and Rousseau (2009) introduced a large neighborhood search (LNS) heuristic that takes advantage of the power of branch and price (BP), a leading methodology for the exact solution of the VRPTW. At each iteration, the LNS heuristic chooses between different neighborhood structures to destroy parts of a current solution and uses a BP heuristic to resconstruct

these parts and potentially yields an improved solution. Compared to the best-known methods, this algorithm produced better solutions, especially in large instances where the number of vehicles used was significantly reduced.

For the VRPTWDR, we propose to use a similar LNS heuristic. In fact, the framework is identical: The same neighborhood-defining operators are applied to destroy the current solution, a BP heuristic that relies on a tabu search (TS) column generator is invoked to construct a new solution, and this TS algorithm allows two possible move types to modify a route under construction, namely, the deletion of a customer from this route and the insertion of a new customer into it. Deleting a customer from a route is easy in either the VRPTW or the VRPTWDR. However, inserting a customer into a route requires a check of the feasibility of the resulting route. This feasibility check can be done fairly easily for the VRPTW. The task is much more complex for the VRPTWDR. Hence, the algorithmic contribution of this paper is to develop an efficient heuristic procedure to check the feasibility of a route after a customer insertion. This procedure uses labels with resource components to represent partial routes and their corresponding breaks and rests as well as resource-extension functions (REFs) to propagate these labels.

Figure 1 illustrates the link between the various components of the proposed solution approach. Starting from an initial solution, an LNS heuristic alternating between a destruction phase and a reconstruction phase is applied to improve it. Reconstruction is performed using a BP heuristic that solves at each iteration a linear relaxation by column generation before fixing a route if the linear relaxation solution is fractional. In a column generation algorithm, a restricted master problem is solved at each iteration before solving a subproblem. In our method, we solve the subproblem using a TS heuristic. In this heuristic, customers are removed or inserted into a current route. The proposed insertion feasibility checker based on REFs is invoked in this TS heuristic.

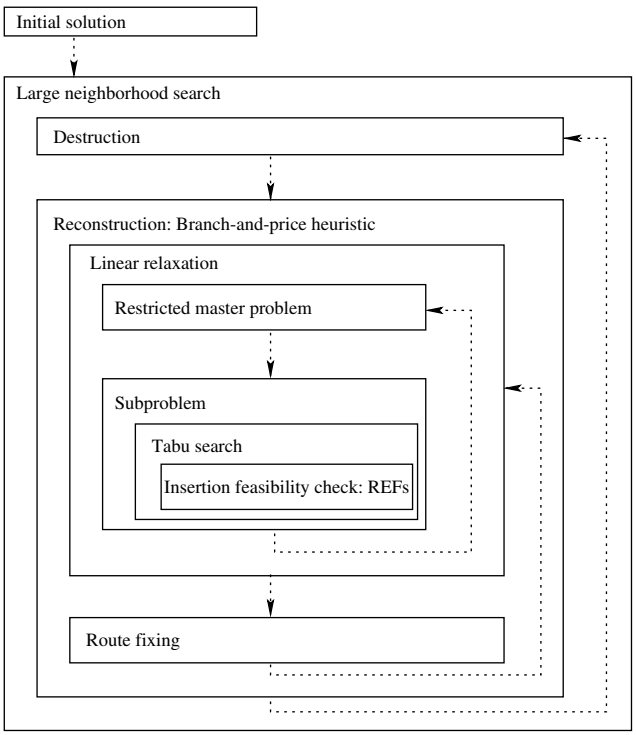

Figure 1 Algorithm Framework

In this section, we briefly describe the components of this solution method that are common with the method of Prescott-Gagnon, Desaulniers, and Rousseau (2009). The reader is referred to Prescott-Gagnon, Desaulniers, and Rousseau (2009) for further details. The insertion feasibility checker will be detailed in §4.

3.1. Large Neighborhood Search

Given an initial solution that consists of a set of routes covering all customers, an LNS heuristic is an iterative method that destroys parts of a current solution at each iteration using a neighborhood-defining operator and constructs a new solution using an optimization algorithm. In our case, a neighborhood-defining operator selects a predefined number of customers to remove (denoted M^{rem}) from the current solution, which leaves a partial solution composed of partial routes that remain fixed in the construction step. Four such operators were used: a proximity operator that selects, one at a time, customers that are related geographically and temporally; a route portion operator that is similar to the proximity operator but also removes for each selected customer some of its adjacent neighbors in its current route; a longest-detour operator that removes the customers yielding the largest distance increases for servicing them; and a time operator that simply selects customers that are serviced almost at the same time. The operator applied at each iteration to destroy the current solution is chosen using a roulette wheel procedure that favors the operators that have yielded improved solutions in the past iterations.

In the construction step, the BP heuristic described in the next subsection is executed to build a solution that contains the partial routes that were kept from the previous solution. The computed solution always becomes the current solution even when its cost is greater than the cost of the previous solution.

The LNS algorithm proceeds in two phases. The first phase aims at minimizing the total number of vehicles used. To do so, it imposes an upper bound m_{ub} on this number of vehicles while allowing to not visit the customers at the expense of a large penalty

for each unserviced customer. When the algorithm succeeds to find a solution that services all customers within a limited number of iterations (denoted $I_1^{\max}$), m_{ub} is lowered by one, and the search for a new feasible solution is started again. When it fails to do so, m_{ub} is increased by one (that is, to the number of vehicles used in the last feasible solution found), and the second phase is triggered. In this second phase, the algorithm performs a fixed number of iterations (denoted $I_2^{\max}$) to reduce the total distance traveled.

3.2. Branch-and-Price Heuristic

At each LNS iteration, a BP heuristic is used to build a new solution that preserves the fixed parts of the current solution. Such an heuristic consists of an heuristic column generation method (to compute linear relaxation solutions) embedded into an heuristic branch-and-bound method (to derive integer solutions). It relies on the following formulation of the VRPTWDR.

Let Ω be the subset of all the feasible routes that respect the fixed parts of the current solution. With each route $p \in \Omega$, associate the following parameters: c_p is its cost, and v_{ip}, $\forall\, i \in \mathcal{N}$ takes value one if customer i is serviced by route p and zero otherwise. Finally, define a binary variable θ_p for each route $p \in \Omega$ indicating whether or not route p is part of the new solution. With this notation, the VRPTWDR (restricted by the fixed route parts) can be formulated as follows:

$$\text{minimize} \quad \sum_{p\in\Omega} c_p\theta_p, \tag{1}$$

$$\text{subject to} \quad \sum_{p\in\Omega} v_{ip}\theta_p = 1, \quad \forall\, i \in \mathcal{N}, \tag{2}$$

$$\sum_{p\in\Omega} \theta_p \le m_{\text{ub}}, \tag{3}$$

$$\theta_p \text{ binary}, \quad \forall\, p \in \Omega. \tag{4}$$

The objective function (1) aims at minimizing total cost. Set-partitioning constraints (2) ensure that each customer is visited exactly once by one vehicle. Note that in the first phase of the LNS algorithm, slack variables (highly penalized in the objective function) are introduced in these constraints to allow customer uncovering. Constraint (3) imposes an upper bound m_{ub} on the number of vehicles that can be used. Recall that this upper bound varies during the first phase of the LNS algorithm and is fixed in the second phase (see §3.1). Finally, binary requirements on the variables are expressed by (4).

In a column generation context, the linear relaxation of model (1)–(4) is called the master problem. Column generation is an iterative process that solves at each iteration the master problem restricted to a subset of the variables, then called the restricted master problem (RMP). The dual solution of this RMP is then transfered to a subproblem whose objective is to generate negative reduced cost variables to be added into the current RMP. The latter is then solved again with the augmented subset of variables. When no negative reduced cost variables exist, column generation stops and the computed solution of the current RMP is also optimal for the master problem.

The column generation subproblem consists of finding a least-reduced-cost route. Let α_i, $i \in \mathcal{N}$, and μ be the dual variable values of the RMP constraints (2) and (3) at the current column generation iteration, respectively. The reduced cost $\bar{c}_p$ of a feasible route $p = i_0 - i_1 - i_2 - \cdots - i_k - i_{k+1}$ starting and ending at the depot $i_0 = i_{k+1} = 0$ and visiting k customers $i_1, i_2, \ldots, i_k$ is computed as

$$\bar{c}_p = \sum_{l=0}^{k} \bar{c}_{i_l i_{l+1}},$$

where

$$\bar{c}_{ij} = \begin{cases} c_{ij} - \alpha_i & \text{if } i \in \mathcal{N} \\ c_{ij} - \mu & \text{if } i = 0. \end{cases}$$

This subproblem can be modeled as an NP-hard resource-constrained elementary shortest-path problem (see Irnich and Desaulniers 2005). In the proposed column generation heuristic, we solve it using the TS heuristic described in the next subsection. Because this TS heuristic might not find a negative reduced-cost route when such a route exists, column generation may terminate prematurely and possibly yield a suboptimal solution for the master problem.

In order to quickly derive an integer solution, we use a heuristic branch-and-bound method that explores a single branch (that is, no backtracking is performed). After solving a linear relaxation for which a fractional solution was computed, the route variable θ_p with the largest fractional value is simply fixed at one, defining a new linear relaxation to solve (or, equivalently, a new branch-and-bound node to explore).

3.3. Tabu Search Column Generator

To solve the column generation subproblem, we use a TS heuristic that can often generate negative reduced-cost columns (routes) in fast computational times. Such a heuristic is an iterative method that starts from an initial solution and applies moves to potentially improve it. In our case, a solution is a route, and two types of moves are considered, namely, inserting a customer into the route and removing a customer from the route. At each iteration, the move creating the best solution is chosen even if the objective value deteriorates. To avoid cycling, a tabu list is used in order to forbid recent moves to be reversed for a given number of iterations, which allows the search to escape from local minima.

In the TS algorithm, the sequences of customers (partial routes) fixed in the destruction phase of the LNS algorithm are treated as aggregated customers, meaning that new customers cannot be inserted within a sequence. Also, the search space is limited to feasible routes. Thus, for each insertion move, route feasibility is checked using the procedure described in §4. No feasibility check needs to be performed for customer removal moves (under the assumption that the travel times respect the triangle inequality).

To diversify the search, the TS heuristic is started multiple times from a set of different initial solutions and limited to a maximum number of iterations for each initial solution. In the first phase of the LNS method, the routes associated with the positive-valued variables in the current RMP solution form the set of initial solutions. In the second phase, this set contains the routes associated with all basic variables. In both cases, all initial solutions have a reduced cost of zero and are thus very good initial solutions because we are looking for negative reduced-cost solutions.

4. Route Feasibility Check

In the TS column generator presented earlier, one needs to validate that each customer insertion yields a route that can be operated by a vehicle and its driver. This validation requires, among other requirements, the placement of breaks and rest periods along the route in accordance with the regulations described in §2.

In this section, we present a labeling algorithm that checks route feasibility and shows how it can be used in conjunction with a filtering algorithm to efficiently assess the feasibility of multiple insertions into a given feasible route. This algorithm relies on resource constraints to model all route feasibility rules (see Irnich and Desaulniers 2005). It uses a total of 15 resources. However, to simplify its description, we consider in this section only a subset of the feasibility rules and options that requires eight resources. These rules are the time windows, vehicle capacity, maximum interval driving time, maximum daily driving time, and maximum daily duration. Furthermore, among all options (short breaks, short rests, reduced rests, and extended maximum daily driving time), we consider only the possibility of using short breaks. How to deal with the other rules and options is exposed in Appendix A.

