The stochastic weekly vehicle routing truck driver scheduling problem with time windows

Erasmus University Rotterdam

Erasmus School of Economics

Master's thesis

Econometrics and Management Science

Analytics and Operations Research In Logistics

Author	
Frijlink, Tenzin	611871

Supervisors	
Spliet, Remy (1st)	Erasmus University
Wagenvoort, Mette (2nd)	Erasmus University
Bliki, Arne	Conundra
Heene, Tom	Conundra

Abstract

Carriers face a twofold optimization problem: planning vehicle routes and workforce scheduling. In practice, this problem is often solved sequentially, first the routes are computed and then drivers are scheduled manually, based on a single forecast scenario. When the actual demand becomes known, this procedure is repeated. This leads to undesirable situations, as drivers could have to work on different days or at different hours than originally planned. Therefore, the stochastic weekly vehicle routing truck driver scheduling problem (SWVRTDSP) with time windows is introduced. The objective is to create a weekly driver schedule, based on a set of forecast scenarios, that minimizes the expected routing and scheduling costs via an integrated approach. The driver schedule conforms to (EC) No. 561/2006 and takes into account drivers' working preferences. To solve the SWVRTDSP, an adaptive large neighbourhood search algorithm is used. The performance of the algorithm is tested on instances from literature and a real-life instance from the Netherlands, containing 671 daily customers. Additionally, a problem-specific neighbourhood is developed that effectively generates a robust driver schedule based on forecast scenario(s). Results show that fewer drivers are required if the driver pool consists of either highly available or highly skilled drivers, compared to a balanced driver pool. Moreover, it is shown that a multi-scenario integrated solving approach results in the best driver schedule, which is both robust and reduces the objective value by 22.94% for the real-life instance.

July 14, 2023

Contents

1 Introduction

Currently, 15% of truck driver positions are unfilled and expected to triple in the coming years (IRU, 2022). Therefore, it is becoming more important for companies to use drivers more efficiently and adhere to their working preferences. For carriers, which supply stores for multiple retailers, there is a twofold optimization problem: planning vehicle routes and workforce scheduling. Currently, this is performed sequentially, each retailer, based on a single forecast, solves the Vehicle Routing Problem with Time Windows (VRPTW) and supplies these routes to the carrier. Drivers are then scheduled manually based on the planned routes from the retailers. When the actual demand for retailers becomes available, the VRPTW is solved again and drivers are rescheduled by hand. This leads to undesirable situations, as drivers could have to work on different days or at different hours than originally scheduled or preferred. Moreover, it could lead to drivers working a significant amount of overtime each week.

Therefore, this report explores the Stochastic Weekly Vehicle Routing and Truck Driver Scheduling Problem (SWVRTDSP) with time windows, denoted as SWVRTDSP. The aim is to create a robust schedule for drivers, such that drivers can rely on the schedule provided in advance, whilst keeping routing and scheduling costs to a minimum. Moreover, the schedule considers the drivers' working preferences, to both use drivers more efficiently and increase driver satisfaction.

The main contributions of this report are fivefold. First, the VRTDSP is extended to include stochastic demand, in this case by adding multiple forecast scenarios. Second, the VRTDSP is extended to include weekly planning, which considers driver preferences, and skills and conforms to (EC) No. 561/2006. Third, a state-of-the-art Adaptive Large Neighbourhood Search (ALNS) algorithm with problem-specific neighbourhoods is developed to construct routes and create driver schedules for the SWVRTDSP. Multiple experiments are performed regarding forecast scenarios, different solving approaches, varying cost prioritization, different driver pools, and neighbourhood performance. Fourth, the adaptive element of ALNS is further adjusted, such that not only the neighbourhood performance is taken into account, but also the applicability for the current solution and time complexity of the neighbourhood. Fifth and last, it is shown that a multi-scenario integrated approach results in a robust driver schedule.

The next section, Section 2, reviews the state-of-the-art literature. In Section 3, the SWVRTDSP is formally defined. In Section 4, the ALNS method is introduced, neighbourhoods are elaborated on and a sequential method is presented. Section 5 introduces the problem instances and shows the findings of the computational study. Lastly, Section 6 provides conclusions of this report and recommends future research directions.

2 Related Work

In this section related works are discussed. First, works on vehicle routing with time windows are discussed in Section 2.1. Second, the truck driver scheduling problem is discussed in Section 2.2. Third and last, the vehicle routing truck driver scheduling problem is discussed in Section 2.3.

