Received: 8 December 2018 | Revised: 21 June 2019 | Accepted: 5 August 2019 | Published on: 14 September 2019

DOI: 10.1002/net.21905

RESEARCH ARTICLE

WILEY

Adaptive large neighborhood search for the vehicle routing problem with synchronization constraints at the delivery location

Briseida Sarasola[1] | **Karl F. Doerner**[1,2]

[1]Christian Doppler Laboratory for Efficient Intermodal Transport Operations, Department of Business Decisions and Analytics, University of Vienna, Vienna, Austria

[2]Data Science @ University of Vienna, Vienna, Austria

Correspondence

Karl F. Doerner, Christian Doppler Laboratory for Efficient Intermodal Transport Operations, Department of Business Decisions and Analytics, University of Vienna, Vienna, Austria.

Email: karl.doerner@univie.ac.at

Funding information

This research was supported by the National Foundation for Research, Technology and Development. Austrian Federal Ministry for Digital and Economic Affairs.

Abstract

A vehicle routing problem with synchronization constraints arises in urban freight transportation, in which context customers require deliveries from one or more logistics service providers. These deliveries should be efficient to reduce idle times at the delivery locations. Idle time is defined as nonservice time between the first and the last delivery received by the customer. We propose a strategy which relies on self-imposed time windows, and we compare our approach with an exact determination of a feasible schedule and fixed time windows. The results show that idle times can be reduced by 54.12%-79.77%, with an average cost rise of 9.87%. In addition, self-imposed time windows provide solutions with 15.74%-21.43% lower costs than feasibility checks for short runtimes and 13.71%-21.15% lower than fixed time windows.

KEYWORDS

adaptive large neighborhood search, cooperation, metaheuristic, self-imposed time windows, synchronized transportation, vehicle routing problem

1 | INTRODUCTION

The daily transportation and delivery of goods is extremely challenging in urban areas. Courier services deliver thousands of packages every day in a highly competitive environment, and their business has dramatically increased with the rise of e-commerce, such that they require more cost-effective solutions with acceptable levels of customer service [17]. Such packages ship to both individuals and companies, and a single recipient frequently might expect to receive several packages from more than one courier on the same day. Simultaneously, shops and stores with multiple branches need to be restocked with consumer goods daily. Such deliveries also often involve several service providers and multiple deliveries. For example, private customers might order several products that are shipped by different courier services, whereas supermarkets are delivered by bakeries, dairy products companies, and fresh produce distributors. In this sense, customers, both private and business, dedicate a significant part of the day to awaiting and attending to deliveries.

To reduce the associated effort, customers and logistics service providers typically need to agree on a service time window [15], aligned with customers' expectations in the service of customer-oriented business models [17]. Such agreements tend to reflect a one-to-one orientation, despite the involvement of multiple service providers. Thus, customers might want to set predefined time windows (e.g., 07:00-10:00) for all suppliers or find a way to cope with deliveries throughout the day. In this context, new business models arise to meet expectations on customer service, when several service providers need to coordinate their service at the customer. Such a model might involve for example a third-party logistics (3PL) company running an online platform for both suppliers and end customers [60]. Customers might order several products on the platform, which might in turn be sold by different suppliers. As a result, deliveries might be carried out by several logistics companies required to cooperate

to satisfy customers' desire to receive products in tight delivery time windows. Similar models to increase service satisfaction could be explored in other sectors, for example, the housing and decoration logistics [52], construction industry, and event organization management.

City logistics has emerged as a specific research field for optimizing transport activities in urban areas, aiming at the reduction of the troubles related to freight transportation while also supporting the economic development of the cities [7]. Most of the classical literature deals with the consolidation of loads of several carriers into smaller vehicles [6]. However, such initiatives seldom continue operating once the project ends; for example, 150 such projects were initiated in Europe within the period 1988-2011, but only five of them survived [55, 57]. This failure is motivated by confidentiality issues in a very competitive field, carrier aspirations for keeping their brand image and relationship with customers, as well as the limited amount of freight share that can be caught by this approach [57]. As a result, we have seen an increase in the variety of initiatives to improve urban freight transportation over the last years, such as access restrictions [37, 59], no parking policies [2], reducing emissions and fuel consumption [18], highly variable travel times [16, 42], and synchronized deliveries at the customer location [52, 60].

We propose an integrated logistics system to tackle these difficulties, for which the goal is to determine a service time window and to provide a compact schedule for delivering goods from multiple service providers. Compact schedules keep idle times below a maximum time span, and idle time is defined as nonservice time between the first and last delivery received by the customer. To achieve this objective, it is necessary to address the concerns of all participating stakeholders, so that carriers minimize their costs and customers minimize their effort by getting all their deliveries during the service time window. Our goal is to provide optimized routes for the deliveries of all carriers in the system, so that all deliveries at a customer location occur within a non-predefined time interval.

The underlying optimization problem can be modeled as a variation of the vehicle routing problem (VRP) with synchronization constraints with the following special features: there are several depots (each belongs to one carrier), each depot has its own set of vehicles and deliveries, each customer can receive deliveries from multiple depots, and all deliveries to a customer must take place within an non-predefined time interval, so that compact schedules are generated for each delivery location. The problem cannot be classified as a multidepot VRP though, because commodities are exclusive of each depot. Due to new inter-route constraints, synchronizing the arrival of several vehicles to customer locations is a challenge for optimization algorithms. The application of VRP with synchronization constraints has expanded to domains such as home health care [19, 34], service technician routing problems [9], airline scheduling [28], ready-mixed concrete delivery [50], forestry [24], snow plowing [49], and military mission planning [45].

Carriers often provide customers with a time window for delivery, as a mechanism to establish customer service [29]. When only one carrier is involved, it is possible to first route customers and then assign them time intervals during which they will receive the service. Customer preferences might also be taken into account [17]. With these considerations, we investigate three techniques to ensure compact service at the delivery location: self-imposed time windows determined by the optimization algorithm during the search, an exact determination of a feasible schedule [35], and fixed time windows imposed before the optimization.

We propose a synchronization of the arrival of an arbitrary number of vehicles belonging to different carriers to the delivery location, which guarantees that the service takes place within a limited, non-predefined time interval. To the best of our knowledge, this idea has only been explored by Shao et al. [52]. In particular, we assume the customer expects one to six orders at the delivery location, and we describe this problem by introducing a mathematical model based on the VRP. Although Shao et al. introduced a similar model, our formulation takes into account that deliveries cannot be attended simultaneously. The VRP with synchronization constraints is a generalization of the VRP and is therefore NP-hard. Because the model cannot be solved to optimality for realistic settings with a commercial solver, we provide a metaheuristic based on the adaptive large neighborhood search (ALNS) framework and investigate three methods to ensure feasible schedules at the delivery location. The choice of ALNS is motivated by its excellent results on rich routing problems, for example, the generalized consistent VRP [33], and routing problems with different synchronization constraints [21, 22, 25, 40]. We also provide two new destroy operators for this problem. To evaluate the algorithms, we use a realistic benchmark set based on real-world data.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. Section 3 describes the problem and introduces the mathematical model. Three methods to ensure compact schedules at the delivery location are presented in Section 4. Our solution method is explained in Section 5. Computational results are discussed in Section 6. Finally, Section 7 contains the conclusion.

2 | RELATED LITERATURE

This section summarizes related work in the relevant areas for the problem considered in this paper. We first discuss some particularly relevant studies on synchronization topics in city logistics. We then provide an overview of the literature on synchronization. Last, we examine related work on time window management.

Some recent city logistics applications require compact delivery schedules, thus involving some degree of synchronization. Xu et al. [60] study an auction mechanism for punctuality and simultaneity in a city region, motivated by a large furniture operator in China that acts as a 3PL company with multiple suppliers and customers. Suppliers compete for logistics services in an auction, and the model is based on the lane covering problem. The nature of this approach is very different from ours due to the competition and auction process. The most related work to our topic is the city logistics synchronization problem with sliding time windows by Shao et al. [52]. They also consider moving time windows to better accommodate deliveries from several vehicles in a compact way. However, there are some important differences with our approach. In their problem, vehicles start at a central depot and first pickup goods at the supplier and then deliver them to the customers. Thus, and although each vehicle that leaves the depot only transports the goods of one supplier, vehicles are stationed at a central depot and can be assigned to every route. In addition, simultaneous arrivals at the customer location can be handled in parallel. Instead, we assume independent carriers that have their own depot and fleet of vehicles. We also consider that conflicts may arise if two (or more) vehicles arrive at the customer at roughly the same time, so that a vehicle might wait while another is being unloaded (see more on resource synchronization in the remainder of this section).

The VRP with synchronization constraints has been surveyed by Drexl [11], who also recently updated the state of the art [12]. Drexl identifies five types of synchronization constraints (task, operation, movement, load, and resource synchronization), and we can classify our problem as requiring both resource and operation synchronization. Resource synchronization means that vehicles compete for common, limited resources [26], whereas for operation synchronization, the time elapsed between the operations performed by several vehicles must fall within a specified, finite interval. In our problem, deliveries are awaited by one person at a time, so the person attending the unloading operation represents a resource shared by every vehicle serving that customer; if a vehicle arrives while another vehicle is being unloaded, it must wait. To avoid idle time at the customer location, this problem also requires that all vehicles serving a customer start and finish their operations within a non-predefined time interval. This time aspect does not match any of the three types of operation synchronization given by Drexl, as it is not purely spatial (we consider time), exact (operations cannot start at the same time due to the restricted resource), or with precedences (there is no required order for the operations).

Operations synchronization with temporal aspects exists in some applications of the VRP and the dial-a-ride problem (DARP). Problems arising in home health care and service technician routing and scheduling, for which two or more employees might be required to complete one task, at the same time or in some predefined sequence, require synchronization. Mankowska et al. [34] study a routing and scheduling problem with synchronization in a home health care setting, with consideration of both simultaneous operation synchronization and precedence constraints, in an effort to minimize total distance, total tardiness, and maximal tardiness. Parragh and Doerner [40] compare three mechanisms to synchronize pairs of requests, embedded in an ALNS, to solve the service technician routing and scheduling problem. Hojabri et al. [27] study the synchronization of two vehicles in a VRP using large neighborhood search (LNS) with constraint programming, and determine that two synchronization-based destroy operators, the synchro vertex removal and the synchro route removal, do not provide good results. The first could be considered a random version of our destroy operators (see Section 5.2 for more detail); locations requiring synchronization are randomly selected and all tasks associated with them are removed from the routes. We have evaluated this operator in our application, but it did not perform well. The second is not relevant for our work, because it uses problem-specific information, which does not apply in our case. Regarding the DARP, synchronization demands arise in multimodal problems that feature the possibility of passenger transfers, such that they demand temporal synchronization with precedence of two vehicles. Reinhardt et al. [46] develop a greedy heuristic for a rich DARP with synchronization constraints that seeks to minimize the number of undelivered passengers with reduced mobility at airports and unnecessary travel time. Schönberger [51] employs a memetic algorithm for a DARP when passengers can change vehicles once or several times.

The aforementioned papers do not consider synchronization of more than two vehicles or tasks, and resource synchronization, when present, only requires the second service to start when the first one has finished, and is therefore easy to manage.

The diverse and application-specific research on resource synchronization includes Ebben et al.'s [14] attempt to tackle the scheduling of automated guided vehicles with limited resources (e.g., number of docks, parking slots, cargo storage capacity). El Hachemi et al. [24] study a forest management application with restricted wood loading machine resources. Grimault et al. [22], investigating a pickup and delivery problem with resource synchronization, introduce a specific synchronization-based destroy operator for their ALNS. The resource destroy operator removes all requests from a randomly selected resource, and could be thus considered similar to Hojabri's synchro vertex removal. By designing a matheuristic based on LNS for the VRP with cross-docking, where synchronization happens at the consolidation stage in a single cross-dock, Grangier et al. [21] introduce one problem-specific destroy operator, the transfer removal, which removes requests transferred between pairs of routes. This operator did not provide good results in our application, probably due to the different nature of the problems (in this case, requests are transferred at a single point, the cross-dock).

Unlike previous work on resource synchronization, in which vehicles can wait as long as necessary despite waiting times, we aim at achieving compact schedules and thus consider resource and operation synchronization simultaneously. Their solution methods highlight the importance of using specific operators that take into account the synchronization requirements.

Several other studies assign time windows to customers, to overcome uncertainty (however, no synchronization is involved). Jabali et al. [29] refer to "self-imposed time windows," determined by the carrier company, that might provide an estimated arrival time to customers even in the presence of uncertain travel times. The basic mechanism allocates time buffers throughout the routes, and the objective function penalizes overtime and tardiness. Finally, Gschwind and Irnich [23] investigate dynamic time windows as induced in the DARP. In a dynamic time window, two operations must be executed within a given time, such that they are similar to self-imposed time windows. However, dynamic time windows constitute intraroute synchronization constraints, bounding the time span between a pickup and a delivery. In addition, and due to the nature of the problem, a pickup must happen before the corresponding delivery.