4.1. Labeling Algorithm

Let $p = i_0 - i_1 - i_2 - \cdots - i_k - i_{k+1}$ be a route starting and ending at the depot $i_0 = i_{k+1} = 0$ and visiting the k customers $i_1, i_2, \ldots, i_k$ (including the inserted customer). Associate a path with this route where i_l, $l \in \{0, 1, \ldots, k+1\}$ are the vertices of this path (two different vertices for the depot) and (i_l, i_{l+1}), $l \in \{0, 1, \ldots, k\}$ its arcs. Denote by $\mathcal{V}_p$ and $\mathcal{A}_p$ these vertex and arc sets, respectively. To check the feasibility of such a path (route), we use a labeling algorithm that starts with an initial label at vertex 0 and propagates this label forward along the arcs of this path to create new labels. A label represents a feasible partial path (which implicitly includes breaks and rests on the arcs) originating from vertex 0 and is given by a resource vector. Such a vector E_i is associated with the destination vertex $i \in \mathcal{V}_p$ of the partial path and contains eight components, one for each resource (all times are expressed in minutes); we have:

$T_i \in [a_i, b_i]$: earliest service start time at vertex i.

$L_i \in [0, Q]$: accumulated demand (load) up to vertex i (including vertex i).

$T_i^{\text{drive, int}} \in [0, 270]$: interval driving time up to vertex i.

$n_i^{sb} \in [0, 1]$: number of short breaks taken before vertex i since the last long break or rest.

$T_i^{\text{drive, day}} \in [0, 540]$: daily driving time up to vertex i.

$LT_i \in [a_i, b_i]$: latest start-of-service time at vertex i that does not yield unnecessary waiting or, equivalently, earliest start-of-service time if the current day starts as late as possible. It is used to compute the daily duration.

$XT_i \in [0, \infty]$: extended latest start-of-service time at vertex i if there were no time windows at this vertex. It is also used to compute the daily duration.

$D_i \in [0, 780]$: current daily duration up to vertex i.

Thus, $E_i = (T_i, L_i, T_i^{\text{drive, int}}, n_i^{sb}, T_i^{\text{drive, day}}, LT_i, XT_i, D_i)$. Let $\mathcal{R}$ be the set of resources. Each resource $r \in \mathcal{R}$ is constrained by a resource window $[a_i^r, b_i^r]$ at each vertex $i \in \mathcal{V}_p$. These windows were presented earlier. A label represents a feasible partial path if the value of each of its resource components falls into its corresponding resource window. If this is the case, we say that the label (or the resource vector) is feasible.

The pseudocode of the label-setting algorithm is given in Algorithm 1, where $\mathcal{E}(i)$ denotes the current subset of untreated labels at vertex $i \in \mathcal{N}_p$. The algorithm starts by defining an initial label E_{i_0} associated with vertex i_0 and initializing the sets $\mathcal{E}(i_l)$ for all $l \in \{0, 1, \ldots, k+1\}$. Then, in the main loop (steps 3–7), it selects at each iteration an untreated label and extends it to generate new labels. The label extension is performed using the procedure described in §4.2 and may consider various possibilities (no break or rest, short break, long break, daily rest, and combinations of these) in addition to traveling from i_0 to i_1. This loop stops when one feasible label is created at vertex i_{k+1} or when all existing labels have been treated. In the hope of meeting the former stopping criterion as soon as possible, the labels are treated in a depth-first order (steps 4–5).

Algorithm 1 (Exact route feasibility check algorithm)

Require: Route (path) $p = i_0 - i_1 - i_2 - \cdots - i_k - i_{k+1}$
1: $E_{i_0} \leftarrow (0, 0, 0, 0, 0, H, H, 0)$
2: $\mathscr{E}(i_0) \leftarrow \{E_{i_0}\}$, $\mathscr{E}(i_l) \leftarrow \varnothing, \forall\, l \in \{1, 2, \ldots, k+1\}$
3: **While** $\mathscr{E}(i_{k+1}) = \varnothing$ and $\bigcup_{l=0}^{k} \mathscr{E}(i_l) \neq \varnothing$, **do**
4: $\quad l^* \leftarrow \max_{l \in \{0,1,\ldots,k\}} \{l \mid \mathscr{E}(i_l) \neq \varnothing\}$
5: $\quad$ Select a label $E_{i_{l^*}}$ in $\mathscr{E}(i_{l^*})$
6: $\quad \mathscr{E}(i_{l^*+1}) \leftarrow \mathscr{E}(i_{l^*+1}) \cup \mathit{ExtendLabel}(E_{i_{l^*}}, i_{l^*}, i_{l^*+1})$
7: $\quad \mathscr{E}(i_{l^*}) \leftarrow \mathscr{E}(i_{l^*}) \backslash \{E_{i_{l^*}}\}$
8: **if** $\mathscr{E}(i_{k+1}) \neq \varnothing$, **then**
9: $\quad$ Return p is feasible
10: **else**
11: $\quad$ Return p is infeasible

4.2. Label Extension

At each iteration of Algorithm 1, a label E_i is extended along an arc $(i, j) \in \mathscr{A}_p$, where $i = i_{l^*}$ and $j = i_{l^*+1}$. This label represents a partial path ending at vertex i that we want to extend along the arc (i, j). This extension includes the time of service at vertex i and the travel time between vertices i and j. It may also include breaks and rests. In fact, several combinations of breaks and rests may yield a feasible label at vertex j.

To generate all feasible labels from E_i, we adopt the following strategy. First, we extend E_i, assuming that no break or rest will be taken along arc (i, j). Then, we go over all possible options involving the placement of breaks and rests along this arc. These options are to insert a short break, insert a long break, and insert a regular long rest.

Using resource-extension functions, the first extension yields a new resource vector E_j that might not be feasible at vertex j with respect to certain driver rules. This resource vector is then modified by other resource-extension functions that are specific to each selected option. All these resource-extension functions are described next.

4.2.1. Resource-Extension Functions. Given a feasible label $E_i = (T_i,\ L_i,\ T_i^{\text{drive, int}},\ n_i^{sb},\ T_i^{\text{drive, day}},\ LT_i,\ XT_i,\ D_i)$ at vertex i, it is first extended along the arc $(i, j) \in \mathscr{A}_p$, assuming that only the service at vertex i and the traveling between i and j are realized along this arc. This extension is performed using the following resource-extension functions to generate a new resource vector $E_j = (T_j,\ L_j,\ T_j^{\text{drive, int}},\ n_j^{sb},\ T_j^{\text{drive, day}},\ LT_j,\ XT_j,\ D_j)$.

$$T_j = T_i + s_i + t_{ij}, \tag{5}$$

$$L_j = L_i + q_j, \tag{6}$$

$$T_j^{\text{drive, int}} = T_i^{\text{drive, int}} + t_{ij}, \tag{7}$$

$$n_j^{sb} = n_i^{sb}, \tag{8}$$

$$T_j^{\text{drive, day}} = T_i^{\text{drive, day}} + t_{ij}, \tag{9}$$

$$XT_j = LT_i + s_i + t_{ij}, \tag{10}$$

$$LT_j = \max\{\min\{XT_j, b_j\}, a_j\}, \tag{11}$$

$$D_j = D_i + LT_j - LT_i + \max\{XT_j - b_j, 0\}. \tag{12}$$

These resource-extension functions are denoted $\text{REF}_{\mathit{first}}(\cdot, \cdot, \cdot)$, that is, $E_j = \text{REF}_{\mathit{first}}(E_i, i, j)$. The component T_j as computed by (5) indicates the earliest arrival time at vertex j, which can be less than a_j. When creating a label in which this component must represent the earliest start-of-service time at vertex j, we use the function $\mathit{AdjTime}(E_j, j)$, which duplicates all components except the T_j component, which is replaced by $\max\{T_j, a_j\}$.

The computation of the current daily duration D_j relies on the components XT_j and LT_j. It assumes that the current day starts as late as possible (that is, the preceding rest has been extended to a maximum). Under this assumption, LT_j provides the earliest start-of-service time at vertex j. To compute LT_j using (11), one needs to compute first what would be this time if there were no time window at vertex j (given by XT_j). If $XT_j < a_j$, then waiting $a_j - XT_j$ minutes before starting service at vertex j is unavoidable and LT_j is set to a_j. If $XT_j \in [a_j, b_j]$, then $LT_j = XT_j$. If $XT_j > b_j$, then the current day must begin earlier (by shortening the preceding daily rest by $XT_j - b_j$ minutes, if possible), and LT_j is set to b_j. Figure 2 illustrates these three cases. In the first case, the start time of the day does not need to be changed and the daily duration D_j can be computed as $D_i + s_i + t_{ij} + a_j - XT_j = D_i + LT_j - LT_i$. In the second case, the start time of the day also remains the same, there is no additional waiting, and D_j computes as $D_i + s_i + t_{ij} = D_i + LT_j - LT_i$. Finally, in the third case, the day start time is pushed backward by $XT_j - b_j$ minutes, and D_j is given by $D_i + s_i + t_{ij} + XT_j - b_j = D_i + LT_j - LT_i + \max\{XT_j - b_j, 0\}$. Note that shifting backward by $XT_j - b_j$ the start time of a day is always feasible unless $T_j > b_j$, in which case the label is declared infeasible and discarded.

A computed resource vector E_j might not be feasible with regard to the driver regulations. However, if $T_j < b_j$ and $L_j \leq Q$, adding breaks and rests along the arc (i, j) might yield a feasible label. Furthermore, even if all driver regulations are met, adding a break or a rest might be advantageous when unavoidable waiting occurs. In these cases, three options can be considered: inserting a short break (sb), a long break (lb), or a regular long rest (lr). Given a resource vector $E_j = (T_j,\ L_j,\ T_j^{\text{drive, int}},\ n_j^{sb},\ T_j^{\text{drive, day}},\ LT_j,\ XT_j, D_j)$, the subset of options applicable along an arc $(i, j) \in \mathscr{A}_p$ is denoted $\Delta(E_j, j) \subseteq \{sb, lb, lr\}$. Applying such an

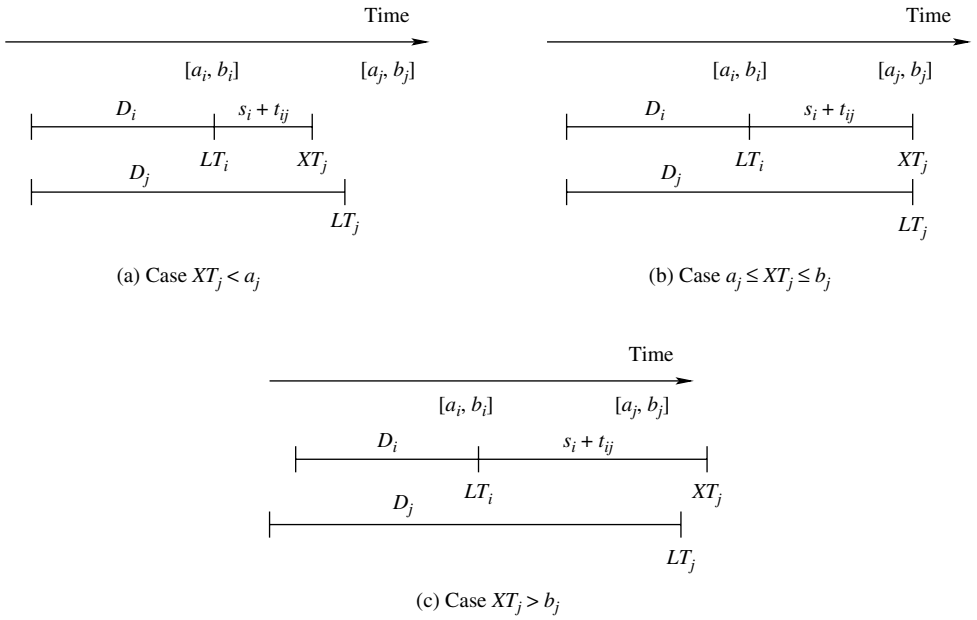

Figure 2 Three Cases for the Computation of the Current Day Duration

option generates a new resource vector $\overline{E}_j = (\overline{T}_j, \overline{L}_j, \overline{T}_j^{\text{drive, int}}, \overline{n}_j^{sb}, \overline{T}_j^{\text{drive, day}}, \overline{LT}_j, \overline{XT}_j, \overline{D}_j)$ that is obtained using resource-extension functions specific to this option. Next, we describe these resource-extension functions for each option, and we also specify under which conditions an option belongs to $\Delta(E_j, j)$.