2.1 Vehicle Routing Problem with Time Windows

The VRPTW is known to be an NP-hard problem since it contains the VRP as a special case, which is NP-hard (Laporte & Nobert, 1987). The VRPTW, if solved exactly, is often modelled as a multi-commodity network flow problem (Salani & Vacca, 2011) or a set covering formulation (Bredstrom & Ronnqvist, 2007) in combination with branch and price (Feillet, 2010). However, Özarık et al. (2021) state that exact algorithms have a well-documented highly-variable performance on most vehicle routing variants, supported by independent research from Uchoa et al. (2017). Therefore, many authors use heuristics which yield near-optimal solutions within relatively short computing times (Özarık et al., 2021) and perform well for larger instances (Uchoa et al., 2017). Castillo-Salazar et al. (2016) come to a similar conclusion after reviewing VRPTW state-of-the-art.

Ropke & Pisinger (2006) first introduced the Adaptive Large Neighbourhood Search (ALNS) which iteratively, via partly destroying and repairing the routes, tries to improve the solution. This metaheuristic is highly popular (Windras Mara et al., 2022) since it allows for diversification, flexibility and is easy to adjust (Sacramento et al., 2019). Windras Mara et al. (2022) surveyed over 250 scientific publications on ALNS and showed that there is increasing use of ALNS in recent years. Moreover, it is noted that more emphasis should be placed on the adaptive mechanism.

However, limited research has been performed into the VRPTW with stochastic demand (Zhang et al., 2016). Often two-stage stochastic programming is used, e.g. by Chang (2011) or Lei et al. (2011), where initial routes are constructed and then adjusted based on actual demand. Another approach is to generate a finite set of possible demand realizations, based on a probability distribution for customer demand, and construct routes based on these realizations (Spliet & Desaulniers, 2015). Boujlil & Lissane Elhaq (2020), after reviewing 50 papers, advice to develop metaheuristics to solve the VRPTW with stochastic demand. The review mainly highlights the research by Lei et al. (2011) who use ALNS to find robust solutions.

2.2 Truck Driver Scheduling Problem

Since April 2007 there are regulations regarding the Hours of Service (HoS) for truck drivers in the form of regulation (EC) No. 561/2006 (European Parliament, 2006). Goel (2008) deemed the workforce scheduling problem adhering to these regulations as the Truck Driver Scheduling Problem (TDSP). The TDSP is a special case of the workforce scheduling problem and as a consequence is NP-hard (Lau, 1996). Moreover, Goel (2010) first found that the legislative constraints made the truck driver scheduling problem much more complex to solve than regular workforce scheduling. Recent research by Sartori et al. (2022) also shows that solving the TDSP results in lengthy computational times.

Van Den Bergh et al. (2013) reviewed over 300 scientific publications on workforce scheduling and found that exact approaches using a set covering formulation were most common. Alternatively, genetic algorithms or tabu search are often used to improve the solution from construction heuristics. The literature survey performed by Castillo-Salazar et al. (2016) confirms these findings. Regardless, the vast majority of literature ignores all types of uncertainty as well as working preferences (Van Den Bergh et al., 2013).

In the TDSP the problem at hand is to assign routes, with a given time duration, to available drivers. The structure of this problem is similar to bin-packing, where items of a certain size need to be allocated to as few bins as possible. Witteman et al. (2021) showed that, in the field of aircraft maintenance, a bin-packing-based approach results in small optimality gaps and limited computational time. Marzouk & Kamoun (2020) explored a bin-packing-based approach for the nurse-to-patient assignment problem, which takes into account nurse skills, different shifts and different assignment zones. The authors found that the bin-packing-based heuristic provided flexible and feasible solutions, regardless of the shift or assignment zone.

2.3 Vehicle Routing and Truck Driver Scheduling Problem

As both the TDSP and VRPTW are NP-hard, the integrated problem is definitively NP-hard. Castillo-Salazar et al. (2016) explore the Workforce Scheduling and Routing Problem (WSRP), which is similar to the VRTDSP, but also adheres to working preferences. In their literature review Castillo-Salazar et al. (2016) find that often exact approaches are used for the routing, but heuristics for the workforce scheduling. Alternatively, as the literature review by Pereira et al. (2020) confirms, (meta)heuristics are used to solve the WSRP as a whole. Pereira et al. (2020) further extend the WSRP to include multi-period workforce scheduling with dependent tasks. They use ant colony optimization, which finds the same solutions as an exact approach in a fraction of the time for most instances.