3 | PROBLEM DEFINITION

We now present the mathematical model to precisely describe the problem. In the VRP with synchronization constraints at the delivery location (VRPSCDL), a set of n deliveries to m customers must be served from p depots. In this work, a delivery is an order served by a vehicle and a customer refers to a delivery location that may receive orders from different depots. The set of deliveries is denoted by D, the set of customers is U, and the depots are represented by the set of departure depot nodes P_1 and the set of arrival depot nodes P_2. The total set of nodes is thus $N = P_1 \cup D \cup P_2$. Each delivery is associated with a customer and a depot, such that D'_i is the subset of deliveries received by customer i, and D''_e is the subset of deliveries to be fulfilled by depot $e \in P_1$. Every depot $e \in P_1$ has an associated fleet of vehicles V_e leaving from e and arriving to $n+p+e \in P_2$, where each vehicle has a maximum capacity Q and must return to its depot by time T at the latest. The subset of nodes associated with depot $e \in P_1$ is denoted N_e, where $N_e = \{e, n+p+e\} \cup D''_e$. The travel time from node i to j, where $i, j \in N$, is c_{ij}. Each delivery $i \in D$ has an associated service time d_i and a demand quantity q_i. For practical reasons, $d_i = 0$ for $i \in P_1 \cup P_2$. The allowed amount of nonservice time between the first and last delivery to customer u is the maximum idle time at u, and it cannot be greater than w_u.

We introduce two groups of binary variables to model the routing and scheduling decisions, such that the variables x_{ijk} equal 1 if vehicle k travels from node i to j, and 0 otherwise, and the variables z_{ij} equal 1 if delivery i is scheduled before delivery j at customer u, $i,j \in D'_u$. To control the timing requirements, we consider three sets of continuous nonnegative variables. Variables s_{ik} model the start of service at node i by vehicle k, and the variables $start_u$ ($last_u$) are the time at which the first (last) delivery at customer $u \in U$ starts (ends), respectively. Note that the routing and timing variables x_{ijk} and s_{ik} refer to deliveries, but synchronization takes place at the customer location, that is, it accounts for all deliveries received by a customer. For this reason, we need variables $start_u$ and $last_u$, and the customer sets. Variables z_{ij} are used to ensure that deliveries are served one at a time at the customer.

The objective function (1) consists of determining a set of vehicle routes with minimal total cost. Constraints (2) ensure that every delivery j that pertains to depot e is served by a vehicle $k \in V_e$. Routes must start and end at the right depot as imposed by constraints (3) and (4). Flow conservation is guaranteed by constraints (5). Constraints (6) and (7) ensure the delivery timing according to the routing variables. The total delivery in a route cannot exceed the vehicle capacity (constraints (8)), nor can vehicles return to their depots after the latest arrival time T (constraints (9)). Constraints (10)-(12) determine the time when service starts and ends at a customer. Each customer u has a predefined maximum idle time w_u which cannot be exceeded. This is expressed by constraints (13). Constraints (14) and (15) ensure that deliveries are served sequentially and constraints (16)-(20) impose domain conditions on the variables.

$$\text{minimize} \qquad \sum_{i \in N} \sum_{j \in N} \sum_{k \in V} x_{ijk} \cdot c_{ij}. \tag{1}$$

$$\sum_{k \in V_e} \sum_{i \in D''_e \cup \{e\}} x_{ijk} = 1 \qquad \forall e \in P_1,\ \forall j \in D''_e. \tag{2}$$

$$\sum_{j \in N_e} x_{ejk} = 1 \qquad \forall e \in P_1,\ \forall k \in V_e. \tag{3}$$

$$\sum_{i \in N_e} x_{i(n+p+e)k} = 1 \qquad \forall e \in P_1,\ \forall k \in V_e. \tag{4}$$

$$\sum_{j \in N_e} x_{ijk} = \sum_{j \in N_e} x_{jik} \qquad \forall e \in P_1,\ \forall k \in V_e,\ \forall i \in D''_e. \tag{5}$$

$$s_{ik} + d_i + x_{ijk} \cdot c_{ij} \leq s_{jk} + M(1 - x_{ijk}) \qquad \forall e \in P_1,\ \forall k \in V_e,\ \forall i \in N_e,\ \forall j \in N_e. \tag{6}$$

$$s_{ik} \leq M \sum_{j \in N_e} x_{ijk} \qquad \forall e \in P_1,\ \forall k \in V_e,\ \forall i \in D''_e \cup \{e\}. \tag{7}$$

$$\sum_{i \in D''_e} \sum_{j \in N_e \setminus \{e\}} x_{ijk} \cdot q_i \leq Q \qquad \forall e \in P_1,\ \forall k \in V_e. \tag{8}$$

$$s_{ik} + d_i + x_{i(n+p+e)k} \cdot c_{i(n+p+e)} \leq T + M(1 - x_{i(n+p+e)k}) \qquad \forall e \in P_1,\ \forall k \in V_e,\ \forall i \in N_e. \tag{9}$$

$$start_u \leq \sum_{k \in V} s_{ik} \qquad \forall u \in U,\ \forall i \in D'_u. \tag{10}$$

$$last_u \geq \sum_{k \in V} s_{ik} + d_i \qquad \forall u \in U,\ \forall i \in D'_u. \tag{11}$$

$$last_u - start_u \geq \sum_{i \in D'_u} d_i \qquad \forall u \in U. \tag{12}$$

$$last_u - start_u - \sum_{i \in D'_u} d_i \leq w_u \qquad \forall u \in U. \tag{13}$$

$$\sum_{k \in K} s_{ik} - \sum_{k \in K} s_{jk} + M z_{ij} \geq d_j \qquad \forall u \in U, i, j \in D'_u, i \neq j. \tag{14}$$

$$\sum_{k \in K} s_{jk} - \sum_{k \in K} s_{ik} + M(1 - z_{ij}) \geq d_i \qquad \forall u \in U, i, j \in D'_u, i \neq j. \tag{15}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in N, \forall k \in V. \tag{16}$$

$$z_{ij} \in \{0, 1\} \qquad \forall u \in U, \forall i, j \in D'_u. \tag{17}$$

$$s_{ik} \geq 0 \qquad \forall i \in N, \forall k \in V. \tag{18}$$

$$start_u \geq 0 \qquad \forall u \in U. \tag{19}$$

$$last_u \geq 0 \qquad \forall u \in U. \tag{20}$$

4 | FEASIBLE PLANS AT THE DELIVERY LOCATION

A solution to the VRPSCDL is feasible if it fulfills the "classical" VRP constraints, such as the vehicle capacity and the latest arrival time at the depot, and the synchronization constraints. This section focuses on the last ones, as they are specific features of the VRPSCDL. Synchronization constraints are difficult to handle due to the so-called interdependence problem [11], which implies that the feasibility of each route cannot be checked independently, as changing one route can make other routes infeasible. However, modern heuristic methods for VRPs require efficient move-evaluation techniques to explore neighborhoods [58]. For this reason, we evaluate three methods to guarantee the feasibility of the customer schedules.

We describe the self-imposed time windows approach in Section 4.1, in which the optimization algorithm manages the creation of and any changes to a feasible time interval during which all deliveries to a customer must be served, so that the solution remains feasible. An alternative approach would perform a complete rescheduling of the solution to check if a feasible schedule exists, as applied by Masson et al. [35] for the DARP with transfers. We explain how to adapt it to our optimization problem in Section 4.2. Finally, we consider two schemas with fixed time windows at customer locations; in the first one, the delivery interval is fixed independently by each customer, whereas the second one uses a clustering method that precedes the optimization and assign similar time windows to customers situated near each other, as we outline in Section 4.3.

4.1 | Self-imposed time windows

Self-imposed time windows define a time interval during which all deliveries to one customer must start. In the standard VRP, service can start as soon as the vehicle arrives at the customer's location. However, the VRPSCDL requires a compact delivery schedule for all vehicles serving the customer, and allowing any start time would lead to unfeasible solutions. If every customer had a time window of the exact required length (i.e., long enough to accommodate all deliveries and short enough to ensure that idle time does not exceed w_u), it would be possible to solve the problem as a VRP with time windows. Assigning regular time windows to solve the VRPSCDL would lead indeed to feasible solutions, though at the cost of exploring only a small fraction of the search space. For this reason, we create an artificial time window for every customer and let the optimization algorithm manage it during the search. We apply the term "self-imposed" to indicate that the windows are not enforced by the problem, but rather that the solution method iteratively modifies their value during the optimization process. Specifically, we update self-imposed time windows every time a delivery is inserted in or removed from the solution, which makes it suitable for our metaheuristic procedure.

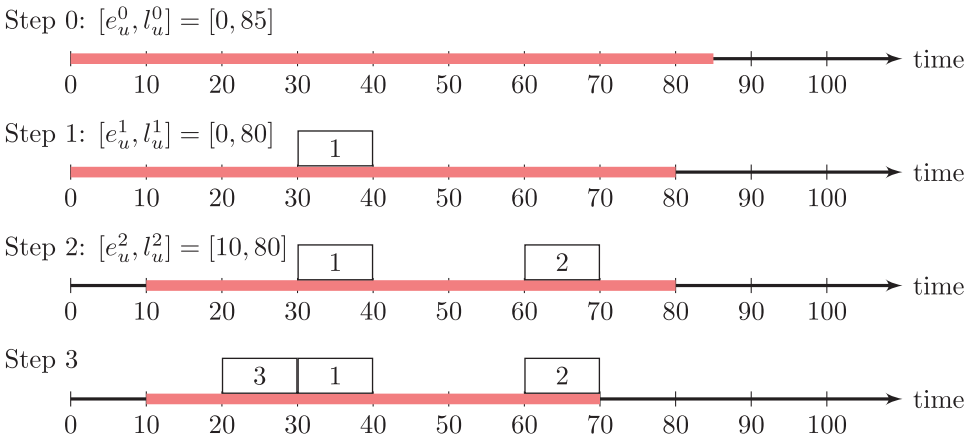

FIGURE 1 Example self-imposed time window update procedure. Three deliveries 1, 2, and 3 $\in D_u'$ are iteratively inserted in a solution. The earliest and latest start times are updated in each step. Start times before or after this interval are not allowed. The example assumes that the latest return time to the depot is $T = 100$, the service time of each delivery to customer u is $d_u = 10$, and the maximum idle time at u is $w_u = 30$. Traveling from u to the depot takes five time units [Colour figure can be viewed at wileyonlinelibrary.com]

As introduced in Section 3, every customer u has a predefined maximum idle time w_u. Although this value could be arbitrary, we choose to make it dependent on the deliveries expected. Specifically, $w_u = \alpha \cdot \sum_{i \in D_u'} d_i$, where $\alpha \geq 0$ can be used to control how much idle time is allowed. If $\alpha = 0$, delivery schedules are extremely compact, and the operation runs uninterruptedly from the first to the last delivery, whereas if $\alpha > 0$, some extra time is allowed, depending on the duration of the total service time at the customer. All deliveries to u thus must take place within a time interval of length $\tau_u = w_u + \sum_{i \in D_u'} d_i = (1 + \alpha) \cdot \sum_{i \in D_u'} d_i$, which includes both the total service time and the allowed idle time at the customer. In the following, we assume that all deliveries to a customer have the same service time and denote it d_u. If deliveries can have different service time durations, separate time windows for each delivery must be introduced.

The procedure that imposes time windows on customers works as follows: initially, we assume that the solution is empty, such that deliveries have not been included in any route. Each customer u has an initial time window $[e_u^0, l_u^0]$ that specifies when service at u can start. Time e_u^0 is the earliest departure time from the depot, whereas l_u^0 is the latest arrival time at the depot T minus the travel time between u and the depot minus the service time d_u. The initial time window determines that service can start at any time after the vehicle leaves the depot but not later than a time point that causes a late arrival to the depot. Then, the first delivery i to customer u is tested for insertion in the route of vehicle k. This first insertion is feasible as long as u is within driving distance of the corresponding depot (we assume they are); the delivery is scheduled at time s_{ik}, and its time window is subsequently updated as $[e_u^1, l_u^1] = [\max(e_u^0, s_{ik} + d_i - \tau_u), \min(l_u^0, s_{ik} + \tau_u - d_u)]$. The motivation is to ensure that the start time of the next delivery $j \in D_u'$ that will be inserted in the solution is feasible with respect to w_u. The earliest start time of j is then calculated, assuming that i was the last delivery served at customer u, so it takes the value of the end time of i ($s_{ik} + d_i$) minus τ_u time units (and in any case, it cannot be earlier than e_u^0). In a similar way, the latest start time of j assumes that i was the first delivery served at u.