Short break. Inserting a 15-minute short break along arc (i, j) only makes sense when there is unavoidable waiting at vertex j that does not last enough to insert a 45-minute long break ($0 < a_j - XT_j < 45$). Furthermore, it is feasible only if $n_j^{sb} = 0$. Finally, it is not dominated by the insertion of a long break if the maximum interval driving time is not exceeded ($T_j^{\text{drive, int}} < 270$) or by the insertion of a rest if both the maximum daily driving time and the maximum daily duration are not exceeded ($T_j^{\text{drive, day}} < 540$ and $D_j < 780$). Thus, $sb \in \Delta(E_j, j)$ if all of these conditions are satisfied. Such a break can start at any time between the end of the last break or rest on the arc (i, j) and the beginning of the unavoidable waiting period. When selecting this option, the resource vector E_j is updated using the following resource-extension functions to create a new resource vector $\overline{E}_j$:

$$\overline{T}_j = T_j + 15, \tag{13}$$

$$\overline{n}_j^{sb} = 1, \tag{14}$$

$$\overline{XT}_j = XT_j + 15, \tag{15}$$

$$\overline{LT}_j = \max\{\min\{\overline{XT}_j, b_j\}, a_j\}, \tag{16}$$

$$\overline{D}_j = D_j + \max\{15 - (a_j - XT_j), 0\}, \tag{17}$$

$$\overline{L}_j = L_j, \overline{T}_j^{\text{drive, int}} = T_j^{\text{drive, int}}, \overline{T}_j^{\text{drive, day}} = T_j^{\text{drive, day}}. \tag{18}$$

We denote these resource-extension functions by $\text{REF}_{sb}(\cdot, \cdot)$, that is, $\overline{E}_j = \text{REF}_{sb}(E_j, j)$. Note that breaks and rests are applied as resource-component modifications. Therefore, there is no trace of the breaks and rests used unless they are kept in a dedicated memory structure.

Long break. A long break can be inserted along arc (i, j) if the maximum interval driving time is exceeded or if there is unavoidable waiting time (otherwise, the insertion can always be postponed to the next arc). It is, however, dominated by a rest if the maximum daily driving time or the maximum daily duration is reached before the maximum interval driving time while driving. Hence, $lb \in \Delta(E_j, j)$ if $\max\{T_j^{\text{drive, int}}, a_j - XT_j\} > 0$ and $T_j^{\text{drive, int}} - 270 > \max\{T_j^{\text{drive, day}} - 540, D_j - \max\{a_j - XT_j, 0\} - 780\}$. If $n_j^{sb} = 1$, the duration k_j of the break is 30 minutes; otherwise, $k_j = 45$ minutes. Such a break starts at the first point in time when the interval driving time reaches 270 minutes or when the unavoidable waiting period starts. The resource-extension functions for the insertion of a long break are as follows:

$$\overline{T}_j = T_j + k_j, \tag{19}$$

$$\overline{T}_j^{\text{drive, int}} = \max\{T_j^{\text{drive, int}} - 270, 0\}, \tag{20}$$

$$\overline{n}_j^{sb} = 0, \tag{21}$$

$$\overline{XT}_j = XT_j + k_j, \tag{22}$$

$$\overline{LT}_j = \max\{\min\{\overline{XT}_j, b_j\}, a_j\}, \tag{23}$$

$$\overline{D}_j = D_j + \max\{k_j - \max\{a_j - XT_j, 0\}, 0\}, \tag{24}$$

$$\overline{L}_j = L_j, \overline{T}_j^{\text{drive, day}} = T_j^{\text{drive, day}}. \tag{25}$$

We denote these resource-extension functions by $\mathrm{REF}_{lb}(\cdot,\cdot)$, that is, $\overline{E}_j = \mathrm{REF}_{lb}(E_j, j)$.

Long rest. A 660-minute long rest can be inserted at the time when the maximum interval driving time, the maximum daily driving time, or the maximum daily duration is exceeded. It can also be inserted as the last activity along arc (i, j) if the maximum daily duration is to be reached during the service at vertex j or during an unavoidable waiting period at vertex j. If taken, the long rest starts at the earliest of these times, denoted h. There is only one restriction on the insertion of a long rest along arc (i, j): It cannot occur during the service at vertex i. Therefore, $lr \in \Delta(E_j, j)$ if h does not fall during the service at vertex i. (Note that additional notation is needed to express this condition formally. This notation has been omitted for reasons of conciseness and clarity.) The resource-extension functions for the insertion of a long rest are as follows:

$$\overline{T}_j = T_j + 660, \tag{26}$$

$$\overline{L}_j = L_j, \tag{27}$$

$$\overline{T}_j^{\,\mathrm{drive,\,int}} = \max\{T_j^{\,\mathrm{drive,\,int}} - 270,\ T_j^{\,\mathrm{drive,\,day}} - 540,\ D_j - \max\{a_j - XT_j, 0\} - 780,\ 0\}, \tag{28}$$

$$\overline{n}_j^{sb} = 0, \tag{29}$$

$$\overline{T}_j^{\,\mathrm{drive,\,day}} = \overline{T}_j^{\,\mathrm{drive,\,int}}, \tag{30}$$

$$\overline{XT}_j = XT_j + 660, \tag{31}$$

$$\overline{LT}_j = b_j, \tag{32}$$

$$\overline{D}_j = \overline{T}_j^{\,\mathrm{drive,\,int}}. \tag{33}$$

In the maximum function of (28), the first three terms indicate the driving time left after the rest if it was taken at the time when the maximum interval driving time, the maximum daily driving time, or the maximum daily duration (offset by the unavoidable waiting time, if any) is exceeded, respectively. When at least one of these maxima is reached before the end of driving, the rest is taken as soon as one of them is reached, which yields the largest driving time left after the rest. This remaining driving time is then scheduled on the day starting after the rest and is thus used to initialize the current daily driving time and the current daily duration components in (30) and (33), respectively. In (31), the extended latest start-of-service time is increased by the duration of a long rest and is used in subsequent extensions along arc (i, j) to compute the remaining unavoidable waiting time, if any. On the other hand, the latest start-of-service time $\overline{LT}_j$ is set to b_j in (32), as service at the first customer visited in a day can always start at the upper bound of this customer time window.

The resource-extension functions (26)–(33) are denoted $\mathrm{REF}_{lr}(\cdot,\cdot)$, that is, $\overline{E}_j = \mathrm{REF}_{lr}(E_j, j)$.

4.2.2. Label-Extension Function. The pseudocode of the label-extension function *ExtendLabel*$(\cdot,\cdot,\cdot)$ used in step 6 of Algorithm 1 is given in Algorithm 2. Taking a feasible label E_i associated with vertex i as input, this function relies on the resource-extension functions presented earlier to generate feasible labels at vertex j if some exist. This function starts by extending E_i using the resource-extension functions $\mathrm{REF}_{first}(E_i, i, j)$ to yield a resource vector E_j. This extension corresponds to adding the service at vertex i and the traveling time between i and j. The time-adjusted vector *AdjTime*(E_j, j) is then tested for feasibility in step 3. In step 5, E_j is tested for local extendability: A label $E_j = (T_j,\ L_j,\ T_j^{\,\mathrm{drive,\,int}},\ n_j^{sb},\ T_j^{\,\mathrm{drive,\,day}},\ LT_j,\ XT_j,\ D_j)$ is said to be extendable if $T_j \le b_j$, $L_j \le Q$, and $\max\{a_j - XT_j,\ T_j^{\,\mathrm{drive,\,int}} - 270,\ T_j^{\,\mathrm{drive,\,day}} - 540,\ D_j + s_j - 780\} > 0$. (That is, one of the following conditions must hold: There is unavoidable waiting at vertex j, the maximum interval driving time is exceeded, the maximum daily driving time is exceeded, or the maximum daily duration is exceeded or will exceed during service at vertex j. If E_j is extendable, the recursive function *RecursiveExtension*(E_j, j) is applied in step 6 to insert breaks and rests along arc (i, j).

The pseudocode of the *RecursiveExtension*(E_j, j) function is given in Algorithm 3. This function extends E_j with each valid break and rest-insertion option (loop starting in step 2) to yield a resource vector $\overline{E}_j$ for each of these options. Each time-adjusted vector *AdjTime*$(\overline{E}_j, j)$ is then checked for feasibility (step 4), and each vector $\overline{E}_j$ is checked for extendability (step 6). Each extendable vector is then extended recursively in step 7.

Algorithm 2 (Function *ExtendLabel*(E_i, i, j))

Require: A label E_i, a vertex i and a vertex j such that $E_i \in \mathscr{E}(i)$ and $(i, j) \in \mathscr{A}_p$

1: $\mathscr{E} \leftarrow \varnothing$
2: $E_j \leftarrow \mathrm{REF}_{first}(E_i, i, j)$
3: **if** *AdjTime*(E_j, j) is feasible at vertex j, **then**
4: $\quad \mathscr{E} \leftarrow \{\textit{AdjTime}(E_j, j)\}$
5: **if** E_j is extendable, **then**
6: $\quad \mathscr{E} \leftarrow \mathscr{E} \cup \textit{RecursiveExtension}(E_j, j)$
7: Return $\mathscr{E}$

Algorithm 3 (Function *RecursiveExtension*(E_j, j))

Require: A resource vector E_j and a vertex j such that E_j is associated with j.