Perumal et al. (2021) researched a problem similar to the VRTSP, where bus drivers and buses are scheduled simultaneously using ALNS. They find that improvements up to 4.37%, in terms of operational costs, can be achieved using ALNS over sequential methods. Mor et al. (2022) find that significant savings can be achieved by using heuristic methods, compared to current practice, in the VRTDSP for long haul transport complying with HoS regulations. Wen et al. (2011) extend the VRTDSP to weekly planning under HoS regulations, time windows and vehicle size. They use a multilevel variable neighbourhood search heuristic with five large neighbourhoods in combination with diversification and intensification, where first the problem size is reduced through node aggregation. Wen et al. (2011) found that, based on real-life data, their heuristic for the VRTDSP outperformed industry standards both in terms of total travelled distance and number of vehicles used.

3 Problem Definition

Consider the complete graph $G = (V, A)$, where $V = \{0, 1, ..., n + 1\}$ correspond to the set of all locations and $V' = \{1, ..., n\}$ the set of customers, where vertices 0 and $n + 1$ represent the depot. More specifically, vertex 0 is where the route starts and vertex $n + 1$ is where the route ends. Let c_{ij} and t_{ij} represent the cost and time respectively to travel from location i to j. The total travel costs are defined as C_t. The service time for each customer is included in the travel time. Define e_i and l_i as the earliest and latest allowed time to arrive at customer i, thus leading to the time window $[e_i, l_i]$. Each customer i has stochastic demand $\mathbf{z}_i$ with a known distribution, where $\mathbf{z}_i$ is non-negative and is less than Y.

There is an unlimited fleet of K trucks available with capacity Q^k, note that $Y \leq \min Q^k$. If customer i can be served by truck k, then $v_i^k = 1$. To account for weekly planning, dummy vertices are introduced, so each customer has a separate vertex for every day of the week, where $m_{dj} = 1$ if customer j needs to be served on day $d \in D$.

Furthermore, there is a sufficiently large set of P drivers available with certain driving skills and working preferences for the week. These preferences relate to working days, total working hours, starting time and end time. If driver p can and wants to drive truck k then $w_p^k = 1$. Each driver has a preferred starting time a_p, end time b_p and weekly working hours h_p. Moreover, if driver p prefers to work on day d then $o_{pd} = 1$. The costs associated with violating these preferences are $c_{tardiness}$ which relate to starting earlier or later than preferred per time unit, c_{over} per time unit of overtime and c_{day} per non-preferred scheduled day respectively. The sum of these costs over the drivers is called the preference violation costs C_p.

Additionally, there are HoS regulations, by European Parliament (2006), that apply to each driver. Since short-haul transport is considered in this report, only constraints with regard to consecutive working time and rest periods apply:

HR.1 Being scheduled for between 6 and 9 hours consecutively requires a break of at least 30 minutes

HR.2 Being scheduled for more than 9 hours consecutively requires a break of at least 45 minutes

HR.3 It is allowed to take the breaks in 15-minute segments

HR.4 A minimum daily rest of 11 hours, which can be reduced to 9 hours, but no more than three times between any 2 weekly rest periods

A route is feasible if (i) the capacity of the truck is not exceeded, (ii) the time windows of served customers are respected, and (iii) the customers are served by the correct truck.

A schedule is feasible if (a) HR.1 through HR.4 are satisfied, (b) all customers are served, (c) all routes are assigned to drivers and (d) the driver can drive the trucks corresponding to the assigned routes.

To model demand uncertainty, consider a finite set of demand scenarios Ω and corresponding probability p_ω such that $\sum_{\omega\in\Omega} p_\omega = 1$. Each customer i has demand d_i^ω in scenario ω. It is assumed that demand follows a discrete distribution, for similar reasons as Spliet & Desaulniers (2015).

The problem can be split into two stages. In the first stage the actual demand realization is unknown and is approximated via scenarios Ω, whereas in the second stage demand is known. In practice, the realized demand is only known a week in advance, but the driver schedules need to be known further in advance. Consequently, in the first stage, a global driver schedule is constructed, which is communicated to drivers and serves as starting schedule for the second stage. In the second stage, routes are computed based on actual demand and a local driver schedule is constructed. In the ideal case, the local driver schedule can be fitted into the global driver schedule without any violations.

The global driver schedule can be seen as the schedule provided to drivers a substantial amount of time in advance, with information on working days and times. Note that the global driver schedule is the same for all demand scenarios, whereas the routes and local driver schedule are specific to the scenario ω. It could be, for some demand scenario ω, that the global driver schedule is violated by the local driver schedule. The global driver schedule is only violated if a driver needs to start later or earlier, or on a different day than planned in the global driver schedule. Therefore, additional costs are introduced: $sc_{tardiness}$ per time unit for being scheduled earlier or later than in the global driver schedule and sc_{day} per differently scheduled day respectively. For a certain scenario ω summing these costs over the drivers results in the schedule change costs, defined as C_s^ω.