In general, every time a delivery $i \in D_u'$ is evaluated for insertion at position p in the route of vehicle k, we check that the calculated start time s_{ik} is in the interval defined by the current time window of customer u. In addition, every other subsequent delivery j to customer v from position p onward in the route of vehicle k must start at a feasible new $\widehat{s}_{jk}$ with respect to the time window of customer v. For simplicity, we consider that a delivery $j \in D_v'$ following i in the route of vehicle k can only be rescheduled if the new $\widehat{s}_{jk}$ is feasible without rescheduling other deliveries $j \in D_v'$. If the delivery is finally inserted at that position, the time window of customer u must be updated according to $[e_u^{n+1}, l_u^{n+1}] = [\max(e_u^n, s_{ik} + d_i - \tau_u), \min(l_u^n, s_{ik} + \tau_u - d_u)]$. The self-imposed time windows of every customer visited in route k from position $p + 1$ onward is also updated.

This updating procedure is illustrated in Figure 1 using an example with $\alpha = 1$. We assume a maximum idle time of $w_u = 30$. All deliveries to u must take place within a time interval of length $\tau_u = 60$. The schedule of customer u is initially empty, and the self-imposed time window is determined by the latest return time to the depot $T = 100$, the service time $d_u = 10$, and the travel time from u to the depot, which takes five time units. Then at step 1, delivery 1 is inserted and scheduled in the solution at time $s_{1k1} = 30$, where 30 is the arrival time of vehicle k_1 serving delivery 1 to customer u. The arrival time is determined by the departure time from the previous node (depot or delivery) in the route and the travel time from that node to customer u. We assume that vehicles always leave the previous node as early as possible. The time window of u is updated using the procedure described previously. The earliest start time does not change, because it is still possible to start the service of subsequently inserted deliveries at time 0 without exceeding w_u (the actual start time cannot be 0 unless the travel time from the depot is 0, but this is not relevant for the time window update procedure). The latest start time needs to be updated; otherwise, we could schedule

a second delivery at time 85, which results in an idle time interval of length 45 between the end of delivery 1 and the beginning of the new delivery. Those 45 idle time units could be reduced to 35 after the insertion of the third delivery, but it would still exceed the maximum idle time $w_u = 30$. For this reason, l_u^1 takes the value $\min(85, s_{1k_1} + \tau_u - d_u) = \min(85, 30+60-10) = 80$. In a similar way, the earliest start time is updated after the insertion of delivery 2 at step 2. Once 2 has been scheduled at time 60, it is not possible to insert the next delivery, for example at time 5, because it would result in 35 idle time units. Finally, the third and last delivery is inserted at step 3. Because it is the last delivery to this customer, it is not necessary to update the time window any more.

Define the arrival time a_{ik} of vehicle k to serve delivery i as the departure time from the previous node in the route plus the required driving time to travel the distance from the previous node to the current one. The calculation of the start time s_{ik} accounts for potential waiting times that might arise in three situations: (1) early arrivals to the customer location (i.e., before the earliest possible start of service), (2) another delivery is being served at the time of arrival, or (3) another delivery j is scheduled to begin before the completion time of delivery i, such that $a_{ik} + d_i > s_{jk}$. The first case is straightforward, in that the earliest start time is determined by the self-imposed time window, whereas in the second one, because two deliveries cannot be attended simultaneously, it is necessary to wait at least until the delivery currently being served has been completed. The latter case is an explicit decision to avoid extensive rescheduling in the solution: when inserting a new delivery i, we choose not to reschedule other deliveries to the same customer u, even if it disrupts the first-come first-serve order. Every change to the customer's schedule has a potentially enormous impact on the solution' feasibility, affecting not only the routes of the rescheduled deliveries but also every other route that "crosses" (i.e., has at least one common customer) the affected ones. Thus it is not guaranteed that rescheduling a big part of the solution will keep the solution feasible, but checking the feasibility of each minor change can be very expensive. Finding a compromise is necessary, and for this reason, we choose not to modify the current scheduled deliveries. The worst case scenario is checking the feasibility of inserting a delivery at the first position of a route that already contains r deliveries. If there are, in the worst case, p deliveries per customer, the complexity of the feasibility algorithm is $\mathcal{O}(p \cdot (1 + r))$, because we have to compute a new start time for each delivery in the route, which might need to account for waiting times arising at each customer location.

In a similar way, self-imposed time windows must be updated whenever a delivery i is removed from the solution. In contrast with the procedure for inserting a customer, we do not need to consider the start and service time of i, because i is being removed, and it no longer determines the time window of u. In this case, the time window is modified only if i was the first or last delivery scheduled at u. Then, the (new) time window at u is $[e_u^{n+1}, l_u^{n+1}] = [s_{\eta\kappa} + d_\eta - \tau_u, s_{\iota\lambda} + \tau_u - \underset{\sim}{d}]$, where η and ι are the (new) last and first delivery scheduled at u, served by vehicles κ and λ, respectively. Furthermore, time windows of subsequent deliveries to i in the route are updated. Unlike insertion, removing a delivery from a route is always feasible. However, waiting times arise if a vehicle arrives at a customer's location before the earliest start time.

4.2 | Solution rescheduling

Another method to determine if a route plan has a feasible schedule consists of modeling the problem as a simple temporal problem (STP), which is a special case of the temporal constraint satisfaction problem that can be solved in polynomial time [8]. This approach has been applied by Masson et al. [35] to solve the DARP with transfers. The main difference is that our model does not impose any precedence on the synchronization points, whereas in the DARP with transfers, an arrival always happens before a departure. As a consequence, it is not possible to solve the STP as is, because the sequence in which deliveries arrive at customer locations is part of the feasible schedule we seek. It is thus necessary to make some assumptions regarding the order in which deliveries should arrive at customer locations. We evaluate two methods: single and multiple delivery sequences (MDS). The first initially determines a fixed sequence of deliveries for every customer location, and the second generates a pool of sequences for every customer location and chooses one that is compatible with the current solution. In addition, our approach differs in the shortest-path (SP) algorithm of choice. In preliminary experiments, we evaluated a number of SP algorithms and chose the best one with respect to runtime.

The rescheduling algorithm proceeds as follows: we perform a feasibility check every time we try to insert a delivery in the solution, which consists of solving the feasibility problem (FP), given the current set of routes and a chosen set of delivery sequences at customer locations. Because this problem is an STP, it can be represented as a distance graph and solved by proving its consistency (i.e., no negative cycles exist) with an SP algorithm. Section 4.2.1 explains how to model the FP as an STP and how to use the distance graph to check for feasible schedules. Section 4.2.2 details the determination of the delivery sequences.

4.2.1 | FP and consistency of the distance graph

The FP consists of determining a feasible schedule for a solution. We use a formulation similar to the model presented in Section 3. In this case, all x_{ijk} variables are known, as is which vehicle will service which customers. We want to determine feasible

values for the start times s_i for every delivery $i \in N$, as well as the begin and end times, $start_u$ and $last_u$, for every customer $u \in U$. If delivery j follows i in a route, j is the successor of i, and we denote it $\sigma(i)$. Similarly, delivery j is denoted $\tau(i)$ if j is scheduled after i at customer u.

Because all constraints specify a single interval, the FP is an STP. The STP considers a set of time point variables $\{X_1, \ldots, X_n\}$, and a set of constraints that restrict the permissible value for each temporal distance $X_j - X_i$ to $[a_{ij}, b_{ij}]$. Every constraint can be represented as $a_{ij} \leq X_j - X_i \leq b_{ij}$ or, alternatively, as a pair of inequalities $X_j - X_i \leq b_{ij}$ and $X_i - X_j \leq -a_{ij}$. Define the beginning of the planning horizon as $\emptyset$, so that $s_\emptyset$ is the earliest departure time for all vehicles. The FP can be defined by the following equations, all of which are formed as an inequality $X_j - X_i \leq b_{ij}$ or $X_i - X_j \leq -a_{ij}$.

$$s_i - s_{\sigma(i)} \leq -d_i - c_{i\sigma(i)} \quad \forall i \in N. \tag{21}$$

$$s_i - s_\emptyset \leq T - d_i - c_{i(n+p+e)} \quad \forall e \in P_1, i \in D. \tag{22}$$

$$start_u - s_i \leq 0 \quad \forall u \in U, \forall i \in D'_u. \tag{23}$$

$$s_i - last_u \leq -d_i \quad \forall u \in U, \forall i \in D'_u \tag{24}$$

$$start_u - last_u \leq -\sum_{i \in D'_u} d_i \quad \forall u \in U. \tag{25}$$

$$last_u - start_u \leq w_u + \sum_{i \in D'_u} d_i \quad \forall u \in U. \tag{26}$$

$$s_i - s_{\tau(i)} \leq -d_i \quad \forall u \in U, i \in D'_u. \tag{27}$$

Constraints (21) guarantee that the service at the successor of i in the route, $\sigma(i)$, does not start earlier than the start of service s_i, plus the service time d_i, plus the travel time between i and $\sigma(i)$. Constraints (22) state that delivery i cannot be served so late that the vehicle cannot return to the depot before T. Constraints (23)-(26) control the begin and times, and the maximum duration, of the self-imposed time window at customer u. Constraints (27) establish that the service at $\tau(i)$, scheduled after i at customer u, does not start earlier than the start of service s_i, plus the service time d_i.

The STP can be associated with a weighted directed graph, where each node is a time variable and each edge is a constraint between two variables. In such a distance graph, a constraint such as $X_j - X_i \leq b_{ij}$ would be represented by a labeled edge from i to j with weight b_{ij}. The STP is consistent if and only if the associated distance graph has no negative cycles.

Furthermore, SP algorithms can be used to identify negative cycles in a graph. Cherkassky et al. [3] show that though some SP algorithms seem more robust than others, no single algorithm dominates, so they recommend running several algorithms on a few problem instances and picking the best. We tested several SP algorithms: classical Bellman-Ford-Moore (BFM) [1, 20, 36], D'Escopo-Pape [39], Pallottino [38], BFM with subtree disassembly (BFCT) [56], linear-stack (LS), greedy LS, depth-first-stack, and a hybrid variant of BFCT [3].

With our optimization method, we build a distance graph, following the initial solution. This distance graph gets updated throughout the whole optimization process: Every time a delivery is inserted in or removed from the solution, the edges corresponding to the STP constraints are updated accordingly, such that changes in the solution are always reflected on the graph.

We perform a feasibility check every time a delivery is considered for insertion. If the SP algorithm finds no negative cycle in the distance graph, it can generate an as-early-as-possible schedule that consists of every start time s_i for each delivery $i \in D$. Each value s_i can be calculated by finding the shortest path $\Pi_{si,\emptyset}$ from node s_i to node $\emptyset$ in the distance graph. Define $\Pi_{si,\emptyset}$ as a sequence of edges in the distance graph. Then, the as-early-as-possible start time s_i can be calculated as $s_i = -\sum_{e \in \Pi_{s_i,0}} w_e$, where w_e is the weight of edge e.

4.2.2 | Determination of the serving order

We consider two strategies to decide the order for deliveries to customers. Assume $D'_u = \{\delta_1, \delta_2, \ldots, \delta_\Upsilon\}$ is the set of Υ deliveries to u, and π is a permutation such as π: $\{\delta_1, \delta_2, \ldots, \delta_\Upsilon\} \rightarrow \{\delta_1, \delta_2, \ldots, \delta_\Upsilon\}$. The serving order is determined by a sequence of deliveries such as $(\pi(\delta_1), \pi(\delta_2), \ldots, \pi(\delta_\Upsilon))$, where $\pi(\delta_1)$ preceding $\pi(\delta_2)$ means that delivery $\pi(\delta_1)$ is served before delivery $\pi(\delta_2)$ at customer u, such that $s_{\pi(\delta_1)} < s_{\pi(\delta_2)}$. Evaluating all possible sequences during the feasibility checks is not practical due to the large number of possibilities.

With single delivery sequences (SDS), the order is fixed for the whole optimization process for each customer. The sequences derive from a solution to the VRPSCDL, achieved with our initialization heuristic (Section 5.1). The initial solution is feasible and provides a sequence for each customer, which we then use during the optimization to test the solution's feasibility.

Alternatively, MDS can be used to add search diversification and further exploration of the search space. The algorithm keeps a pool of sequences for each customer. Every time a feasibility check is performed, the algorithm chooses one sequence for the corresponding customer u, while the schedules of the other customers in the solution remain intact. The chosen sequence

must be compatible with the current schedule of u, because it is possible that some deliveries in D'_u already exist in the solution. For example, if a customer expects three deliveries $\{\delta_1, \delta_2, \delta_3\}$ and two of them are already scheduled in the solution so that $s_{\delta_2} < s_{\delta_3}$, we can insert δ_1 and perform a feasibility check by choosing a delivery sequence such as $(\delta_1, \delta_2, \delta_3)$, $(\delta_2, \delta_1, \delta_3)$, or $(\delta_2, \delta_3, \delta_1)$, but not $(\delta_1, \delta_3, \delta_2)$, $(\delta_3, \delta_1, \delta_2)$, or $(\delta_3, \delta_2, \delta_1)$. To generate the initial pool of sequences, we build a large set of solutions using the initialization heuristic.