1: $\mathscr{E} \leftarrow \varnothing$
2: **for all** $o \in \Delta(E_j, j)$, **do**
3: $\quad \overline{E}_j \leftarrow \mathrm{REF}_o(E_j, j)$
4: **if** *AdjTime*$(\overline{E}_j, j)$ is feasible at vertex j, **then**
5: $\quad \mathscr{E} \leftarrow \mathscr{E} \cup \{\textit{AdjTime}(\overline{E}_j, j)\}$
6: **if** $\overline{E}_j$ is extendable, **then**
7: $\quad \mathscr{E} \leftarrow \mathscr{E} \cup \textit{RecursiveExtension}(\overline{E}_j, j)$
8: Return $\mathscr{E}$

Table 1 Example of Resource Vectors Generated by the Label-Extension Function

	E_i	$E_1 = \text{REF}_{first}(E_i)$	$E_2 = \text{REF}_{lb}(E_1)$	$E_3 = \text{REF}_{lr}(E_2)$	$E_4 = \text{REF}_{lr}(E_1)$
T	1,300	1,660	1,705	2,365	2,320
L	50	70	70	70	70
$T^{\text{drive, int}}$	150	450	180	80	180
n^{sb}	0	0	0	0	0
$T^{\text{drive, day}}$	320	620	620	80	180
XT	1,580	1,940	1,985	2,645	2,600
LT	1,580	1,940	1,985	2,800	2,800
D	400	760	805	80	180
TA feas.		No	No	Yes	Yes
Extend.		Yes	Yes	No	No
Options		*lb, lr*	*lr*		

To illustrate the application of the label-extension function $ExtendLabel(E_i, i, j)$, consider the following example that extends a feasible label E_i along an arc $(i, j) \in \mathcal{A}_p$. In this example, $[a_i, b_i] = [1{,}300, 1{,}600]$, $[a_j, b_j] = [1{,}700, 2{,}800]$, $s_i = 60$, $s_j = 90$, $t_{ij} = 300$, $q_j = 20$, and $Q = 200$. The values of the components of E_i are specified in Table 1, together with the values of the components of the generated resource vectors. In Table 1, the first and second lines provide a resource-vector identifier and the resource-extension functions used to derive this vector, respectively. Note that the arguments i and j in those functions have been omitted to reduce the table width. Lines 1 to 8 indicate the component values. For each generated resource vector, lines 9 to 11 specify if the corresponding time-adjusted resource vector is feasible (*TA feas.*), if the vector is extendable (*extend.*) and, if this is the case, the subset of valid options. Table 2 details the sequence of instructions executed in Algorithms 2 and 3 to yield vectors E_1 to E_4.

For this example, the label-extension function produces two resource vectors E_3 and E_4 (which correspond to their time-adjusted vectors) that are feasible labels at vertex j. E_3 was generated by adding a long break and a long rest along arc (i, j) whereas E_4 was obtained by adding only a long rest. Service at vertex j can start earlier if E_4 (instead of E_3) is further extended along the arc originating at j. On the other hand, a break or a rest will be needed earlier in this case.

The complexity of the exact algorithm for checking route feasibility is thus exponential in the number of possible options that can be inserted along every arc of a path. This complexity makes the application of the exact algorithm impractical in the TS column generator (see §3.3), where the algorithm would need to be invoked for every customer insertion move. This algorithm can, however, be transformed into a heuristic as follows. The set of all resource vectors (including the feasible labels and the intermediate resource vectors) generated during the exact algorithm can be seen as the vertex set of an oriented tree in which an arc (i, j) indicates that resource vector j was generated from resource vector i. The proposed heuristic algorithm also explores this tree using a depth-first search strategy but limits to a predefined maximum number M^{back} the number of backtracks that it can perform and stops as soon as a feasible placement of breaks and rests is found for the whole route.

In the following, we propose a fast approximate route feasibility check that can be used when feasibility needs to be checked for several individual customer insertions into the same route. This check relies, in part, on the heuristic version of the labeling algorithm (Algorithm 2).

Table 2 Application of Algorithms 2 and 3 Yielding the Resource Vectors of Table 1

(Algorithm, lines)	Description
(2, 2)	Starting from the feasible label E_i, the function $ExtendLabel(E_i, i, j)$ uses the extension functions $\text{REF}_{first}(E_i, i, j)$ to generate a first resource vector E_1.
(2, 3)	Because $T^{\text{drive, int}} = 450 > 270$ in E_1, $AdjTime(E_1, j)$ is not feasible and is thus not added to $\mathcal{E}$.
(2, 5)	E_1 is extendable because $T = 1{,}660 \le 2{,}800$, $L = 70 \le 200$ and $\max\{a_j - XT, T^{\text{drive, int}} - 270, T^{\text{drive, day}} - 540, D + s_j - 780\} = 180 > 0$. Options *lb* and *lr* are possible because $T^{\text{drive, int}} > 270$ and $T^{\text{drive, day}} > 540$.
(2, 6)	For extending E_1, function $RecursiveExtension(E_1, j)$ is called.
(3, 3)	Vector E_1 is extended for option *lb* using $\text{REF}_{lb}(E_1, j)$, yielding vector E_2.
(3,4)	Because $T^{\text{drive, day}} = 620 > 540$ in E_2, $AdjTime(E_2, j)$ is not feasible.
(3,6)	E_2 is extendable because $T = 1{,}705 \le 2{,}800$, $L = 70 \le 200$ and $\max\{a_j - XT, T^{\text{drive, int}} - 270, T^{\text{drive, day}} - 540, D + s_j - 780\} = 80 > 0$. Option *lr* is possible because $T^{\text{drive, day}} > 540$.
(3, 7)	Function $RecursiveExtension(E_2, j)$ is called for extending E_2.
(3, 3)	Vector E_2 is extended for option *lr* using $\text{REF}_{lr}(E_2, j)$, yielding vector E_3.
(3, 4–5)	Label $AdjTime(E_3, j)$ is feasible and added to $\mathcal{E}$.
(3, 6)	Because $\max\{a_j - XT, T^{\text{drive, int}} - 270, T^{\text{drive, day}} - 540, D + s_j - 780\} \not> 0$, E_3 is not extendable. The function returns $\mathcal{E}$ (no more options for E_2).
(3, 3)	Vector E_1 is extended for option *lr* using $\text{REF}_{lr}(E_1, j)$, yielding vector E_4.
(3, 4–5)	Label $AdjTime(E_4, j)$ is feasible and added to $\mathcal{E}$.
(3, 6)	Because $\max\{a_j - XT, T^{\text{drive, int}} - 270, T^{\text{drive, day}} - 540, D + s_j - 780\} \not> 0$, E_4 is not extendable. The function returns $\mathcal{E}$ (no more options for E_1).

4.3. Approximate Feasibility Check for Multiple Insertions

At each iteration, the TS heuristic described in §3.3 evaluates a certain number of individual customer insertions into the current solution (a feasible path $p = i_0 - i_1 - i_2 - \cdots - i_k - i_{k+1}$). Each potential insertion results in a new route that must be checked for feasibility. In this subsection, we propose an approximate checking procedure that can be used to identify rapidly certain infeasible routes obtained by a customer insertion. This strategy works in two steps. First, a preprocessing step that computes lower and upper bounds on certain resource values at each vertex of path p is performed once for this path. Then, in a second step, an inexpensive check is executed for each customer insertion to verify if these bounds can be respected by this insertion.

4.3.1. Resource Bound Tightening. This preprocessing step relies on the assumption that a low resource value in a label at a given vertex always offers more flexibility than a higher resource value for further extensions. This is true, for instance, for the load resource. Indeed, to respect the vehicle capacity (that is, the load resource upper bound at every vertex), a low resource value is preferable over a higher resource value. As stated in Irnich and Desaulniers (2005), this property holds for every resource whose resource-extension functions are nondecreasing. It is, however, not the case for the current daily duration resource, because its future value also depends on how much the start of the current day can be shifted backward. It is also not the case for the earliest start-of-service resource whose extension functions depend on the number of short breaks. Indeed, the duration of a long break, which influences the start-of-service time resource, depends on the previous use or not of a short break. Because we want to compute only lower and upper bounds, we can, however, forbid the placement of short breaks for these computations and assume that all long breaks last 30 minutes. With these assumptions, the (modified) extension functions for the start-of-service time resource become nondecreasing. Consequently, the procedure involves only the following resources: earliest start-of-service time, load, interval driving time, and daily driving time. The set of these resources is denoted $\bar{R}$.

Let $\mathrm{lb}_{i_l}(r)$ and $\mathrm{ub}_{i_l}(r)$ be the lower and upper bounds computed for resource $r \in \bar{R}$ at vertex i_l, $l \in \{0, 1, \ldots, k+1\}$. To compute these bounds, we use two labeling algorithms where a label E contains one component for each resource $r \in \bar{R}$, denoted $E(r)$. The first algorithm is a forward labeling algorithm that starts from a label at vertex i_0 and uses the resource-extension functions presented in §4.2.1 (except that the duration of all long breaks is 30 minutes as mentioned earlier) and the label extension function $ExtendLabel(\cdot,\cdot,\cdot)$ (see Algorithm 2) without the short break option to create forward labels at vertices i_l, $l \in \{1, 2, \ldots, k+1\}$. The second algorithm is a backward labeling algorithm that starts from a label at vertex i_{k+1} and uses backward resource-extension functions and a backward label-extension function to compute backward labels at vertices i_l, $l \in \{0, 1, 2, \ldots, k\}$. Such reverse functions can easily be determined (see Irnich 2008). For instance, for the earliest start-of-service time resource and the load resource, the backward extension functions along an arc $(i, j) \in \mathcal{A}_p$ (the counterparts of (5)–(6)) are

$$T_i = T_j - s_i - t_{ij}, \tag{34}$$

$$L_i = L_j - q_i, \tag{35}$$

with $T_{i_{k+1}} = b_{i_{k+1}} = H$ and $L_{i_{k+1}} = Q$. For the sake of conciseness, all the other backward resource-extension functions are not stated here. The sets of forward and backward labels generated at a vertex i_l, $l \in \{0, 1, 2, \ldots, k+1\}$, are denoted $\mathscr{E}^{fw}_{i_l}$ and $\mathscr{E}^{bw}_{i_l}$, respectively.

The pseudocode of the preprocessing algorithm is given in Algorithm 4. In step 1, only the lower bounds $\mathrm{lb}_{i_0}(r)$ at the source vertex i_0 of path p are initialized. No initial lower bounds are required for the subsequent vertices along p. Step 2 initializes all the upper bounds $\mathrm{ub}_{i_l}(r)$. In steps 5–7, the forward labeling algorithm is used to compute resource value lower bounds. At each vertex i_l, a single lower bound label lb_{i_l} is obtained by first extending forward in step 6 the single label $\mathrm{lb}_{i_{l-1}}$ to generate the forward label set $\mathscr{E}^{fw}_{i_l}$ and then computing in step 7 the minimal value $E(r)$ over all labels $E \in \mathscr{E}^{fw}_{i_l}$ for each resource $r \in \bar{R}$. Symmetrically, backward labeling and upper bound computation are performed in steps 8–10. Note that not all feasible labels E are accepted in steps 6 and 9. To be accepted, they must respect the conditions $E(r) \le \mathrm{ub}_{i_l}(r), \forall\, r \in \bar{R}$ for a forward label and the conditions $E(r) \ge \mathrm{lb}_{i_l}(r), \forall\, r \in \bar{R}$ for a backward label. These conditions ensure that the computed label can be feasibly extended up to vertex i_{k+1} or vertex i_0 along path p. For instance, assume that at a given vertex $i_{\bar{l}}$ along p, $[a^T_{i_{\bar{l}}}, b^T_{i_{\bar{l}}}] = [1{,}800, 2{,}100]$ and $\mathrm{ub}_{i_{\bar{l}}}(T) = 2{,}000$, where $r = T$ denotes the earliest start-of-service time resource. The upper bound $\mathrm{ub}_{i_{\bar{l}}}(T)$ indicates that to reach the sink vertex i_{k+1} from vertex $i_{\bar{l}}$ along p, the service at vertex $i_{\bar{l}}$ cannot start later than 2,000. Now, the forward labeling algorithm can generate a feasible label E with $E(T) = 2{,}050$. Even if it is feasible, this label cannot yield a feasible path up to i_{k+1}, and it would not be included in set $\mathscr{E}^{fw}_{i_{\bar{l}}}$. Because the upper bounds are initially set to the resource window upper bounds in step 2, the first execution of

the backward labeling algorithm in steps 9–10 can often reduce these upper bounds. In this case, it can be advantageous to recompute the lower bounds, taking into account these updated upper bounds. In fact, as soon as one set of bounds change, the other set can be recomputed to yield tighter bounds. Consequently, steps 4–10 are embedded into a repeat loop that stops either when none of the lower and upper bounds changed in the last iteration or when a maximum number of iterations is reached. Preliminary tests showed that a maximum of two iterations is sufficient to obtain good quality bounds. Note that each iteration starts by emptying the sets $\mathscr{E}_{i_l}^{fw}$ and $\mathscr{E}_{i_l}^{bw}$ (step 4).