Now the objective of the SWVRTDSP can be formally stated: find a global driver schedule which minimizes the expected costs over the demand scenarios. The costs are threefold per scenario: travel cost (C_t), preference violation cost (C_p) and schedule change cost (C_s). The travel costs for scenario ω can be calculated by multiplying the used arcs in the routes with the corresponding costs c_{ij}. To calculate the expected costs, the costs (C^w) per scenario ω are multiplied with probability p_ω and summed over all scenarios Ω. Consequently, the total objective value f is:

$$f = C_p\|\Omega\| + \sum_{\omega\in\Omega}(p_\omega C_t^\omega + p_\omega C_s^\omega) \tag{1}$$

4 Adaptive Large Neighbourhood Search

In this section, an ALNS algorithm is presented to solve the SWVRTDSP. Based on Section 2, (meta)heuristics appear to be more suitable than exact methods due to the size of the instances, which include 100 up to 671 customers per day, and the complexity of the problem. More specifically, Wen et al. (2011) found that ALNS, or a variant thereof, is applicable to weekly planning and Lei et al. (2011) show it is suitable to use in case of stochastic demand. Therefore, ALNS is deemed fit to purpose to solve the SWVRTDSP in this report.

Section 4.1 describes the generation of demand scenarios and how HoS regulations are implemented. To create an initial solution, first, a construction heuristic for routes and then a bin-packing-based heuristic for scheduling are used, see Section 4.2. An overview of all neighbourhoods including time complexity is provided in Section 4.3. A sequential approach to the SWVRTDSP using the already available neighbourhoods is described in Section 4.4. The specifics of the probabilistic neighbourhood selection mechanism are elaborated on in Section 4.5. After the stopping criterion, see Section 4.6, has been met, the best solution is returned.

4.1 Scenarios and hours of service regulations

The complexity of the problem mainly comes from the demand uncertainty, as described in Section 3. For large instances, the number of possible scenarios is simply too large (Verweij et al., 2003). Consequently, a Sample Average Approximation (SAA) approach is used. With SAA the objective value is approximated by an average of objective values from a random sample. This random sample is a subset of demand scenarios, which is considered as Ω for the ALNS and is generated via the underlying distribution, with the mean equal to the expected demand of each customer. Do note that the actual demand scenario is not necessarily included in Ω for large datasets. The total objective value is calculated according to Equation (1).

Recall that any feasible schedule needs to satisfy HoS regulations HR.1 to HR.4. This is enforced in the following way. When scheduling, it is assumed that a route with a duration between 6 and 9 hours contains 30 minutes of rest according to HR.1. Do note that the route rest time is not explicitly checked for. If a route of 7 hours and a route of 3 hours are assigned to a driver, then it is assumed HR.1 is already satisfied by the route of 7 hours, so there need to be at least 15 minutes between the routes according to HR.3 to satisfy HR.2. If a route with a duration of more than 9 hours is assigned to a driver then HR.2 is always satisfied, regardless of other routes assigned to the driver. The nightly rest times are explicitly checked for and need to satisfy HR.4.

Not checking routes for rest time explicitly was done for two reasons. First and foremost, the main challenge lies in enforcing the nightly rest time according to experts in the field, as route rest times are usually respected effortlessly. Second, route rest checking resulted in doubling the run time, which due to the time frame of this project would have meant half of the experiments. Therefore, it was the author's choice was to not explicitly check for route rest time.

4.2 Initial solution

A Push Forward Insertion Heuristic (PFIH) is used to provide initial feasible routes for the VRPTW. Wang et al. (2014) state that this heuristic, first introduced by Solomon (1987), efficiently provides good initial routes. The customers are split per day and per retailer to speed up this process.

For the TDSP it is reasonable to assume that bin-packing-based heuristics provide good solutions, since Marzouk & Kamoun (2020) showed bin-packing is effective for worker preferences and shifts. For the TDSP the routes are known, so the problem is how to assign the routes to the drivers. For each route, it is precomputed whether it violates driver preferences or the driver's skills. This makes assigning routes to drivers much easier since the only thing left to check is if routes overlap or HoS regulations are violated.