4.3 | Fixed time windows

In addition to self-imposed time windows and solution rescheduling, we evaluate fixed time windows at customer locations. Section 4.3.1 outlines a simple approach in which each customer independently determines its own time interval, during which all deliveries must be served. Section 4.3.2 introduces a method to cluster customers and assign similar time windows to all customers in one cluster.

4.3.1 | Customer-imposed time windows

In this approach, customers independently choose the most suitable time interval to be served. A fixed time window is generated for every customer u that requires two or more deliveries, whereas customers that get only one delivery can be served at any time during the work day. To ensure that a feasible solution exists, the choice of the earliest service time e_u of customer u must account for the total service duration and driving times from and back to the depot. Specifically, e_u is randomly chosen in the interval $\left[\underline{c}_u, T - \overline{c}_u - \sum_{i \in D'_u} d_i\right]$, where $\underline{c}_u$ is the largest travel time from any depot serving customer u to customer u, and $\overline{c}_u$ is the largest travel time from customer u to any depot serving u.

4.3.2 | Proximity-based clustered time windows

Customers are classified in k clusters according to agglomerative hierarchical clustering [5]; this step is performed only once before the optimization begins. The clustering algorithm proceeds as follows: initially, each customer represents a cluster. Then, as long as the number of clusters is larger than k, the two closest clusters are merged into a single one. The distance between two clusters is determined by the linkage criterion. We previously evaluated several well-known linkage criteria, such as single (minimum), complete (maximum), average (unweighted pair-group method with arithmetic mean, UPGMA), and centroid-linkage approaches, as well as a varying number of clusters between 1 and 4. We prefer UPGMA with three clusters, because it provides more compact and homogeneous clusters, each of which contains a similar number of locations and therefore leads to better results during the optimization. Figure 2 shows the results for a test instance with 150 locations and k equal to 3. Here, UPGMA produces three clusters of similar size, one for the city center of an urban area and two at the outskirts (west and east) of the city.

After k clusters have formed, we assign time windows to customers. The work day length T comprises k nonoverlapping intervals of equal length T/k. Each customer location u is assigned a random time window of length $\alpha \times \sum_{i \in D'_u} d_i$ within the interval. To make it more realistic, time windows start every hour and every half hour. Locations in the same cluster will thus have similar time windows.

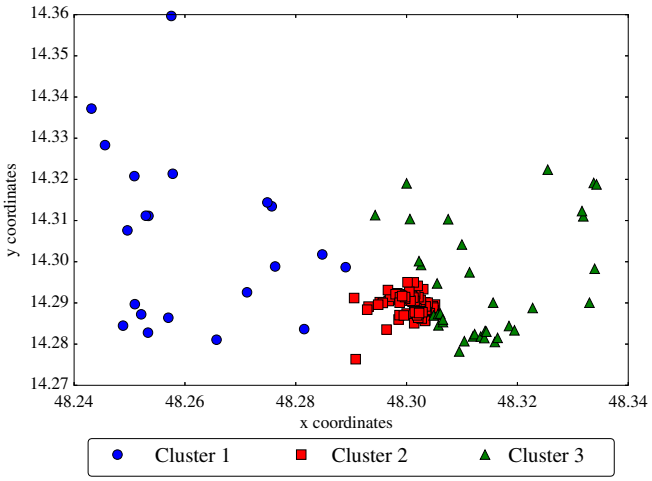

FIGURE 2 Locations in a test instance, classified in three clusters using hierarchical clustering with the UPGMA linkage criterion [Colour figure can be viewed at wileyonlinelibrary.com]

5 | ALNS FOR THE VRPSCDL

To solve the VRPSCDL, we develop an ALNS. The ALNS is a metaheuristic algorithm originally proposed by Ropke and Pisinger [47] for the pickup and delivery problem with time windows. It derives from the LNS introduced by Shaw [53], which itself is based on the ruin-and-recreate principle, such that the current solution is partially destroyed and then repaired in each iteration. If the new solution is better than the former one, the current solution is replaced by the new one. In addition, ALNS introduces several destroy and repair operators and an adaptive mechanism to decide which operators contribute most to solution improvement. In the following, we describe the components of our method.

5.1 | Solution initialization

A solution to the VRPSCDL consists of a set of VRP solutions (one for each depot). We build the initial solution by assigning a self-imposed time window to each customer (except when we use fixed time windows, because then the same window remains in place throughout the optimization). The initial self-imposed time window allows deliveries to be inserted during the complete work day. Then, a VRP solution using the Solomon C1 heuristic [54] is constructed for each depot, with the depots selected in random order. Every time a delivery is inserted in a route, the time window of the corresponding customer changes using the previously established method (Section 4.1). The initial solution then becomes the incumbent solution in the ALNS.

5.2 | Destroy operators

We now describe the destroy operators used in the algorithm. We first introduce two specific operators for the VRPSCDL, the *proximity-based customer removal* and the *schedule-based customer removal*. The four next ones (random, worst, related, and cluster operators) have been detailed by Ropke and Pisinger [47], and we adapt them to the VRPSCDL, by selecting one random depot and only remove deliveries related to that depot. According to our preliminary tests, this adaption yields better results than considering all deliveries in the solution for removal. The number of deliveries to remove, ξ, is randomly chosen in the range $[\rho, \phi]$. The random, worst, and related operators remove exactly ξ deliveries; the cluster and customer operators iterate until at least ξ deliveries have been removed.

The *proximity-based customer removal* randomly chooses one customer v from the set of customers with two or more deliveries $P = \{u : u \in U, |D'_u| \geq 2\}$ and removes all its deliveries from their corresponding routes. Then, if more deliveries have to be removed, a new customer v' is randomly selected among the nearest $\alpha_{prox} \cdot |P|$ customers to v. The parameter $\alpha_{prox} \in (0, 1]$ controls whether the algorithm removes only nearby customers (values of α_{prox} close to 0) or considers farther customers too (values of α_{prox} close to 1). All the deliveries of v' are removed from the solution, and the process repeats until ξ deliveries have been removed. Restricting the procedure to locations that expect more than one delivery worked better in our preliminary experiments. The motivation is to provide an operator that removes all deliveries taking place at a customer location, to allow them to be rescheduled later in the algorithm; it also seeks to remove deliveries that take place in close proximity, to make it more likely that they can be inserted in other routes if doing so reduces the solution cost.

The *schedule-based customer removal* works in a similar way: It selects a customer $v \in P$ and removes all deliveries in $D_{v'}$ from the solution. If more deliveries have to be removed, a new customer v' is randomly selected among the $\alpha_{sched} \cdot |P|$ customers with the most similar schedule to v. Schedule similarity is defined according to the customer's self-imposed time windows. If v has a current time window $[e_v, l_v]$ and v' has a current time window $[e_{v'}, l_{v'}]$, we compute the similarity between their schedules as $|e_v - e_{v'}| + |l_v - l_{v'}|$ The parameter $\alpha_{sched} \in (0, 1]$ controls how many different schedules are considered. All the deliveries of v' are removed from the solution, and the process repeats until ξ deliveries have been removed. Restricting the procedure to locations expecting more than one delivery worked better in our preliminary experiments. The motivation of this operator is to remove all deliveries taking place at a customer location, to facilitate their rescheduling later in the algorithm, but it also removes deliveries that happen in a similar time interval, so that they can be more easily rescheduled in a more time-saving fashion.

The *random removal* operator randomly selects ξ deliveries from a randomly selected depot.

The *worst removal* operator first randomly selects one depot e, then calculates a cost measure for every delivery $d \in D''_e$ associated with that depot, so that the ξ deliveries with higher costs are removed from the solution. The cost is defined as the difference between the cost of the current solution and the cost of the solution after removing the delivery. The operator is randomized to avoid always selecting the same deliveries, by iteratively drawing a random number r in the interval [0, 1) and removing the delivery with the $r^{p_{worst}} \cdot |D''_e|$-th highest cost, where $p_{worst} \geq 1$ controls the degree of randomization.

In the *related removal*, the objective is to remove similar deliveries that could be easily interchanged. We initially select a random depot and define the relatedness of two deliveries i and j as follows:

$$R(i,j) = \frac{1}{2}\left(\frac{c_{ij}}{c_{\max}} + \frac{|q_i - q_j|}{q_{\max}}\right), \tag{28}$$

where $c_{\max}$ is the maximum distance between any two pairs of customers and $q_{\max}$ is the maximum demand value of a delivery. The related removal is randomized using the procedure described for the worst removal operator.

Finally, the *cluster removal* heuristic is similar to the related removal, except that it attempts to remove related deliveries from a small number of routes. For that purpose, it uses Kruskal's algorithm for the minimum-spanning tree problem to partition selected routes into two clusters of deliveries, achieved by stopping the algorithm when two connected components remain. As noted for the previous operators, a depot is first selected randomly. Next, a random route with two or more deliveries is chosen and partitioned into two clusters. One of these clusters is randomly selected, and all its deliveries are removed from the route. If the number of removed deliveries is smaller than ξ, the procedure repeats and selects one of the removed deliveries, finds the closest delivery in a still unselected route that contains at least two deliveries, and applies the partition algorithm to this new route.

5.3 | Repair operators

The repair operators insert previously removed deliveries. They correspond closely to the greedy and regret heuristics of Ropke and Pisinger [47], such that a delivery can only be inserted in a route departing from the depot to which it belongs, and self-imposed time windows have to be updated after inserting every delivery.

The *greedy insertion* operator calculates the cost of inserting each delivery at the best insertion position (i.e., the position that increases the objective value least). It iteratively chooses the delivery with the lowest insertion cost and inserts it at the best possible position, until all deliveries have been inserted.

The *regret insertion* calculates a regret value for each delivery that has to be inserted, then inserts those deliveries with the highest regret values first. The regret value is the difference between the cost of insertion in the best route at its best position and the cost of insertion in the second best route at its best position. It can be generalized as a k-regret heuristic, where the k best possible insertions are considered.

5.4 | Acceptance criteria and adaptive mechanism

In the standard ALNS, new solutions are accepted only if they improve the solution cost. However, we observe that our algorithm can better escape local optima by sometimes accepting deteriorating solutions. Therefore, we always accept improving solutions; adopting the approach suggested by Polacek et al. [44] based on threshold accepting [13], we allow solutions with a higher cost if the number of iterations counted from the last accepted solution is equal to or greater than a certain threshold, and the cost is not higher than a certain percentage of the best solution cost.

In addition, ALNS assigns scores to operators to determine which contribute most to the solution improvement. Operators that provide better results are chosen more often in the subsequent iterations. Our preliminary experiments suggested that the adaptive layer provides better results than a pure LNS in our problem. Following Kovacs et al. [31, 32], we give scores $\psi_{(d,r)}$ to each pair (d, r) of destroy and repair operators, instead of assigning scores to each single operator. We consider all combinations of n_d destroy operators and n_r repair operators, so that the set of all operator pairs is $\Omega = \{(d, r)\colon 1 \le d \le n_d \wedge 1 \le r \le n_r\}$. We use three reward values, σ_1, σ_2, and σ_3, which are added to $\psi_{(d,r)}$ if (d, r) finds a new best solution, the new solution is better than the current one, or the new solution is accepted even if it is not better than the current one, respectively. As in previous work, we periodically recalculate the probability of choosing a destroy/repair pair using both historical information and new scores gathered in the previous iterations. At the beginning, all pairs are chosen with the same probability $\phi_{(d,r)} = 1/(n_d \cdot n_r)$, where n_d and n_r are the number of destroy and repair operators, respectively. The scores $\psi_{(d,r)}$ are initially zero. Then, information about the performance of the destroy/repair pairs is gathered during a certain time period, and the new weight $\chi_{(d,r)}$ of each pair is calculated as $\chi_{(d,r)} = (1 - p_{react}) \cdot \chi_{(d,r)} + p_{react} \cdot \frac{\psi_{(d,r)}}{\max(1, C_{(d,r)})}$, where p_{react} controls the influence of historical and new data, and $C_{(d,r)}$ is the number of times that (d, r) was used in the previous period. These weights are then used to calculate the new probabilities as $\varphi_{(d,r)} = \frac{\chi_{(d,r)}}{\sum_{(d',r')\in\Omega} \chi_{(d',r')}}$. The scores are set to zero for the new time period, and the process repeats periodically during the optimization.