Algorithm 4 (Computing lower and upper bounds for resource values along a given route)

Require: Route (path) $p = i_0 - i_1 - i_2 - \cdots - i_k - i_{k+1}$

1: $\text{lb}_{i_0}(r) \leftarrow a_{i_0}^r, \ \forall r \in \bar{R}$
2: $\text{ub}_{i_l}(r) \leftarrow b_{i_l}^r, \ \forall l \in \{0, 1, \ldots, k+1\}, r \in \bar{R}$
3: **repeat**
4: $\quad \mathscr{E}_{i_l}^{fw} \leftarrow \varnothing, \mathscr{E}_{i_l}^{bw} \leftarrow \varnothing, \ \forall l \in \{0, 1, \ldots, k+1\}$
5: $\quad$ **for** $l = 1, 2, \ldots, k+1$, **do**
6: $\quad\quad$ Compute $\mathscr{E}_{i_l}^{fw}$, the set of all feasible labels E obtained by extending forwardly label $\text{lb}_{i_{l-1}}$ along arc (i_{l-1}, i_l) and such that $E(r) \le \text{ub}_{i_l}(r), \ \forall r \in \bar{R}$.
7: $\quad\quad \text{lb}_{i_l}(r) \leftarrow \min_{E \in \mathscr{E}_{i_l}^{fw}} E(r), \ \forall r \in \bar{R}$
8: $\quad$ **for** $l = k, k-1, \ldots, 0$, **do**
9: $\quad\quad$ Compute $\mathscr{E}_{i_l}^{bw}$, the set of all feasible labels E obtained by extending backwardly label $\text{ub}_{i_{l+1}}$ along arc (i_l, i_{l+1}) and such that $E(r) \ge \text{lb}_{i_l}(r)$, $\forall r \in \bar{R}$.
10: $\quad\quad \text{ub}_{i_l}(r) \leftarrow \max_{E \in \mathscr{E}_{i_l}^{bw}} E(r), \ \forall r \in \bar{R}$
11: **until** some stop criterion is met

To illustrate this preprocessing step, consider the example presented at the end of §4.2.2 (discarding all resources not in $\bar{R}$). Assume that at the beginning of an iteration of the repeat loop, $\text{lb}_i = (1{,}300,\ 50,\ 150,\ 320)$ (that is, $\text{lb}_i = E_i$ in Table 1) and $\text{ub}_j = (2{,}650,\ 100,\ 150,\ 220)$, where the components in these vectors correspond to components T, L, $T^{\text{drive, int}}$, and $T^{\text{drive, day}}$ of the resource vectors. By extending label lb_i along arc (i, j) in step 6, only two feasible labels at vertex j are created, namely, $E_3 = (2{,}365,\ 70,\ 80,\ 80)$ and $E_4 = (2{,}320,\ 70,\ 180,\ 180)$. Then, both labels respect the upper bound conditions and $\mathscr{E}_j^{fw} = \{E_3, E_4\}$. Computing the lowest value over all labels in $\mathscr{E}_j^{fw}$ for each resource in step 7, we obtain $\text{lb}_j = (2{,}320,\ 70,\ 80,\ 80)$. Afterward, the forward labeling algorithm extends this new label until vertex i_{k+1} is reached, and then the backward labeling algorithm in step 9 begins. Now, assume that this latter algorithm computes the following three backward labels when extending the label ub_g along arc (j, g): $E_5 = (2{,}250,\ 100,\ 50,\ 200)$, $E_6 = (2{,}575,\ 100,\ 140,\ 140)$, and $E_7 = (2{,}450,\ 100,\ 120,\ 160)$. Because $E_5(T) = 2{,}250 < \text{lb}_j(T) = 2{,}320$, E_5 does not respect the lower bound conditions. Thus, $\mathscr{E}_j^{bw} = \{E_6, E_7\}$ and ub_j can be updated to $\text{ub}_j = (2{,}575,\ 100,\ 140,\ 160)$ in step 10. If there was a next iteration in the repeat loop and both labels E_3 and E_4 were generated again in step 6, then E_4 would not belong to $\mathscr{E}_j^{fw}$ because it would violate the updated upper bound conditions. This would also yield an updated lower bound vector lb_j. Note that there will always be at least one label in each set $\mathscr{E}_j^{fw}$ and $\mathscr{E}_j^{bw}$. Otherwise, path p would not be feasible.

4.3.2. Filtering Infeasible Insertions. Once the lower and upper bounds are computed for path p, the route feasibility check for each customer insertion can be performed. If customer j is to be inserted between customers i_l and i_{l+1} in p, then we extend label lb_{i_l} along the arcs (i_l, j) and (j, i_{l+1}) using the resource- and label-extension functions of §§4.2.1 and 4.2.2. If there are no feasible labels E at vertex i_{l+1} that satisfy $E(r) \le \text{ub}_{i_{l+1}}(r), \ \forall r \in \bar{R}$, then the insertion is infeasible. The complexity of this check is thus independent of the route length, except for the insertion of an aggregated customer (sequence of customers fixed for the current LNS iteration) that requires the extension of label lb_{i_l} through the whole arc sequence.

Furthermore, in the present context of column generation, the reduced cost $\text{rc}(i)$ of a route created by a customer insertion i can easily be computed in constant time. When multiple (say, k) insertions need to be checked, one can retain the reduced cost $\text{rc}_{\min}$ of the best feasible insertion tested so far. After evaluating insertion $j \in \{1, 2, \ldots, k-1\}$, this minimal reduced cost is $\text{rc}_{\min} = \min_{i \in I_j} > \text{rc}(i)$, where $I_j \subseteq \{1, 2, \ldots, j\}$ is the subset of feasible insertions already tested. Then, if $I_j \neq \varnothing$ and $\text{rc}(j+1) \ge \text{rc}_{\min}$, it is not necessary to check the feasibility of insertion $j+1$ because, even if it were feasible, it would be dominated by the cheapest insertion found so far.

4.3.3. Identifying Feasible Insertions. Every customer insertion that has not been filtered out (proven infeasible) by the previous procedure needs to be checked for feasibility. This check is performed using the heuristic version of the labeling algorithm exposed in §§4.1 and 4.2. Note that during this search, the bounds lb_j and ub_j computed in the preprocessing step of the bound tightening procedure (Algorithm 4) are used to restrict the resource values at each vertex i instead of the original resource windows $[a_i^r, b_i^r]$ for all $i \in \mathcal{N}_p$ and $r \in R$.

If this heuristic check cannot prove the feasibility of the resulting route, then the insertion is considered

infeasible (even though it could be feasible) and is rejected in the TS column generator. Unfortunately, rejecting feasible insertions reduces the possibility of finding certain negative reduced-cost columns with the heuristic TS column generator.

5. Computational Experiments

This section presents some computational results on the instances proposed by Goel (2009). These instances are derived from the well-known benchmark VRPTW instances of Solomon (1987) that are grouped into six classes: R1, C1, RC1, R2, C2, and RC2. In the R1 and R2 classes, customers are randomly distributed in a square region. They are clustered in the C1 and C2 classes. The customer distribution is mixed in the RC1 and RC2 classes. The R2, C2, and RC2 have larger time windows than the R1, C1, and RC1 instances, which considerably increases the number of feasible routes. In total, there are 56 instances (between 8 and 12 per instance class) that all involve 100 customers. These VRPTW instances are modified as follows for the VRPTWDR. The time windows are multiplied proportionally to obtain a time horizon of 144 hours, and the traveling speed is set at 5 units of distance per hour instead of at 60, as in the initial Solomon (1987) instances. Service time at every customer is fixed to 60 minutes. With these modifications, certain customers in certain instances cannot be visited without their time windows being violated because of the additional time required for breaks and rests in the transit between the depot to the customer or between the customer to the depot. Consequently, to yield feasible instances, the time windows of these customers are further modified to ensure that they can always be reached directly from the depot and that the depot can be reached after visiting them (see Goel 2009).

For all experiments with the proposed LNS algorithm, the values of the parameters (see §3.1) were as follows: $I_1^{\max} = 100$ (maximum number of iterations to reduce the number of vehicles used by one), $I_2^{\max} = 100$ (maximum number of iterations to reduce the total distance), and $M^{\text{rem}} = 60$ (number of customers to remove). Furthermore, the heuristic route feasibility check algorithm of §§4.1 and 4.2 was limited to $M^{back} = 5$ backtracks per customer insertion. All of our experiments were conducted on an AMD Opteron processor clocked at 2.3 GHz.

As in Kok et al. (2009), we performed three sets of experiments that differ by the driver rules considered. The first tests aimed at evaluating the performance of our algorithm on the necessary set of rules that enforce feasibility with respect to Regulation (EC) No. 561/2006 (European Union 2006) alone, namely, a driver must take a 45-minute break after 4.5 hours of driving, he must take an 11-hour rest after 9 hours of driving, the weekly driving time is limited to 56 hours, and a daily rest must be taken within 24 hours after the end of the last daily rest. It is thus a set of strict rules that do not allow exceptions nor break or rest splitting. Table 3 summarizes the computed results (columns PDDR) by instance class and compares them with the results obtained by Goel (2009) and Kok et al. (2009) (column KMKS). For each instance class, the first line indicates the average number of vehicles used over all of the instances in the class, whereas the second line provides the average total distance. For our algorithm and that of Goel (2009), which both include randomness, five runs were performed for each instance. The *Best* columns give the best results over the five runs whereas the *Avg.* columns provide the average of the results. The lines CNV and CTD present the cumulative number of vehicles used and the cumulative total distance over all instances, respectively. Finally, the processor used (CPU: Pentium 4 for Goel 2009, Pentium M for Kok et al. 2009) and the average computational time per instance and run (in minutes) are given at the bottom of Table 3 together with the number of runs per instance.