6 | EXPERIMENTS

The ALNS was implemented in Java 1.7. Exact solutions were calculated, when required, using CPLEX 12.6.2. All experiments were carried out on a Xeon core at 2.50 GHz with 64 GB of RAM (shared with 20 other cores) and deactivated hyperthreading. Because the ALNS is not a deterministic algorithm, we always perform 10 runs, and report the average and best result. In the remainder of this section, we describe the algorithm parameterization, the test data, and the experimental results. We first examine the technical quality of ALNS by evaluating it on standard instances of the capacitated VRP and comparing it with an exact

solver on small VRPSCDL instances. Then, we answer managerial questions concerning the cost of enforcing synchronization, the benefits of using self-imposed time windows instead of other methods inspired from the literature, and the influence of the maximum allowed idle time.

6.1 | Parameter settings

The ALNS parameters were set during a preliminary testing phase and remain the same for all reported experiments. The threshold for the number of iterations without improvement after which a worse solution can be accepted is 1000. Worse solutions are then accepted only if the new cost is not higher than 20% of the best cost so far. The number of deliveries to remove is a random integer between ρ and ϕ. Following Pisinger and Ropke [43], we set ρ to $\min\{30, 0.1 \cdot |D''_e|\}$ and ϕ to $\min\{60, 0.4 \cdot |D''_e|\}$ for a selected depot e. Scores assigned to operator pairs depending on their success correspond to those used in previous work [47]. The reward granted to a destroy/repair operator pair (d, r) when a new overall best solution is found is $\sigma_1 = 33$. Likewise, the reward σ_2 if the new current solution is better than the previous one is $\sigma_2 = 9$, and unseen solutions that are accepted despite their cost being worse than the cost of the current solution are rewarded with $\sigma_3 = 13$. Operator probabilities are updated every 10 000 iterations. The parameter p_{react} that controls the influence of past and present information about the performance of destroy/repair pairs has been set to 0.5. Regarding the regret insertion, we use three regret operators with $k \in \{2, 3, 4\}$. The maximum runtime is 600 seconds, though the algorithm stops before if it finds the optimal solution (for instances whose optimum is known).

6.2 | Test instances

We evaluate the ALNS with the test instances defined by Christofides et al. [4] for the VRP, which feature between 50 and 199 customers. All of them have a maximum capacity constraint, and some also have a maximum tour length constraint (instances 6, 7, 8, 9, 10, 13, and 14).

Regarding the VRP with synchronization constraints, we generate an appropriate set of instances based on real-world data that describe the city of Linz, Austria [30]. We gathered these data through interviews of representatives of several business branches of companies, and they reveal the average number of deliveries per week and the average delivery size (i.e., customer demand), depending on the store size. Depots belonging to carriers were assigned locations in the industrial outskirts of the city, and we account for the freight vehicle load on roads coming into the urban area [10]. The complete data set contains 575 deliveries to 296 customer locations performed by eight carriers. The vehicle capacity is 18 units. Serving a delivery takes 15 minutes. The travel times between the locations are calculated using real floating car data, supplied by our industry partner. Times were provided in minutes with three decimal places, and we multiplied them by 1000 to achieve positive integers. Using these data, we generate multiple instances of different sizes.[1] Unless otherwise stated, $\alpha = 1$ for determining the allowed amount of idle time.

6.3 | Capacitated VRP

As noted, we start by comparing our ALNS with the best known solutions (BKS) of the benchmark set [4]. All instances are capacity-constrained. In addition, instances vrpnc6, vrpnc7, vrpnc8, vrpnc9, vrpnc10, vrpnc13, and vrpnc14 are also route-length-constrained. All but one of the BKS values are known to be optimal; in particular, Pecin et al. [41] solve instances with only capacity constraints to optimality, and Sadykov et al. [48] prove the optimality of instances with a maximum route length, except vrpnc13. Because ALNS does not need to update the customer time windows, the self-imposed time windows module is deactivated.

Table 1 shows the size and optimal cost for each instance, along with our average and best result, average and best gap, and the average time required to solve the instance. Regarding the average results, ALNS finds the optimal solution in all runs in 9 of 14 instances, with an average gap of only 0.14%. If we consider the best result obtained by the ALNS, we find the optimum in 11 of 14 instances (best gap is 0.08%). More than 500 seconds are needed for large instances (4, 5, 9, 10, and 13, with 120-199 customers), but smaller and medium-sized instances can be solved in 132 seconds (instance 3) or less (0.4-71 seconds).

These results can be compared with the aggregated results of an ALNS for the capacitated VRP, among other optimization problems, as reported by Pisinger and Ropke [43]. They evaluate their ALNS with two different stop conditions, 25 000 and 50 000 evaluations (ALNS-25K and ALNS-50K), and report the average and the best of 10 runs on a Pentium 4 with 3 GHz. Their average optimal gaps are 0.39% and 0.31% for ALNS-25K and ALNS-50K, respectively, which we improve on, in that we achieve an average gap of 0.14%, at the cost of longer running times (228.81 seconds for ALNS vs 55.8 and 105 seconds,

[1] All instances can be downloaded from https://plis.univie.ac.at/research/test-instances/

TABLE 1 Numerical results for the ALNS compared with the best known solutions (BKS) in previous literature

#	*m*	BKS	Avg.	Best	Avg. gap	Best gap	Time (s)
vrpnc1	50	**524.61***	**524.61**	**524.61**	0	0	3.61
vrpnc2	75	**835.26***	**835.26**	**835.26**	0	0	70.89
vrpnc3	100	**826.14***	**826.14**	**826.14**	0	0	132.80
vrpnc4	150	**1028.42***	1030.58	**1028.42**	0.21	0	577.03
vrpnc5	199	**1291.29***	1302.93	1297.58	0.90	0.49	600.00
vrpnc6	50	**555.43***	**555.43**	**555.43**	0	0	2.50
vrpnc7	75	**909.68***	**909.68**	**909.68**	0	0	17.06
vrpnc8	100	**865.94***	**865.94**	**865.94**	0	0	50.18
vrpnc9	150	**1162.55***	1163.14	**1162.55**	0.05	0	524.63
vrpnc10	199	**1395.85***	1406.25	1404.56	0.75	0.62	600.00
vrpnc11	120	**1042.11***	**1042.11**	**1042.11**	0	0	22.64
vrpnc12	100	**819.56***	**819.56**	**819.56**	0	0	0.53
vrpnc13	120	**1541.14**	1542.59	1541.21	0.09	0.00	600.00
vrpnc14	100	**866.37***	**866.37**	**866.37**	0	0	1.41
Total	-	13 664.36	13 690.59	13 679.42	-	-	-
Avg.	-	-	-	-	0.14	0.08	228.81

Bolded entries indicate the BKS has been found; asterisks denote proven optima.

respectively). Their best gaps are 0.15% and 0.11%, which we also improve by attaining 0.08%, again with longer runtimes (2288.1 seconds compared with 559.8 and 1050 seconds, respectively). Our ALNS is thus comparable to other ALNS algorithms in prior literature.

6.4 | Comparison of ALNS and CPLEX solver

In Table 2 we compare the results obtained by ALNS with those of the CPLEX solver. We use a set of 20 small instances (0-19) with 10-40 deliveries and 1-3 depots. In addition, we include the results for three larger instances (20-22) for which the exact solver could find at least one feasible solution. CPLEX stops after a maximum runtime of 24 hours and can use up to 20 GB RAM. For each instance, we show the number of depots p, number of deliveries n, number of customers m, and the results obtained by CPLEX and ALNS. In the case of CPLEX, we report the cost (preceded by an asterisk if the optimal solution has been proven) and two time values (in seconds). The first value is the total runtime, and the second one is the time required to find the optimal solution. For the ALNS, we report the average and best cost, the average runtime (in seconds), and the gap with the average and best cost found by CPLEX.

CPLEX solves 9 of 23 instances to optimality (0-5 and 10-12) and also finds the BKS in 5 additional instances (6, 8, 13, 14, and 15). We find that ALNS is always able at least to match the results obtained by CPLEX, and it does so considerably faster than the exact solver. In particular, CPLEX requires a total runtime of 35 789.91 seconds, while the ALNS needs 8404.38 seconds (it is thus 4.26 times faster). If we consider only those instances whose optimum is known, CPLEX needs 1128.20 seconds to find the optimum, while the ALNS needs 4.38 seconds (257.89 times faster). Regarding instances for which CPLEX finds the BKS, ALNS also finds it in every case and faster; specifically, ALNS runs for a maximum runtime of 600 seconds in each run, but CPLEX needs 1000-3000 seconds, depending on the instance. Finally, for the remaining nine instances, ALNS can improve the solution cost, by 6%-84% in the average and the best case (21% on average). The total is improved by more than 43% by using ALNS rather than CPLEX, across both the average and the best results.

6.5 | Cost of synchronization

Synchronization enforcement can lead to higher costs, because longer routes might be necessary to meet the additional constraints. For this reason, it is critical to determine how much costs increase in such a centralized system compared with a decentralized environment in which each carrier can independently optimize its own deliveries. We compare the centralized and decentralized strategies using the ALNS and solving larger instances of the problem (50-300 deliveries and 1-6 depots).

Table 3 presents the results of the centralized and decentralized strategies. For each instance, we list the number of depots, number of deliveries, and number of customer locations, as well as the average number of routes, average cost and average total idle time obtained by each strategy. In addition, it reveals the increase (positive value) or decrease (negative value) in the number of routes per depot and percentages of cost and idle time in the centralized strategy compared with the decentralized one. Finally, we also report two intervals for the minimum and maximum increase in the number of routes and percentage of

TABLE 2 Numerical results for the ALNS compared with CPLEX for small VRPSCDL instances with $\alpha = 1$

				Exact			ALNS				
#	*p*	*n*	*m*	Cost	*t*	*t* to best	Avg.	Best	*t*	Avg. gap	Best gap
0	1	10	10	**41.67***	41.30	12.74	**41.67**	**41.67**	0.01	0.00	0.00
1	1	10	10	**55.20***	1496.98	423.39	**55.20**	**55.20**	0.01	0.00	0.00
2	2	10	6	**89.38***	11.21	4.70	**89.38**	**89.38**	0.08	0.00	0.00
3	2	10	6	**79.71***	10.68	4.58	**79.71**	**79.71**	0.03	0.00	0.00
4	2	15	10	**64.98***	2801.15	213.35	**64.98**	**64.98**	0.90	0.00	0.00
5	2	15	10	**93.11***	1829.48	114.48	**93.11**	**93.11**	2.10	0.00	0.00
6	2	20	13	**117.06**	1165.15	-	**117.06**	**117.06**	600.00	0.00	0.00
7	2	20	13	**77.94**	1287.67	-	**77.94**	**77.94**	600.00	0.00	0.00
8	2	30	23	134.66	583.11	-	**109.93**	**109.93**	600.00	−18.36	−18.36
9	2	30	22	136.52	1255.26	-	**127.76**	**127.76**	600.00	−6.42	−6.42
10	3	15	6	**125.36***	101.94	15.66	**125.36**	**125.36**	0.05	0.00	0.00
11	3	15	6	**83.21***	21.00	6.33	**83.21**	**83.21**	0.63	0.00	0.00
12	3	20	8	**124.08***	6386.82	332.96	**124.08**	**124.08**	0.57	0.00	0.00
13	3	20	8	**109.07**	2827.41	-	**109.07**	**109.07**	600.00	0.00	0.00
14	3	25	10	**115.60**	1625.28	-	**115.60**	**115.60**	600.00	0.00	0.00
15	3	25	10	**144.78**	3015.77	-	**144.78**	**144.78**	600.00	0.00	0.00
16	3	30	13	129.57	1817.25	-	**113.57**	112.30	600.00	−12.35	−13.33
17	3	30	13	170.73	2539.86	-	**170.58**	**170.58**	600.00	−0.08	−0.08
18	3	40	17	292.74	994.60	-	**227.53**	226.27	600.00	−22.28	−22.71
19	3	40	17	288.51	869.19	-	**182.70**	182.62	600.00	−36.67	−36.70
20	1	50	50	269.38	1618.96	-	**199.20**	198.60	600.00	−26.05	−26.28
21	1	50	50	210.45	2831.57	-	**145.80**	**145.80**	600.00	−30.72	−30.72
22	2	100	75	2300.63	658.30	-	**358.20**	349.80	600.00	−84.43	−84.80
Total	-	-	-	5254.34	35 789.91	-	**2956.42**	2944.82	8404.38	−43.73	−43.95

Bolded entries highlight the best solutions; asterisks indicate that the optimal solution was proven by the exact solver.

cost for each depot. Total idle time is measured as the sum of the idle time at every customer location. All instances require synchronization except the two smallest ones, which have one depot each.