For these first experiments, our LNS algorithm clearly outperforms the algorithms of Goel (2009) and Kok et al. (2009). Indeed, the solutions computed by our algorithm require much fewer vehicles (the solutions of Goel 2009 and Kok et al. 2009 use, respectively, 46% and 21% more vehicles than our solutions), which is the primary objective of the VRPTWDR. Furthermore, the cumulative total distance is also reduced significantly. Finally, notice that our LNS algorithm is faster than the algorithm of

Table 3 **Results for the Necessary Set of Rules of Regulation (EC) No. 561/2006 (European Union 2006)**

	PDDR		Goel (2009)		
	Best	Avg.	Best	Avg.	KMKS
C1	10.00	10.00	11.11	12.04	10.33
	847.69	847.70	1,054.45	1,096.58	965.44
C2	4.38	4.38	8.38	9.58	5.00
	688.64	694.47	954.64	1,008.71	770.42
R1	8.08	8.13	10.92	11.93	9.67
	997.18	993.91	1,144.23	1,180.96	1,152.39
R2	5.00	5.04	10.27	11.27	7.55
	948.51	957.64	1,107.14	1,151.24	1,100.83
RC1	9.00	9.00	11.13	12.10	10.25
	1,112.51	1,117.82	1,347.75	1,373.10	1,300.60
RC2	5.88	5.93	10.00	11.43	8.13
	1,127.75	1,127.77	1,347.26	1,389.50	1,266.64
CNV	396	397.4	580	640.4	479
CTD	53,460	53,611	64,596	66,875	61,328
CPU	OPT 2.3 GHz	OPT 2.3 GHz	P4 2.8 GHz	P4 2.8 GHz	PM 2.0 GHz
Time (min)		10		30	1
Runs		5		5	1

Table 4 **Results for the Necessary Set of Rules of Regulation (EC) No. 561/2006 and Directive 2002/15/EC (European Union 2006, 2002)**

	PDDR		
	Best	Avg.	KMKS
C1	10.00	10.00	10.33
	847.61	847.63	949.31
C2	5.00	5.00	5.75
	724.08	730.88	834.37
R1	8.17	8.22	9.67
	987.95	987.94	1,155.89
R2	5.73	5.73	7.91
	932.95	940.17	1,097.26
RC1	9.00	9.00	10.25
	1,112.93	1,118.23	1,300.14
RC2	6.25	6.40	8.50
	1,122.04	1,117.37	1,264.52
CNV	413	414.8	492
CTD	53,419	53,558	61,677
CPU	OPT 2.3 GHz	OPT 2.3 GHz	PM 2.0 GHz
Time (min)		11	1
Runs		5	1

Table 5 **Results for the Complete Set of Rules**

	PDDR		(Kok et al. 2009)
	Best	Avg.	KMKS
C1	10.00	10.00	10.11
	847.61	847.62	937.08
C2	4.63	4.63	5.25
	689.95	694.95	773.80
R1	8.08	8.08	9.33
	971.64	975.91	1,142.62
R2	5.45	5.67	7.36
	933.93	928.04	1,084.70
RC1	9.00	9.00	10.00
	1,107.27	1,111.36	1,322.41
RC2	6.13	6.15	8.13
	1,090.80	1,096.59	1,247.37
CNV	405	407.6	471
CTD	52,665	52,771	60,826
CPU	OPT 2.3 GHz	OPT 2.3 GHz	PM 2.0 GHz
Time (min)		88	1
Runs		5	1

Goel (2009) but much slower than the algorithm of Kok et al. (2009).

In addition to the first set of rules, we then considered the working time rules of Directive 2002/15/EC (European Union 2002), which specify that the working time (composed of driving and servicing time) may not exceed 6 hours consecutively and 60 hours weekly. Considering this augmented set of rules now yields feasible solutions with respect to all regulations because only exceptions and relaxations that provide more flexibility are omitted. Table 4 compares the results of our LNS algorithm for this second set of rules against those obtained by Kok et al. (2009), the only authors who take working time regulations into account. Again, our results are much better than those of Kok et al. (2009). As expected, considering additional constraints yields solutions that require more vehicles. However, the total distance traveled is more or less the same. Finally, observe that the treatment of these additional rules has almost no impact on the computational times.

The third set of experiments involved the complete set of rules including the use of short breaks, short rests, and reduced rests as well as extending the maximum daily driving time. Table 5 reports the results for these experiments. Again, we clearly see the superiority of the proposed LNS algorithm over the algorithm of Kok et al. (2009): Compared to our solutions, the solutions of Kok et al. (2009) require 16% more vehicles and produce an increase of 15% in the total distance traveled. However, it is important to notice that the approach of Kok et al. (2009) is extremely fast, as it computes solutions within only one minute of computation time, even when all the rules are considered, as opposed to an average of almost 90 minutes for our algorithm.

Finally, we performed a last series of tests to assess the quality of the solutions produced by the proposed LNS algorithm for reduced computational times. Still considering the complete set of rules, we achieved faster computational times by reducing the size of the neighborhoods explored at each LNS iteration (that is, by reducing the value of M^{rem}, the number of customers removed) and keeping the same values for the other parameters. Table 6 provides the results of these experiments for values of M^{rem} varying between 60 (the value used in the previous tests) and 10. These results present averages over five runs. As expected, reducing the size of the neighborhoods (without changing the number of LNS iterations) accelerates the average computational time but deteriorates solution quality. However, we can see that our algorithm still outperforms the algorithm of Kok et al. (2009) for similar computational times.

Table 6 **Results for the Complete Set of Rules for Varying Neighborhood Sizes**

	M^{rem}	CNV	CTD	Time (min.)
PDDR	60	407.6	52,771	88
	50	409.4	52,906	52
	40	411.4	53,081	27
	30	415.6	53,554	13
	20	421.8	54,307	5
	10	435.6	57,712	1
KMKS (Kok et al. 2009)	—	471	60,826	1

6. Conclusion

In this paper, we proposed a large neighborhood search algorithm based on a column generation heuristic to solve the VRPTWDR. The column generation heuristic relies on a tabu search algorithm for generating routes dynamically as well as a heuristic labeling algorithm for checking the feasibility of the routes. The proposed solution method, which extends the previous work of Prescott-Gagnon, Desaulniers, and Rousseau (2009), yields state-of-the-art results for known VRPTWDR benchmark instances.

Although it has been shown in Archetti and Savelsbergh (2009) that it is possible to construct, in polynomial time, a feasible driver schedule for a given route under the legislation of the United States, it remains unknown if this problem is polynomial or nonpolynomial when the European regulations are considered. Identifying this complexity would constitute an interesting contribution for the VRPTWDR and could lead to improved solution methods.

Acknowledgments

The authors thank Christoph Rang for very helpful advice on the legal aspects of driver rules. This research was funded by the Bundesministerium für Wirtschaft und Technologie (German Federal Ministry of Economics and Technology) under Grant 19G7032A and by a team research grant from the Fonds Québécois de La Recherche Sur La Nature et Les Technologies.

Appendix A. Additional Resources and Extension Functions

In §4, we presented a procedure for checking the feasibility of a route. To simplify its presentation, we did not consider all driver rules and options. In this appendix, we present the additional resources and extension functions required to handle all rules and options in this procedure.

Beside the eight resource components introduced in §4.1, a resource vector E_i at a vertex $i \in \mathcal{N}_p$ of a route p contains the following seven additional resource components (listed with their corresponding resource windows).

$\beta_i^{\text{drive,ext}} \in [0,1]$ indicates whether, at vertex i, the maximum daily driving time is extended or not to 10 hours for the current day.

$n_i^{\text{dur,ext}} \in [0,2]$ is the number of maximum daily driving time extensions taken up to vertex i.

$\beta_i^{\text{dur,ext}} \in [0,1]$ indicates whether, at vertex i, the current maximum daily duration is extended to 15 hours.

$n_i^{\text{red}} \in [0,3]$ is the number of reduced daily rests taken up to vertex i.

$T_i^{\text{drive,week}} \in [0,3{,}360]$ is the weekly driving time up to vertex i.

$T_i^{\text{work,int}} \in [0,360]$ is the interval working time up to vertex i.

$T_i^{\text{work,week}} \in [0,3{,}600]$ is the weekly working time up to vertex i.

Thus, $E_i = (T_i,\ L_i,\ T_i^{\text{drive,int}},\ n_i^{sb},\ T_i^{\text{drive,day}},\ LT_i,\ XT_i,\ D_i,\ \beta_i^{\text{drive,ext}},\ n_i^{\text{drive,ext}},\ \beta_i^{\text{dur,ext}},\ n_i^{\text{red}},\ T_i^{\text{drive,week}},\ T_i^{\text{work,int}},\ T_i^{\text{work,week}})$.

In the function $ExtendLabel(E_i, i, j)$ presented in Algorithm 2, the label E_i is extended along arc $(i,j) \in \mathcal{A}_p$ using first the resource-extension functions $\text{REF}_{first}(E_i, i, j)$ introduced in §4.2.1 to produce a resource vector $E_j = (T_j,\ L_j,\ T_j^{\text{drive,int}},\ n_j^{sb},\ T_j^{\text{drive,day}},\ LT_j,\ XT_j,\ D_j,\ \beta_j^{\text{drive,ext}},\ n_j^{\text{drive,ext}},\ \beta_j^{\text{dur,ext}},\ n_j^{\text{red}},\ T_j^{\text{drive,week}},\ T_j^{\text{work,int}},\ T_j^{\text{work,week}})$. Beside functions (5)–(12), this set of extension functions also includes the following functions:

$$\beta_j^{\text{drive,ext}} = \beta_i^{\text{drive,ext}}, \quad n_j^{\text{drive,ext}} = n_i^{\text{drive,ext}},$$
$$\beta_j^{\text{dur,ext}} = \beta_i^{\text{dur,ext}}, \quad n_j^{\text{red}} = n_i^{\text{red}}, \tag{A1}$$
$$T_j^{\text{drive,week}} = T_i^{\text{drive,week}} + t_{ij}, \tag{A2}$$
$$T_j^{\text{work,int}} = T_i^{\text{work,int}} + s_i + t_{ij}, \tag{A3}$$
$$T_j^{\text{work,week}} = T_i^{\text{work,week}} + s_i + t_{ij}. \tag{A4}$$

In step 5 of Algorithm 2, E_j is tested for extendability using the following definition that takes into account the additional resources: a label E_j is said to be extendable if $T_j \le b_j$, $L_j \le Q$, $T_j^{\text{drive,week}} \le 3{,}360$, $T_j^{\text{work,week}} \le 3{,}600$, and $\max\{T_j^{\text{drive,int}} - 270,\ T_j^{\text{drive,day}} - 540,\ D_j + s_j - 780,\ T_j^{\text{work,int}} - 360\} > 0$.

The resource vector E_j might be infeasible, but when it is extendable different options can be applied to possibly yield feasible labels. In §4.2.1, only three options were considered: inserting a short break (sb), inserting a long break (lb), or inserting a regular long rest (lr). Four additional options are available: inserting a short rest (sf), extending the maximum daily duration (ed), inserting a reduced daily rest (rr), or extending the maximum daily driving time (et). Given a resource vector E_j, the subset of options $\Delta(E_j, j)$ applicable along an arc $(i,j) \in \mathcal{A}_p$ is thus a subset of $\{sb, lb, lr, sr, ed, rr, et\}$. Applying such an option generates a new resource vector $\bar{E}_j = (\bar{T}_j,\ \bar{L}_j,\ \bar{T}_j^{\text{drive,int}},\ \bar{n}_j^{sb},\ \bar{T}_j^{\text{drive,day}},\ \bar{LT}_j,\ \bar{XT}_j,\ \bar{D}_j,\ \bar{\beta}_j^{\text{drive,ext}},\ \bar{n}_j^{\text{drive,ext}},\ \bar{\beta}_j^{\text{dur,ext}},\ \bar{n}_j^{\text{red}},\ \bar{T}_j^{\text{drive,week}},\ \bar{T}_j^{\text{work,int}},\ \bar{T}_j^{\text{work,week}})$ that is obtained using resource-extension functions specific to this option. Some of the extension functions for the options sb, lb, and lr were described in §4.2.1. Next, we complete these extension functions for the resources introduced in this appendix. We also provide the extension functions for the options sr, ed, rr, and et.

Short break. Inserting a short break has no impact on the value of the additional resources. Such an option belongs to $\Delta(E_j, j)$ if the conditions stated in §4.2.1 hold and the additional condition $T_j^{\text{work,int}} < 360$ is also satisfied (otherwise, a long break or a rest is mandatory). The resource-extension functions $\text{REF}_{sb}(E_j, j)$ for this option include (13)–(18) and the following functions:

$$\bar{\beta}_j^{\text{drive,ext}} = \beta_j^{\text{drive,ext}}, \quad \bar{n}_j^{\text{drive,ext}} = n_j^{\text{drive,ext}},$$
$$\bar{\beta}_j^{\text{dur,ext}} = \beta^{\text{dur,ext}}, \quad \bar{n}_j^{\text{red}} = n_j^{\text{red}} \tag{A5}$$
$$\bar{T}_j^{\text{drive,week}} = T_j^{\text{drive,week}}, \quad \bar{T}_j^{\text{work,int}} = T_j^{\text{work,int}},$$
$$\bar{T}_j^{\text{work,week}} = T_j^{\text{work,week}}. \tag{A6}$$

Long break. Inserting a long break also modifies the value of the interval working time resource. The values of

the other additional resources do not change. Such an option belongs to $\Delta(E_j, j)$ if $\max\{T_j^{\text{drive, int}} - 270,\ T_j^{\text{work, int}} - 360\} > 0$ or $0 < a_j - XT_j < 180$ and if $\max\{T_j^{\text{drive, int}} - 270,\ T_j^{\text{work, int}} - 360\} > \max\{T_j^{\text{drive, day}} - 540,\ D_j - \max\{a_j - XT_j, 0\} - 780\}$. These conditions indicate that a long break is needed if the maximum interval driving time or the maximum working time is reached, or if there is unavoidable waiting that does not exceed the minimum duration of a short rest (180 minutes). However, the insertion of a long break is dominated by the insertion of a long rest if the maximum daily drive time or the maximum daily duration is reached before the maximum interval driving or working time. Note also that $lb \notin \Delta(E_j, j)$ if the break is to be inserted during service at vertex i.

The resource-extension functions $\text{REF}_{lb}(E_j, j)$ for this option include (19)–(25), except for function (20), which is replaced by

$$\overline{T}_j^{\text{drive, int}} = \max\{T_j^{\text{drive, int}} - 270,\ T_j^{\text{work, int}} - 360,\ 0\} \tag{A7}$$

and the following functions:

$$\overline{T}_j^{\text{work, int}} = \overline{T}_j^{\text{drive, int}} \tag{A8}$$

$$\overline{\beta}_j^{\text{drive, ext}} = \beta_j^{\text{drive, ext}}, \quad \overline{n}_j^{\text{drive, ext}} = n_j^{\text{drive, ext}},$$

$$\overline{\beta}_j^{\text{dur, ext}} = \beta_j^{\text{dur, ext}} \tag{A9}$$

$$\overline{n}_j^{\text{red}} = n_j^{\text{red}}, \quad \overline{T}_j^{\text{drive, week}} = T_j^{\text{drive, week}},$$

$$\overline{T}_j^{\text{work, week}} = T_j^{\text{work, week}}. \tag{A10}$$

Long rest. Taking a long rest resets some of the resource values. The duration of this rest is 660 minutes if $\beta_j^{\text{dur, ext}} = 0$ and 540 minutes otherwise. As before, $lr \in \Delta(E_j, j)$ if this rest does not have to be taken during service at vertex i. The resource-extension functions $\text{REF}_{lr}(E_j, j)$ for this option include (26)–(33), except for (28), which is replaced by

$$\overline{T}_j^{\text{drive, int}} = \max\{T_j^{\text{drive, int}} - 270, T_j^{\text{work, int}} - 360, T_j^{\text{drive, day}} - 540, D_j - \max\{a_j - XT_j, 0\} - 780, 0\} \tag{A11}$$

and the following functions:

$$\overline{\beta}_j^{\text{drive, ext}} = 0, \quad \overline{\beta}_j^{\text{dur, ext}} = 0, \tag{A12}$$

$$\overline{T}_j^{\text{work, int}} = \overline{T}_j^{\text{drive, int}}, \tag{A13}$$

$$\overline{n}_j^{\text{drive, ext}} = n_j^{\text{drive, ext}}, \quad \overline{n}_j^{\text{red}} = n_j^{\text{red}}, \quad \overline{T}_j^{\text{drive, week}} = T_j^{\text{drive, week}}, \quad \overline{T}_j^{\text{work, week}} = T_j^{\text{work, week}}. \tag{A14}$$

Short rest. The insertion of a short rest (option sr) is treated similarly to the insertion of a long break, the only difference being the duration of the rest, which is set to 180 minutes. It belongs to $\Delta(E_j, j)$ if $\max\{T_j^{\text{drive, int}} - 270, T_j^{\text{work, int}} - 360\} > \max\{T_j^{\text{drive, day}} - 540, D_j - \max\{a_j - XT_j, 0\} - 780\}$ and $\beta_j^{\text{dur, ext}} = 0$ unless the rest has to be taken during service at vertex i. The second condition ensures that no two consecutive short rests are taken or that a short rest is taken before a reduced daily rest (in which case it is dominated). The resource-extension functions $\text{REF}_{sr}(E_j, j)$ for this option include the extension functions (19), (21)–(25), and (A7) with $k_j = 180$. Because the next long rest after a short rest only needs to last 9 hours, the maximum daily duration can be extended to 15 hours. This can be enforced by keeping the upper bound to 13 hours and reducing the updated daily duration by 2 hours. Also, setting $\beta_j^{\text{dur, ext}}$ to one allows a next long rest of nine hours and ensures that no other short rests will be taken until then. This is adjusted by the following resource-extension functions:

$$\overline{D}_j = D_j - 120, \tag{A15}$$

$$\overline{\beta}_j^{\text{dur, ext}} = 1. \tag{A16}$$

The rest of the resource-extension functions $\text{REF}_{sr}(E_j, j)$ are as follows:

$$\overline{T}_j^{\text{work, int}} = \overline{T}_j^{\text{drive, int}}, \tag{A17}$$

$$\overline{\beta}_j^{\text{drive, ext}} = \beta_j^{\text{drive, ext}}, \quad \overline{n}_j^{\text{drive, ext}} = n_j^{\text{drive, ext}}, \tag{A18}$$

$$\overline{T}_j^{\text{drive, week}} = T_j^{\text{drive, week}}, \quad \overline{T}_j^{\text{work, week}} = T_j^{\text{work, week}}. \tag{A19}$$

Extended maximum daily duration. Extending the maximum daily duration from 13 hours to 15 hours (option ed) induces a reduction of the duration of the next long rest (from 11 to 9 hours). It belongs to $\Delta(E_j, j)$ if $D_j \geq 780$, $D_j - \max\{a_j - XT_j, 0\} - 780 > \max\{T_j^{\text{work, int}} - 360,\ T_j^{\text{drive, int}} - 270,\ T_j^{\text{drive, day}} - 540\}$, $\beta_j^{\text{dur, ext}} = 0$, and $n_j^{\text{red}} < 3$. This means that this option can be taken if the maximum daily duration is exceeded and if this occurs before the maximum interval driving time, the maximum daily driving time, and the maximum interval working time are exceeded. Furthermore, if the maximum daily duration is already extended ($\beta_j^{\text{dur, ext}} = 1$), it cannot be extended further. Note that this condition ($\beta_j^{\text{dur, ext}} = 1$) is also set when taking a short rest, which can be shown to be dominated by the extension of the maximum daily duration. Finally, option ed is not allowed if three reduced daily rests were already taken during the week.

For this option, the resource extension-functions $\text{REF}_{ed}(E_j, j)$ are as follows:

$$\overline{D}_j = D_j - 120, \tag{A20}$$

$$\overline{\beta}_j^{\text{dur, ext}} = 1, \tag{A21}$$

$$\overline{n}_j^{\text{red}} = n_j^{\text{red}} + 1 \tag{A22}$$

$$\overline{\beta}_j^{\text{drive, ext}} = \beta_j^{\text{drive, ext}}, \quad \overline{n}_j^{\text{drive, ext}} = n_j^{\text{drive, ext}},$$

$$\overline{T}_j^{\text{drive, week}} = T_j^{\text{drive, week}} \tag{A23}$$

$$\overline{T}_j^{\text{work, int}} = T_j^{\text{work, int}}, \quad \overline{T}_j^{\text{work, week}} = T_j^{\text{work, week}},$$

$$\overline{T}_j = T_j, \quad \overline{L}_j = L_j \tag{A24}$$

$$\overline{T}_j^{\text{drive, int}} = T_j^{\text{drive, int}}, \quad \overline{T}_j^{\text{drive, day}} = T_j^{\text{drive, day}},$$

$$\overline{n}_j^{sb} = n_j^{sb}, \quad \overline{XT}_j = XT_j, \quad \overline{LT}_j = LT_j. \tag{A25}$$

Reduced daily rest. As mentioned earlier, a long rest can be of reduced duration if $\beta_j^{\text{drive, ext}} = 1$, that is, if a short rest was taken before or if the maximum daily duration

was extended. Besides these cases, a reduced daily rest of 540 minutes can also be inserted instead of a 660-minute long rest just to save time and be able to reach a customer before the end of its time window. In this case, such an insertion is treated similarly to the insertion of a long rest. Therefore, option *rr* belongs to $\Delta(E_j, j)$ if $\beta_j^{\text{dur, ext}} = 0$ and $n_j^{\text{red}} < 3$, unless it is to be scheduled during service at vertex i.

The resource-extension functions $\text{REF}_{rr}(E_j, j)$ are the same as those of option *lr*, except that the rest always lasts 540 minutes and the component counting the number of reduced daily rests is updated as follows:

$$\bar{n}_j^{\text{red}} = n_j^{\text{red}} + 1. \tag{A26}$$

Extended maximum daily driving time. Extending the maximum daily driving time from 9 to 10 hours (option *et*) belongs to $\Delta(E_j, j)$ if $T_j^{\text{drive, day}} \geq 540$, $\beta_j^{\text{drive, ext}} = 0$, $n_j^{\text{drive, ext}} < 2$, and $T_j^{\text{drive, day}} - 540 > \max\{T_j^{\text{work, int}} - 360,\ T_j^{\text{drive, int}} - 270,\ D_j - \max\{a_j - XT_j, 0\} - 780\}$. These conditions state that the maximum daily driving time must be exceeded, this maximum has not been extended for the current day, this maximum has not been extended twice during the week, and the maximum daily driving time must be exceeded before the other resources. The daily driving time resource is reduced by 1 hour in order to extend the maximum daily driving time from 9 to 10 hours, which results in the following resource-extension functions:

$$\bar{T}_j^{\text{drive, day}} = T_j^{\text{drive, day}} - 60, \tag{A27}$$

$$\bar{\beta}_j^{\text{drive, ext}} = 1, \tag{A28}$$

$$\bar{n}_j^{\text{drive, ext}} = n_j^{\text{drive, ext}} + 1. \tag{A29}$$

The other resource values are not affected by this option. Thus, besides functions (A27)–(A29), the resource-extension functions $\text{REF}_{ed}(E_j, j)$ include the following functions:

$$\bar{T}_j = T_j, \quad \bar{L}_j = L_j, \quad \bar{T}_j^{\text{drive, int}} = T_j^{\text{drive, int}}, \quad \bar{n}_j^{sb} = n_j^{sb}, \tag{A30}$$

$$\overline{XT}_j = XT_j, \quad \overline{LT}_j = LT_j, \quad \bar{D}_j = D_j$$
$$\bar{T}_j^{\text{work, int}} = T_j^{\text{work, int}}, \tag{A31}$$

$$\bar{T}_j^{\text{work, week}} = T_j^{\text{work, week}}, \quad \bar{n}_j^{\text{red}} = n_j^{\text{red}}$$
$$\bar{T}_j^{\text{drive, week}} = T_j^{\text{drive, week}}. \tag{A32}$$

The introduction of the additional resources also has an impact on the resource bound tightening procedure described in §4.3.1. Indeed, when computing the lower and upper bounds $\text{lb}_{i_l}(r)$ and $\text{ub}_{i_l}(r)$ for the resources $r \in \bar{R}$ at all vertices i_l of a path $p = i_0 - i_1 - \cdots - i_k - i_{k+1}$, neither short breaks nor short rests are considered. Instead, the duration of all long breaks is set to 30 minutes, and that of all long rests is set to 9 hours. Moreover, the following relaxation is also used: The value of the daily driving time resource is set to −60 at the beginning of each day to take into account the extended 10-hour maximum daily driving time. These relaxations do not necessarily yield the tightest bounds, but they do ensure the validity of the computed bounds. Considering them, the set $\bar{R}$ of resources involved in the resource bound tightening algorithm can be augmented by the weekly driving time, the interval working time, and the weekly working time resources.