For all instances with two or more depots, the decentralized strategy obtains lower costs at the expense of higher idle times. Specifically, the cost of synchronization can be estimated as 2%-7% for instances with 150 or fewer deliveries, and 7%-16% for those greater than 200 (9.18% on average). However, idle times are always reduced by more than 50% in the synchronized approach (64.76% on average). As a result, assuming slightly longer routes can drastically reduce idle time, thus allowing compact delivery arrivals to the customer and providing carriers with a time interval during which delivery will take place. The impact grows as the instance size increases, which could be due not only to the greater number of deliveries, but also the greater number of deliveries per customer.

Although the number of routes or vehicles is not part of the objective function, it provides additional insight into how solutions change when synchronization is enforced. The total number of routes, as the cost, increases in the centralized approach (344.90) compared with the decentralized one (328.00). However, the average number of additional routes per depot is very small for most instances, and it is close to 0 even for many large ones. Regarding the impact on particular service providers, our results confirm that the number of routes does not increase by more than 1 in the worst case. Cost intervals, however, are more diverse. For small instances (less than four depots), interval lengths are small, which means that costs increase in a similar way for each provider. For larger instances (four or more depots), cost increases are often unevenly distributed, so that the cost of synchronization is much higher on some providers than on others. Balance could be achieved by adding a fairness objective or constraint, which is not in the scope of this paper.

In order to highlight the effect of synchronization on the solutions, Figure 3 shows the detailed results for two specific instances. For each instance and each depot, we show the average increase in the number of routes Δ_{NR} and the average cost increase percentage $\Delta_{Cost}\%$ in the centralized approach with respect to the decentralized one. These values are averages over 10 runs. If the bar does not appear in the plot, the corresponding value is 0. In instance 28 (Figure 3A), the number of routes does not change for 3 of 4 depots, and it only increases slightly for depot 2. The cost increases in a similar percentage for most depots, that is, the cost of synchronization is similar for all logistics service providers. In instance 36 (Figure 3B), however, the required number of routes increases for all depots except one, whereas the cost increases drastically for all depots. Moreover, although

TABLE 3 Cost of synchronization using a centralized optimization strategy compared with a decentralized strategy

				Decentralized			Centralized							
#	*P*	*n*	*m*	NR	Cost	IT	NR	Cost	IT	$\Delta_{NR/p}$	$\Delta_{Cost}\%$	$\Delta_{IT}\%$	$I_{\Delta NR/p}$	$I_{\Delta Cost\%}$
20	1	50	50	5.00	198.92	0.00	5.00	198.90	0.00	0.00	−0.01	0.00	[0.00, 0.00]	[−0.01, −0.01]
21	1	50	50	5.00	145.84	0.00	5.00	145.84	0.00	0.00	0.00	0.00	[0.00, 0.00]	[0.00, 0.00]
22	2	100	69	10.00	400.68	1551.82	10.00	412.23	439.60	0.00	2.88	−71.67	[0.00, 0.00]	[2.70, 3.13]
23	2	100	75	9.00	345.43	884.84	9.60	358.10	288.85	0.30	3.67	−67.36	[0.00, 0.60]	[3.38, 4.00]
24	2	100	65	10.00	398.38	2173.06	10.00	411.48	536.05	0.00	3.29	−75.33	[0.00, 0.00]	[3.08, 3.52]
25	3	150	75	14.00	538.29	3042.93	14.10	571.09	875.40	0.03	6.09	−71.23	[0.00, 0.10]	[4.64, 9.08]
26	3	150	75	15.00	553.29	2406.84	15.00	576.95	735.59	0.00	4.28	−69.44	[0.00, 0.00]	[3.24, 4.87]
27	3	150	75	14.00	481.01	4084.51	14.00	508.73	826.21	0.00	5.76	−79.77	[0.00, 0.00]	[4.27, 8.53]
28	4	200	66	18.00	644.70	4596.71	18.20	692.99	1635.15	0.05	7.49	−64.43	[0.00, 0.20]	[6.59, 8.73]
29	4	200	66	16.00	636.09	5276.29	17.80	726.79	1620.65	0.45	14.26	−69.28	[0.00, 0.80]	[8.13, 19.24]
30	4	200	66	18.00	601.48	4711.42	18.60	655.55	1550.08	0.15	8.99	−67.10	[0.00, 0.60]	[7.10, 14.75]
31	4	200	66	18.00	621.77	4335.56	18.00	668.87	1588.58	0.00	7.58	−63.36	[0.00, 0.00]	[4.16, 12.31]
32	5	250	83	20.00	703.31	5519.62	21.10	782.84	2157.15	0.22	11.31	−60.92	[0.00, 0.50]	[5.99, 16.07]
33	5	250	83	22.00	794.62	5589.22	23.10	871.81	2106.16	0.22	9.71	−62.32	[0.00, 0.90]	[5.94, 16.35]
34	5	250	83	21.00	734.58	4829.23	21.10	797.23	2139.73	0.02	8.53	−55.69	[0.00, 0.10]	[6.53, 10.51]
35	5	250	83	20.00	868.79	5973.13	21.90	964.69	2182.94	0.38	11.04	−63.45	[0.00, 1.00]	[5.69, 13.91]
36	6	300	100	23.00	972.84	6965.07	26.00	1119.85	2580.82	0.50	15.11	−62.95	[0.00, 1.00]	[7.89, 25.13]
37	6	300	100	23.00	981.08	7722.31	24.90	1132.60	2525.41	0.32	15.44	−67.30	[0.00, 1.00]	[10.74, 27.05]
38	6	300	100	24.00	859.21	5817.96	24.80	986.18	2669.39	0.13	14.78	−54.12	[0.00, 0.50]	[8.19, 23.66]
39	6	300	100	23.00	945.31	6775.22	26.70	1068.98	2531.11	0.62	13.08	−62.64	[0.00, 1.00]	[10.45, 16.77]
Total	-	-	-	328.00	12 425.62	82 255.74	344.90	13 651.70	28 988.87	0.22	9.87	−64.76	-	-

NR and IT stand for number of routes and idle time, respectively. Intervals $I_{\Delta_{NR}}$ and $I_{\Delta_{Cost}\%}$ report the smallest and largest changes in Δ_{NR} and $\Delta_{Cost}\%$, considering all depots in the instance. Instances were solved assuming $\alpha = 1$.

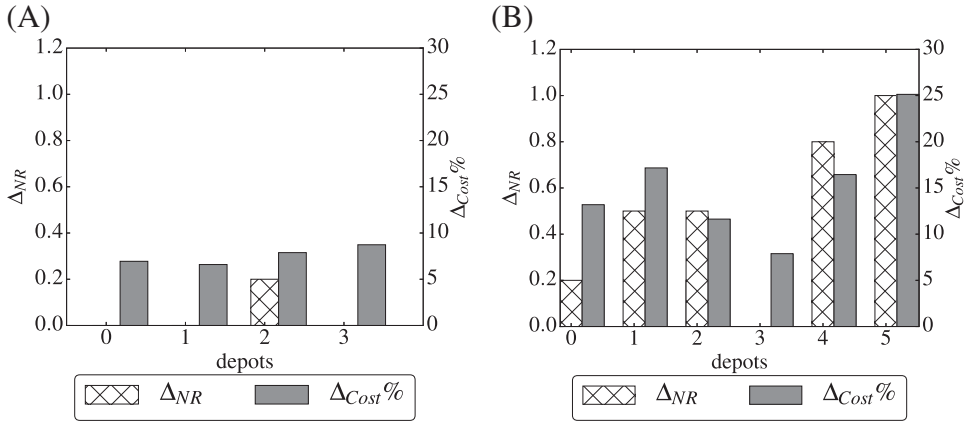

FIGURE 3 Impact of synchronization on particular service providers using instances 28 (left) and 36 (right). (A) Instance 28. (B) Instance 36

TABLE 4 Numerical results for the self-imposed time windows strategy compared with the rescheduling strategies

	ALNS-SITW		ALNS-SPFA-SDS		ALNS-SPFA-MDS	
Runtime	**Avg.**	**Best**	**Avg. gap**	**Best gap**	**Avg. gap**	**Best gap**
600 s	13 651.69	13 402.29	15.74	13.07	21.43	17.19
8 h	13 223.72	13 067.97	1.21	−0.11	0.43	−0.69
24 h	13 189.53	13 055.10	0.45	−0.82	−0.79	−2.14

A positive (negative) gap means that the results are worse (better) than those of ALNS-SITW. Instances were solved assuming $\alpha = 1$.

Δ_{NR} takes values smaller than or equal to 1 for all depots, Δ_{Cost}% is not balanced. Differences between the two instances are probably due to the larger instance size and higher number of synchronization requirements in instance 36, in which up to six vehicles must perform their deliveries around the same time. Similar results can be observed when comparing other small and medium-sized instances with large ones.

6.6 | Self-imposed time windows versus solution rescheduling

In this section, we compare the results obtained by ALNS using self-imposed time windows (ALNS-SITW) with those of ALNS using the feasibility checks to find a feasible schedule, both with simple delivery sequences (ALNS-SPFA-SDS) and multiple delivery sequences (ALNS-SPFA-MDS).

In a preliminary phase, we evaluated several SP algorithms that are used to prove the consistency of the STP, as explained in Section 4.2.1. The results showed that the stack-based algorithm LS detected negative cycles faster than the rest of the options in our particular instances, and the optimization algorithm therefore obtained better results (it could perform more iterations in the same runtime). For this reason, we use LS in the remainder of this section. Although LS is similar to BFM, it uses stacks instead of queues.

The solver that relies on rescheduling methods is slower than the self-imposed time windows approach, because the computational effort associated with the feasibility checks is relatively large. For this reason, we evaluate the algorithms with increasing runtimes (600 seconds, 8 hours, and 24 hours). Table 4 presents the aggregated average and best results for ALNS-SITW, ALNS-SPFA-SDS, and ALNS-SPFA-MDS. In the case of the solution rescheduling algorithms, we show the relative gap to the results obtained by ALNS-SITW. Detailed results can be found in the appendix (Tables A1-A3).

When the available runtime is short, ALNS-SITW provides the best results (both the average and the best gap are well above 10%). Also, ALNS-SPFA-SDS obtains better results than the ALNS-SPFA-MDS in this case, because managing multiple schedules is more time consuming than using a fixed one during the search. The gap closes when more computation time is available. For runtimes of 8 hours, ALNS-SITW still obtains better average results (gaps around 1%), but the ALNS-SPFA methods find better average best results (small gaps of −0.11% and −0.69%), such that they appear able to find very good solutions in single runs, but they do not perform better in the average case. In addition, ALNS-SPFA-MDS outperforms ALNS-SPFA-SDS in both average and best results. For even longer runtimes (24 hours), ALNS-SITW is slightly better than the ALNS-SPFA-SDS in the average case (gap of 0.45%), but ALNS-SPFA-MDS now issues the best average results (gap of −0.79%).

TABLE 5 Results obtained by ALNS with fixed time windows. The time windows were assigned assuming $\alpha = 1$ for all instances

	Customer-imposed time windows			Proximity-based clustered time windows		
#	Gap_{Avg}	Gap_{Best}	IT%	Gap_{Avg}	Gap_{Best}	IT%
20	−0.02	0.00	0.00	0.00	0.00	0.00
21	0.00	0.00	0.00	0.00	0.00	0.00
22	11.12	10.56	−73.53	9.35	7.85	−73.40
23	11.71	9.56	−71.31	11.29	11.84	−71.93
24	13.15	11.12	−78.50	10.79	5.15	−71.89
25	19.56	18.84	−56.34	15.34	13.97	−40.10
26	21.89	18.22	−49.10	17.13	14.73	−43.23
27	25.53	23.24	−45.53	17.35	16.38	−38.27
28	25.80	23.48	−38.86	15.26	13.36	−36.66
29	26.11	26.38	−41.36	17.11	20.05	−30.47
30	24.82	21.42	−40.88	15.96	14.36	−25.63
31	27.12	26.36	−35.84	17.47	13.35	−27.42
32	24.90	22.32	−29.12	16.21	15.47	−23.59
33	22.44	18.93	−34.25	10.91	9.80	−27.46
34	26.89	26.47	−32.45	19.74	19.69	−22.13
35	19.37	15.10	−32.74	11.49	10.61	−29.82
36	19.95	19.33	−27.92	14.29	13.17	−17.11
37	19.67	19.34	−25.56	9.36	8.21	−22.93
38	22.66	21.82	−31.65	15.00	14.12	−22.25
39	19.10	18.24	−25.91	11.03	9.78	−22.26
Total	21.15	19.51	−35.58	13.71	12.55	−28.34

The ALNS-SITW results are not included, because they are available in Table 3.

6.7 | Self-imposed versus fixed time windows

This section compares self-imposed time windows with fixed time windows, using ALNS as the optimization algorithm. Fixed time windows are known in advance and do not change during the execution of the program. As explained in Section 4.3, we evaluated both random time windows and proximity-based clustered time windows.