Appendix B. Detailed Results

In §5 Tables 3–5 reported a summary of the results obtained by the proposed LNS algorithm for three different sets of rules. In this appendix, we provide in Table B.1 the details of these results, namely, the number of vehicles used (NV) and the total distance traveled (TD) in the computed solution for each individual VRPTWDR instance and each set of rules. Again, we report best and average results over five runs.

Table B.1 Detailed Results Obtained by the Proposed LNS Algorithm

	Basic rules				With working time rules				All rules			
	Best		Avg.		Best		Avg.		Best		Avg.	
	NV	TD	NV	TD	NV	TD	NV	TD	NV	TD	NV	TD
C101	10	931.37	10	931.37	10	931.37	10	931.37	10	931.37	10	931.37
C102	10	904.25	10	904.34	10	904.25	10	904.50	10	904.25	10	904.34
C103	10	833.92	10	833.92	10	833.92	10	833.92	10	833.19	10	833.19
C104	10	819.81	10	819.81	10	819.81	10	819.81	10	819.81	10	819.81
C105	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94
C106	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94
C107	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94	10	828.94
C108	10	827.38	10	827.38	10	827.38	10	827.38	10	827.38	10	827.38
C109	10	825.65	10	825.65	10	825.65	10	825.65	10	825.65	10	825.65
C201	6	882.23	6	916.36	6	881.50	6	891.41	5	810.05	5	834.29
C202	5	774.87	5	781.71	5	813.40	5	835.64	5	695.75	5	702.92
C203	4	665.10	4	665.10	5	698.13	5	703.58	4	661.84	4	662.28
C204	4	646.25	4	647.55	4	662.01	4	667.79	4	649.70	4	651.66
C205	4	639.22	4	639.27	5	684.42	5	685.56	5	678.70	5	681.02
C206	4	628.93	4	630.25	5	685.63	5	688.30	5	676.57	5	677.99
C207	4	646.50	4	647.16	5	693.97	5	698.44	5	674.67	5	675.40
C208	4	626.04	4	628.36	5	673.61	5	676.28	4	672.30	4	674.02

Table B.1 (Cont'd.)

	Basic rules				With working time rules				All rules			
	Best		Avg.		Best		Avg.		Best		Avg.	
	NV	TD	NV	TD	NV	TD	NV	TD	NV	TD	NV	TD
R101	9	1,503.75	9.6	1,400.33	10	1,326.78	10	1,326.93	9	1,319.88	9	1,324.66
R102	8	1,227.28	8	1,246.53	8	1,263.11	8.4	1,227.93	8	1,177.31	8	1,189.03
R103	8	972.26	8	972.29	8	1,263.11	8	978.04	8	967.96	8	969.35
R104	8	859.68	8	864.03	8	868.88	8.2	870.21	8	855.72	8	860.05
R105	8	1,113.99	8	1,122.94	8	1,116.68	8	1,121.31	8	1,090.69	8	1,090.90
R106	8	1,008.74	8	1,016.19	8	1,019.20	8	1,022.87	8	998.35	8	1,000.62
R107	8	900.93	8	902.01	8	902.76	8	904.89	8	892.90	8	896.49
R108	8	849.81	8	851.89	8	848.45	8	852.11	8	840.95	8	846.22
R109	8	928.61	8	930.14	8	928.61	8	931.69	8	923.28	8	923.60
R110	8	885.43	8	889.81	8	884.83	8	890.71	8	880.19	8	884.71
R111	8	881.69	8	890.21	8	882.07	8	888.77	8	881.27	8	886.33
R112	8	834.01	8	840.55	8	836.72	8	839.82	8	831.13	8	838.96
R201	7	1,242.18	7	1,248.90	7	1,254.84	7	1,266.72	7	1,220.86	7	1,232.59
R202	6	1,108.22	6	1,113.82	6	1,116.22	6	1,118.67	6	1,093.46	6	1,104.26
R203	5	968.26	5	976.45	6	918.82	6	922.62	5	957.70	5.8	916.71
R204	4	789.93	4	812.29	5	776.57	5	783.14	5	770.21	5	772.34
R205	5	1,060.08	5	1,080.76	6	1,011.29	6	1,021.03	6	1,000.40	6	1,003.45
R206	5	936.74	5	939.02	6	929.44	6	932.90	5	958.17	5.8	924.81
R207	4	863.80	4.4	858.25	5	857.74	5	865.09	5	840.61	5	850.21
R208	4	749.46	4	751.03	5	746.39	5	751.66	5	754.21	5	756.61
R209	5	940.29	5	949.57	6	905.62	6	911.78	5	950.53	5.8	911.95
R210	5	982.53	5	1,002.79	6	943.64	6	948.29	6	938.69	6	942.13
R211	5	792.13	5	801.13	5	801.93	5	819.93	5	788.35	5	793.39
RC101	9	1,303.24	9	1,307.29	9	1,305.09	9	1,308.14	9	1,293.82	9	1,298.37
RC102	9	1,180.67	9	1,186.06	9	1,186.64	9	1,193.30	9	1,177.51	9	1,181.76
RC103	9	1,084.10	9	1,092.00	9	1,082.37	9	1,086.51	9	1,085.66	9	1,087.71
RC104	9	993.19	9	994.73	9	993.19	9	995.13	9	993.13	9	993.55
RC105	9	1,220.76	9	1,222.64	9	1,224.41	9	1,227.81	9	1,203.34	9	1,207.21
RC106	9	1,095.18	9	1,107.52	9	1,093.62	9	1,099.70	9	1,093.96	9	1,103.05
RC107	9	1,033.79	9	1,040.18	9	1,029.58	9	1,039.70	9	1,028.11	9	1,036.29
RC108	9	989.17	9	992.19	9	988.54	9	995.55	9	982.59	9	982.94
RC201	7	1,424.54	7	1,326.88	8	1,384.01	8	1,385.84	7	1,395.15	7	1,398.04
RC202	6	1,247.44	6	1,186.69	7	1,193.12	7	1,193.91	7	1,153.45	7	1,156.78
RC203	5	1,120.97	5.4	949.37	6	1,037.39	6	1,040.52	6	1,016.92	6	1,020.71
RC204	5	859.40	5	792.51	5	877.17	5	885.75	5	863.22	5	869.84
RC205	7	1,294.55	7	1,294.63	7	1,310.16	7	1,312.01	7	1,270.78	7	1,273.89
RC206	6	1,138.35	6	1,150.42	6	1,179.85	6.4	1,166.05	6	1,129.39	6.2	1,133.85
RC207	6	1,073.72	6	1,075.92	6	1,087.68	6	1,089.52	6	1,046.83	6	1,056.93
RC208	5	863.02	5	869.05	5	906.90	5.8	865.35	5	850.63	5	862.71
Total	396	53,460	397.4	53,611	413	53,419	414.8	53,558	405	52,665	407.6	52,771

References

Archetti, C., M. Savelsbergh. 2009. The trip scheduling problem. *Transportation Sci.* **43**(4) 417–431.

Bartodziej, P., U. Derigs, D. Malcherek, U. Vogel. 2009. Models and algorithms for solving combined vehicle and crew scheduling with rest constraints: An application to road feeder service planning in air cargo transportation. *OR Spectrum* **31**(2) 405–429.

Bräysy, O., M. Gendreau. 2005a. Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation Sci.* **39**(1) 104–118.

Bräysy, O., M. Gendreau. 2005b. Vehicle routing problem with time windows, Part II: Metaheuristics. *Transportation Sci.* **39** 119–139.

European Union. 2002. Directive 2002/15/EC of the European Parliament and of the Council of 11 March 2002 on the organisation of the working time of persons performing mobile road transport activities. *Official J. Eur. Union*, L 080.

European Union. 2006. Regulation (EC) No. 561/2006 of the European Parliament and of the Council of 15 March 2006 on the harmonisation of certain social legislation relating to road transport and amending Council Regulations (EEC) No. 3821/85 and (EC) No. 2135/98, and repealing Council Regulation (EEC) No. 3820/85. *Official J. Eur. Union*, L 102.

Goel, A. 2009. Vehicle routing and scheduling with drivers' working hours. *Transportation Sci.* **43**(1) 17–26.

Goel, A., V. Gruhn. 2006. Drivers' working hours in vehicle routing and scheduling. *Proc. 9th IEEE Internat. Conf. Intelligent Transportation Systems (ITSC 2006), Toronto, Ontario*, 1280–1285.

Golden, B., S. Raghavan, E. Wasil, eds. 2008. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Operations Research/Computer Science Interfaces Series, Springer, New York.

Irnich, S. 2008. Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum* **30**(1) 113–148.

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

Kallehauge, B., J. Larsen, O. B. G. Madsen, M. M. Solomon. 2005. Vehicle routing problem with time windows. G. Desaulniers, J. Desrosiers, M. M. Solomon, eds. *Column Generation*. Springer, New York, 67–98.

Kok, A. L., C. M. Meyer, H. Kopfer, J. M. J. Schutten. 2009. Dynamic programming algorithm for the vehicle routing problem with time windows and EC social legislation. Working paper 270, University of Twente, Enschede, The Netherlands.

Prescott-Gagnon, E., G. Desaulniers, L.-M. Rousseau. 2009. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*. **54** 190–204.

Savelsbergh, M., M. Sol. 1998. Drive: Dynamic routing of independent vehicles. *Oper. Res.* **46**(4) 474–490.

Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Oper. Res.* **35**(2) 254–265.

Toth, P., D. Vigo, eds. 2002. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia.

Xu, H., Z. L. Chen, S. Rajagopal, S. Arunapuram. 2003. Solving a practical pick-up and delivery problem. *Transportation Sci.* **37**(3) 347–364.

Zäpfel, G., M. Bögl. 2008. Multi-period vehicle routing and crew scheduling with outsourcing options. *Internat. J. Production Econom.* **113**(2) 980–996.