Table 5 reports, for each fixed time window, the average cost, the best cost, and the total idle time, all relative to the results of the self-imposed time windows approach. These latter values are available in Table 3. The methods using fixed time windows always obtain worse cost results than the self-imposed time windows algorithm, for both the average (gaps are 21.15% and 13.71%) and the best (gaps are 19.51% and 12.55%) results. The algorithm using hierarchical clustering obtains better results than the random time windows approach, as we expected, because customers located in close proximity have similar time windows.

Because the allowed idle time is modeled as a constraint, the actual value is not crucial as long as the solution remains feasible. However, idle time is much lower in both fixed time windows approaches, compared with the self-imposed time windows method. Adapting the time window during the search makes better use of the available time, stretching the time window as much as possible, and therefore larger idle times arise. Thus, our results show that total idle time is 25.56%-78.50% smaller in the customer-imposed time windows approach than in the self-imposed time windows, excluding instances 20 and 21 that do not require synchronization. The method using proximity-based clustered time windows obtains slightly smaller idle time reductions in general than the former one (17.11%-73.40%, depending on the instance).

6.8 | Influence of the maximum idle time

Next we evaluate the effect of varying the maximum idle time on solution quality. As explained in Section 4.1, we set the maximum idle time allowed at a customer location depending on the total service time duration at the customer and a parameter α.

Table 6 summarizes the results for $\alpha \in \{0, 0.25, 0.50, 0.75, 1, 2, 3, 4, \infty\}$, where 0 means that all deliveries must uninterruptedly take place one after the other, and ∞ means that any amount of idle time is allowed. The values in between define the width of the time interval during which all deliveries must take place. For each value of α, the table reports the aggregated results over the 20 large instances used in the previous sections. As expected, idle times are smaller when little or no idle time is allowed (smaller values of α), but it also results in an increased number of routes and higher costs, because more vehicles

TABLE 6 Influence of the maximum allowed idle time

α	Num. routes	Avg.	Best	Total IT	Max. IT
0.00	520.00	19 614.00	18 294.00	0.00	0.00
0.25	426.60	17 364.60	16 372.20	8931.00	281.40
0.50	364.90	14 675.40	14 157.00	16 369.20	562.80
0.75	353.20	14 013.00	13 611.00	22 350.00	837.60
1.00	344.90	13 651.80	13 402.20	28 987.80	1116.60
2.00	332.70	12 924.60	12 756.60	55 201.20	2206.80
3.00	328.50	12 632.40	12 540.60	74 150.40	3171.60
4.00	328.20	12 548.40	12 481.20	85 415.40	3813.00
∞	328.10	12 493.80	12 437.40	97 892.40	4579.80

A value of $\alpha \in [0, 4]$ allows a maximum idle time that equals α times the total service time expected at the customer location; $\alpha = \infty$ means that idle time is not restricted.

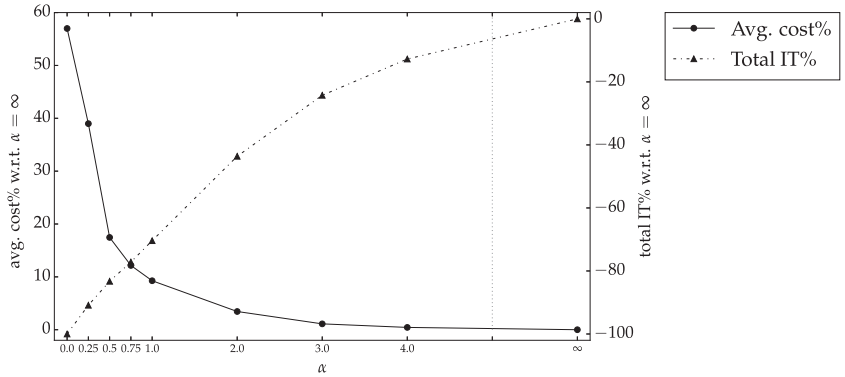

FIGURE 4 Percentage of the average cost and total idle time w.r.t. $\alpha = \infty$ for increasing values of α

are needed to meet the tighter problem constraints. For example, the number of routes, average cost, and best cost increase by around 30%, 39%, and 32%, respectively, when $\alpha = 0.25$ with respect to $\alpha = \infty$, whereas the total and maximum idle time are reduced by 91% and 94%, respectively.

Figure 4 shows the aggregated results of the average cost and the total idle time as a percentage of the results obtained with $\alpha = \infty$. The largest differences regarding the solution cost emerge for small values of α; if $\alpha < 1$, the average cost is 10%-60% higher than the case when no synchronization is enforced. However, values of $\alpha \in [1, 4]$ lead to solutions whose cost is closer to the model with no synchronization; for example, if $\alpha = 2$, the average cost is 3.45% higher than the nonsynchronized algorithm. Considering the idle times, the largest differences also happen when $\alpha = 0$, because idle times are nonexistent in that case. However, the differences in the total idle time do not decay as fast as the cost diminishes. For example, allowing a cost increase of 3.45% for $\alpha = 2$ leads to an idle time reduction of 43.61%, such that it is possible to obtain solutions with relatively small idle times without large sacrifices in the solution cost.

7 | CONCLUSION

We have introduced and modeled a VRPSCDL. We argue that this approach is relevant to solve some shortcomings in urban freight transportation. In particular, we address applications in which customers await several deliveries shipped by different carriers, and these deliveries must take place in a compact way. This situation arises whenever shipments are needed at roughly the same time, for example, the construction industry, housing and decoration logistics, and event management.

Synchronization constraints are hard to handle by optimization algorithms due to the interdependence problem, which means that a change in a route can make other routes infeasible. Although VRPs with synchronization constraints have been studied in the past, there is a research gap in the synchronization of arbitrary number vehicles at the customer location. Moreover, we are among the first to address a new type of operation synchronization constraint, which imposes that all deliveries must happen within a non-predefined time interval. In addition, we also take into account resource synchronization, so that two deliveries cannot be served at the customer at the same time.

To tackle this problem, we investigate three methods to ensure that the solution remains feasible during optimization. The first consists of creating and managing a self-imposed time window for each delivery location. The second method uses a

state-of-the-art approach to determine if a plan has a feasible schedule, and uses it to reschedule deliveries at the customer location. The third simply assigns fixed time windows to customers. We present an ALNS metaheuristic for the VRPSCDL that uses one of these three methods to provide a solution with compact schedules at the delivery location, and provide two destroy operators for this problem.

When we evaluate our algorithm using standard benchmark instances for the VRP, the average gap from the optimum is 0.14%. We also show that the ALNS is equally effective and much faster than an exact commercial solver for small instances of the VRPSCDL. We evaluate the cost of synchronization, estimated as 9.13% on average for the set of test instances. Moreover, self-imposed time windows always provide better results than solution rescheduling (unless long runtimes of up to 24 hours are granted) and fixed time windows. Finally, we show that it is possible to achieve a good trade-off between cost and idle time by changing the length of the self-imposed time window. In addition, we provide new instances for the VRPSCDL based on realistic data, which we make available online.

ACKNOWLEDGMENTS

Financial support from the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged. Thanks are due to Risc Software GmbH for providing the travel times based on real floating car data.

ORCID

Briseida Sarasola https://orcid.org/0000-0002-0677-3026
Karl F. Doerner https://orcid.org/0000-0001-8350-1393

REFERENCES

[1] R. Bellman, *On a routing problem*, Q. Appl. Math. **16** (1958), 87–90.
[2] T.-S. Chang and H.-M. Yen, *City-courier routing and scheduling problems*, Eur. J. Oper. Res. **223** (2012), 489–498.
[3] B.V. Cherkassky, L. Georgiadis, A.V. Goldberg, R.E. Tarjan, and R.F. Werneck, *Shortest-path feasibility algorithms: An experimental evaluation*, J. Exp. Algorithmics **14** (2010), 1–37.
[4] N. Christofides, A. Mingozzi, and P. Toth, "*The vehicle routing problem*," *Combinatorial Optimization*, chapter 11, N. Christofides, A. Mingozzi, C. Sandi, and P. Toth (eds), John Wiley & Sons, Chichester, 1979.
[5] R.M. Cormack, *A review of classification*, J. R. Stat. Soc. Ser. A: Gen. **134** (1971), 321–367.
[6] T.G. Crainic, "City logistics," *INFORMS TutORials in Operations Research*, 2014, pp. 181–212.
[7] T.G. Crainic, N. Ricciardi, and G. Storchi, *Models for evaluating and planning city logistics systems*, Transp. Sci. **43** (2009), 432–454.
[8] R. Dechter, I. Meiri, and J. Pearl, *Temporal constraint networks*, Artif. Intell. **49** (1991), 61–95.
[9] A. Dohn, E. Kolind, and J. Clausen, *The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach*, Comput. Oper. Res. **36** (2009), 1145–1157.
[10] DORIS, 2015. DORIS – Digitales Oberösterreichisches Raum-Informations-System. https://doris.ooe.gv.at/
[11] M. Drexl, *Synchronization in vehicle routing – A survey of VRPs with multiple synchronization constraints*, Transp. Sci. **46** (2012), 297–316.
[12] M. Drexl, A generic heuristic for vehicle routing problems with multiple synchronization constraints, Technical report, Gutenberg School of Management and Economics, Johannes Gutenberg-Universität Mainz, 2014.
[13] G. Dueck and T. Scheuer, *Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing*, J. Comput. Phys. **90** (1990), 161–175.
[14] M.J.R. Ebben, M.C. van der Heijden, and A. van Harten, *Dynamic transport scheduling under multiple resource constraints*, Eur. J. Oper. Res. **167** (2005), 320–335.
[15] J.F. Ehmke and A.M. Campbell, *Customer acceptance mechanisms for home deliveries in metropolitan areas*, Eur. J. Oper. Res. **233** (2014), 193–207.
[16] J.F. Ehmke, A. Steinert, and D.C. Mattfeld, *Advanced routing for city logistics service providers based on time-dependent travel times*, J. Comput. Sci. **3** (2012), 193–205.
[17] J.F. Ehmke, A.M. Campbell, and T.L. Urban, *Ensuring service levels in routing problems with time windows and stochastic travel times*, Eur. J. Oper. Res. **240** (2015), 539–550.
[18] J.F. Ehmke, A.M. Campbell, and B.W. Thomas, *Optimizing for total costs in vehicle routing in urban areas*, Transp. Res. E: Logist. Transp. Rev. **116** (2018), 242–265.
[19] P. Eveborn, P. Flisberg, and M. Rönnqvist, *Laps care – An operational system for staff planning of home care*, Eur. J. Oper. Res. **171** (2006), 962–976.
[20] L.R. Ford, Network flow theory, Technical report, The Rand Corporation, 1956.
[21] P. Grangier, M. Gendreau, F. Lehuédé, and L.-M. Rousseau, *A matheuristic based on large neighborhood search for the vehicle routing problem with cross-docking*, Comput. Oper. Res. **84** (2017), 116–126.
[22] A. Grimault, N. Bostel, and F. Lehuédé, *An adaptive large neighborhood search for the full truckload pickup and delivery problem with resource synchronization*, Comput. Oper. Res. **88** (2017), 1–14.
[23] T. Gschwind and S. Irnich, *Effective handling of dynamic time windows and its application to solving the dial-a-ride problem*, Transp. Sci. **49** (2015), 335–354.
[24] N. El Hachemi, M. Gendreau, and L.-M. Rousseau, *A heuristic to solve the synchronized log-truck scheduling problem*, Comput. Oper. Res. **40** (2013), 666–673.

[25] V.C. Hemmelmayr, J.-F. Cordeau, and T.G. Crainic, *An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics*, Comput. Oper. Res. **39** (2012), 3215–3228.
[26] C. Hempsch and S. Irnich, "*Vehicle routing problems with inter-tour resource constraints*," *The Vehicle Routing Problem: Latest Advances and New Challenges*, B. Golden, S. Raghavan, and E. Wasil (eds), Springer US, Boston, MA, 2008, pp. 421–444.
[27] H. Hojabri, M. Gendreau, J.-Y. Potvin, and L.-M. Rousseau, *Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints*, Comput. Oper. Res. **92** (2018), 87–97.
[28] I. Ioachim, J. Desrosiers, F. Soumis, and N. Bélanger, *Fleet assignment and routing with schedule synchronization constraints*, Eur. J. Oper. Res. **119** (1999), 75–90.
[29] O. Jabali, R. Leus, T. Van Woensel, and T. de Kok, *Self-imposed time windows in vehicle routing problems*, OR Spectr. **37** (2015), 331–352.
[30] F.J. Koll and M. Teufel, *City Logistik; Warenflussanalyse einer Mikrostadt zur Erstellung eines fiktiven City Logistik Datensatzes*, Johannes Kepler Universität Linz, Diplomarbeit, 2014.
[31] A.A. Kovacs, S.N. Parragh, K.F. Doerner, and R.F. Hartl, *Adaptive large neighborhood search for service technician routing and scheduling problems*, J. Sched. **15** (2012), 579–600.
[32] A.A. Kovacs, S.N. Parragh, and R.F. Hartl, *A template-based adaptive large neighborhood search for the consistent vehicle routing problem*, Networks **63** (2014), 60–81.
[33] A.A. Kovacs, B.L. Golden, R.F. Hartl, and S.N. Parragh, *The generalized consistent vehicle routing problem*, Transp. Sci. **49** (2015), 796–816.
[34] D.S. Mankowska, F. Meisel, and C. Bierwirth, *The home health care routing and scheduling problem with interdependent services*, Health Care Manag. Sci. **17** (2014), 15–30.
[35] R. Masson, F. Lehuédé, and O. Péton, *The dial-a-ride problem with transfers*, Comput. Oper. Res. **41** (2014), 12–23.
[36] E.F. Moore, *The shortest-path through a maze*, Proceedings of the International Symposium on the Theory of Switching, 1959, pp. 285–292.
[37] J. Muñuzuri, R. Grosso, P. Cortés, and J. Guadix, *Estimating the extra costs imposed on delivery vehicles using access time windows in a city*, Comput. Environ. Urban Syst. **41** (2013), 262–275.
[38] S. Pallottino, *Shortest-path methods: Complexity, interrelations and new propositions*, Networks **14** (1984), 257–267.
[39] U. Pape, *Implementation and efficiency of Moore-algorithms for the shortest route problem*, Math. Program. **7** (1974), 212–222.
[40] S.N. Parragh and K.F. Doerner, *Solving routing problems with pairwise synchronization constraints*, Cent. Eur. J. Oper. Res. **26** (2018), 443–464.
[41] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa, *Improved branch-cut-and-price for capacitated vehicle routing*, Math. Program. Comput. **9** (2017), 61–100.
[42] V. Pillac, M. Gendreau, C. Guéret, and A.L. Medaglia, *A review of dynamic vehicle routing problems*, Eur. J. Oper. Res. **225** (2013), 1–11.
[43] D. Pisinger and S. Ropke, *A general heuristic for vehicle routing problems*, Comput. Oper. Res. **34** (2007), 2403–2435.
[44] M. Polacek, R.F. Hartl, K. Doerner, and M. Reimann, *A variable neighborhood search for the multi depot vehicle routing problem with time windows*, J. Heuristics **10** (2004), 613–627.
[45] N.-H. Quttineh, T. Larsson, K. Lundberg, and K. Holmberg, *Military aircraft mission planning: A generalized vehicle routing model with synchronization and precedence*, EURO J. Transp. Logist. **2** (2013), 109–127.
[46] L.B. Reinhardt, T. Clausen, and D. Pisinger, *Synchronized dial-a-ride transportation of disabled passengers at airports*, Eur. J. Oper. Res. **225** (2013), 106–117.
[47] S. Ropke and D. Pisinger, *An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows*, Transp. Sci. **40** (2006), 455–472.
[48] R. Sadykov, E. Uchoa, and A. Pessoa, A bucket graph based labeling algorithm with application to vehicle routing, Cadernos do LOGIS 2017/7, Universidade Federal Fluminense, 2017.
[49] M.A. Salazar-Aguilar, A. Langevin, and G. Laporte, *Synchronized arc routing for snow plowing operations*, Comput. Oper. Res. **39** (2012), 1432–1440.
[50] V. Schmid, K.F. Doerner, R.F. Hartl, and J.-J. Salazar-González, *Hybridization of very large neighborhood search for ready-mixed concrete delivery problems*, Comput. Oper. Res. **37** (2010), 559–574.
[51] J. Schönberger, Hybrid search and the dial-a-ride problem with transfer scheduling constraints, Technical report no. 3/2015, Diskussionsbeiträge aus dem Institut für Wirtschaft und Verkehr, Working Paper, 2015.
[52] S. Shao, G. Xu, M. Li, and G.Q. Huang, *Synchronizing e-commerce city logistics with sliding time windows*, Transp. Res. E: Logist. Transp. Rev. **123** (2019), 17–28.
[53] P. Shaw, *Using constraint programming and local search methods to solve vehicle routing problems*, M. Maher and J.-F. Puget (eds.), Principles and Practice of Constraint Programming – CP98, Lecture Notes in Computer Science, vol. 1520, Springer, Berlin Heidelberg, 1998, pp. 417–431.
[54] M.M. Solomon, *Algorithms for the vehicle routing and scheduling problems with time window constraints*, Oper. Res. **35** (1987), 254–265.
[55] SUGAR, City logistics best practices: A handbook for authorities, Technical report, 2011.
[56] R.E. Tarjan, Shortest-paths, Technical report, AT&T Bell Laboratories, 1981.
[57] R. Vahrenkamp, *25 years city logistics: Why failed the urban consolidation centres?*, Conference Logistik Management, 2013.
[58] T. Vidal, T.G. Crainic, M. Gendreau, and C. Prins, *A unified solution framework for multi-attribute vehicle routing problems*, Eur. J. Oper. Res. **234** (2014), 658–673.
[59] L. Wen and R. Eglese, *Minimum cost VRP with time-dependent speed data and congestion charge*, Comput. Oper. Res. **56** (2015), 41–50.
[60] S.X. Xu, S. Shao, T. Qu, J. Chen, and G.Q. Huang, *Auction-based city logistics synchronization*, IISE Trans. **50** (2018), 837–851.

How to cite this article: Sarasola B, Doerner KF. Adaptive large neighborhood search for the vehicle routing problem with synchronization constraints at the delivery location. *Networks.* 2020;75:64–85. https://doi.org/10.1002/net.21905

APPENDIX A: A SELF-IMPOSED TIME WINDOWS VERSUS SOLUTION RESCHEDULING

Tables A1–A3 give further detail about the results reported in Section 6.6 for varied running times (600 seconds, 8 hours, and 24 hours, respectively). Each table reports, for each instance, the number of depots, deliveries, and customer locations; the average and the best cost of ALNS-SITW; and the average and best gap obtained by each rescheduling strategy w.r.t. ALNS-SITW.

TABLE A1 Results for the self-imposed time windows strategy compared with the rescheduling strategies running for 600 s

				ALNS-SITW		ALNS-SPFA-SDS		ALNS-SPFA-MDS	
#	*P*	*n*	*m*	Avg	Best	Avg. Gap	Best Gap	Avg. Gap	Best Gap
20	1	50	50	198.90	**198.77**	0.49	0.07	0.27	0.09
21	1	50	50	**145.84**	**145.84**	2.03	**0.00**	1.69	**0.00**
22	2	100	69	412.23	408.13	0.50	0.18	1.88	0.30
23	2	100	75	358.10	350.09	1.21	1.36	1.80	1.68
24	2	100	65	411.48	408.59	1.23	−0.02	1.56	−0.34
25	3	150	75	571.09	564.02	5.25	0.92	6.36	4.65
26	3	150	75	576.96	570.73	2.16	0.54	2.65	1.71
27	3	150	75	508.73	501.06	4.24	3.74	5.51	2.15
28	4	200	66	692.99	673.43	12.96	8.86	23.99	18.69
29	4	200	66	726.79	698.16	13.43	10.98	20.07	15.22
30	4	200	66	655.55	647.19	11.70	10.30	13.04	10.53
31	4	200	66	668.87	650.64	8.45	7.87	13.19	8.91
32	5	250	83	782.84	767.19	23.60	19.52	34.67	28.50
33	5	250	83	871.81	861.68	18.95	15.79	25.99	19.40
34	5	250	83	797.23	778.00	20.97	18.72	30.34	23.15
35	5	250	83	964.69	945.94	13.77	11.18	19.47	16.13
36	6	300	100	1119.85	1103.97	24.96	21.55	35.14	30.82
37	6	300	100	1132.60	1115.43	22.21	17.89	29.84	22.13
38	6	300	100	986.18	965.37	28.34	22.00	35.51	31.95
39	6	300	100	1068.98	1048.08	26.94	26.57	30.61	26.65
Total	-	-	-	13 651.69	13 402.29	15.74	13.07	21.43	17.19

Bolded entries indicate that the best known solution has been found.

TABLE A2 Results for the self-imposed time windows strategy compared with the rescheduling strategies running for 8 h

				ALNS-SITW		ALNS-SPFA-SDS		ALNS-SPFA-MDS	
#	*p*	*n*	*m*	Avg	Best	Avg. Gap	Best Gap	Avg. Gap	Best Gap
20	1	50	50	198.79	**198.77**	0.05	**0.00**	0.06	**0.00**
21	1	50	50	**145.84**	**145.84**	**0.00**	**0.00**	**0.00**	**0.00**
22	2	100	69	407.86	406.36	−0.57	−1.10	−0.51	−0.86
23	2	100	75	353.85	349.03	−1.10	−1.21	−1.26	−0.59
24	2	100	65	409.15	407.79	−0.71	−0.99	−1.26	−1.80
25	3	150	75	562.08	557.04	−1.04	−0.97	−0.92	−1.92
26	3	150	75	566.13	562.64	−0.14	−0.78	−0.49	−0.72
27	3	150	75	499.90	496.57	0.48	0.03	−0.63	−1.03
28	4	200	66	673.42	669.22	2.22	−0.55	1.26	0.41
29	4	200	66	688.34	671.22	−0.46	0.38	−1.44	−2.20
30	4	200	66	633.14	623.90	−0.39	−0.52	0.30	−1.30
31	4	200	66	642.05	639.04	0.48	**−1.16**	1.03	−0.58
32	5	250	83	746.07	733.20	5.21	2.25	1.48	0.70
33	5	250	83	850.36	841.59	0.58	−0.20	−0.16	−2.07
34	5	250	83	768.38	761.97	3.86	0.95	2.60	1.23
35	5	250	83	938.59	916.41	−0.16	−0.07	−2.22	−0.57
36	6	300	100	1077.06	1065.93	2.76	0.32	1.51	−0.80
37	6	300	100	1088.83	1073.92	1.98	0.04	0.47	−0.14
38	6	300	100	946.80	928.77	0.61	−1.75	1.23	−1.09
39	6	300	100	1027.09	1018.78	3.10	1.08	2.92	−0.58
Total	-	-	-	13 223.72	13 067.97	1.21	−0.11	0.43	−0.69

Bolded entries indicate that the best known solution has been found.

TABLE A3 Results for the self-imposed time windows strategy compared with the rescheduling strategies running for 24 h

				ALNS-SITW		ALNS-SPFA-SDS		ALNS-SPFA-MDS	
#	*p*	*n*	*m*	Avg	Best	Avg. Gap	Best Gap	Avg. Gap	Best Gap
20	1	50	50	**198.77**	**198.77**	0.06	**0.00**	0.08	**0.00**
21	1	50	50	**145.84**	**145.84**	**0.00**	**0.00**	**0.00**	**0.00**
22	2	100	69	408.29	405.56	−0.98	**−0.74**	−1.01	**−0.74**
23	2	100	75	353.96	348.72	−1.99	−0.82	−1.96	**−0.91**
24	2	100	65	408.21	406.95	−0.75	−1.18	−1.33	**−1.72**
25	3	150	75	562.90	557.86	−2.01	−2.47	−2.03	**−2.67**
26	3	150	75	564.19	561.26	−0.42	−1.08	−0.60	**−1.39**
27	3	150	75	497.65	494.25	−0.11	**−0.87**	−0.82	−0.73
28	4	200	66	674.56	670.82	1.91	−0.85	0.93	**−2.01**
29	4	200	66	691.81	670.62	−2.12	−0.24	−2.84	**−2.80**
30	4	200	66	629.98	621.89	−1.48	−1.27	−1.68	**−2.39**
31	4	200	66	639.50	635.25	0.23	**−0.57**	0.16	−0.32
32	5	250	83	746.19	732.34	1.62	1.14	0.56	**−0.27**
33	5	250	83	847.79	840.59	0.04	−1.23	−0.73	**−2.95**
34	5	250	83	769.31	761.96	1.45	**−1.00**	1.15	−0.30
35	5	250	83	925.15	915.49	−0.28	−0.93	−0.69	**−2.78**
36	6	300	100	1077.42	1067.24	2.82	**−3.14**	−1.19	−3.04
37	6	300	100	1084.38	1071.79	0.14	−1.54	−1.24	**−4.25**
38	6	300	100	943.44	932.86	1.65	0.21	−2.86	**−4.81**
39	6	300	100	1020.20	1015.04	2.79	1.28	0.67	**−1.36**
Total	-	-	-	13 189.53	13 055.10	0.45	−0.82	−0.79	−2.14

Bolded entries indicate that the best known solution has been found.