The following procedure is used: First, the lists of drivers and routes are randomly shuffled. This is important as this is the order routes and drivers get iterated over. Second, it is attempted to construct a schedule that conforms to all driver preferences. A greedy approach is used, where the route is assigned to the first driver that can handle the route during the iteration process. If this is not possible, the driver preferences are ignored and the only objective is assigning all routes greedily to drivers. If not all routes are assigned to drivers, then return to the first step. Due to the randomness of this procedure, a local search is added to improve the solution quality. The following neighbourhoods are visited in order: (a) moving routes from one driver to another, (b) swapping routes between drivers and (c) swapping days between drivers. More information on these neighbourhoods can be found in Section 4.3.11, Section 4.3.12 and Section 4.3.13 respectively.

Experiments with the ALNS algorithm showed that the bin-packing-based heuristic without local search took relatively long to run and did not provide good quality solutions. Especially with the addition of HoS regulations. Therefore, the greedy scheduling approach, described in Section 4.3.5, was developed, which found better quality solutions in significantly less run time. The bin-packing-based heuristic with local search is still used to construct initial schedules, but the greedy scheduling approach is used to find a schedule after each removal and insertion operation.

4.3 Neighbourhoods

This report makes use of thirteen different neighbourhoods. There are four neighbourhoods focused on vertex removal and four neighbourhoods focused on vertex insertion in routes. If selected, q vertices are removed and inserted each day; q is selected randomly in the interval $[\lceil 0.1n \rceil, \lceil 0.2n \rceil]$, as also used by Jia et al. (2023) and Lei et al. (2011), where n indicates the total number of nodes. After insertion, the old local driver schedule is likely not feasible anymore. Therefore, each insertion neighbourhood has a specific schedule construction mechanism. Additionally, there are two problem-specific neighbourhoods and three neighbourhoods purely focused on the local driver schedule. For each neighbourhood, the general idea and approach are described, as well as the time complexity.

4.3.1 Random Removal (RR)

To diversify the search, this neighbourhood selects q vertices at random to remove. This is of great importance as poor local optima differ significantly from the global optimum (Lourenço et al., 2003). Selecting a random vertex to remove takes $\mathcal{O}(1)$ time, which is repeated $\mathcal{O}(q)$ times. Therefore, this neighbourhood is of complexity $\mathcal{O}(q)$.

4.3.2 Similarity Removal (SR)

First proposed by Shaw (1998) and also called Shaw Removal, this neighbourhood selects a first vertex to remove randomly and removes vertices that are similar in terms of proximity and time window until q vertices are removed. The same similarity measure as Lei et al. (2011) is used:

$$S(i,j) = \frac{1}{\frac{c_{ij}}{c_i^{max}} + \frac{1}{\tau_{ij}+\tau_{ji}}} \tag{2}$$

where c_{ij} is the travel cost from customer i to j, c_i^{max} is the maximum cost between i and any other vertex. The time window similarity, τ, is defined below in Equation (3). Do note that u in the left-hand side in the maximum should be chosen carefully. It prevents τ_{ij} from becoming negative, if the right-hand side is negative, so should be chosen such that $u > 0$. In this application time is expressed in minutes, so $u = 1$ is chosen. However, if t_{ij} was defined in hours, u should be chosen much smaller. To prevent a high similarity measure when there is no similarity.

$$\tau_{ij} = \max\{u, \min\{l_j, l_i + t_{ij}\} - \max\{e_j, e_i + t_{ij}\}\} \tag{3}$$

As the location and time windows of vertices do not differ for different scenarios, the similarity measure is calculated for all vertices in advance. This process takes $\mathcal{O}(n^2)$ time since for all $\mathcal{O}(n)$ vertices the similarity measure to all other $\mathcal{O}(n)$ vertices is calculated. Per vertex, the other vertices are then sorted in decreasing order of similarity measure, which takes $\mathcal{O}(n \log n)$ time. So per vertex, the computation time is $\mathcal{O}(n + n \log n)$ reducing to $\mathcal{O}(n \log n)$. Resulting in a total complexity of $\mathcal{O}(n^2 \log n)$. In each iteration selecting the first vertex to remove takes $\mathcal{O}(1)$ time, and selecting the remaining vertices takes $\mathcal{O}(q)$ time as there are presorted lists available. Therefore, the time complexity per iteration is only $\mathcal{O}(q)$ due to the preprocessing. Do note that it is also possible to calculate the similarity measure during each iteration. Then the time complexity would be $\mathcal{O}(n \log nq)$ per iteration. This means that time savings are achieved after roughly $\frac{n}{q}$ iterations by preprocessing. However, significantly more memory is required.

4.3.3 Worst Removal (WR)

This neighbourhood aims to remove the vertices with the largest removal costs. The removal cost is defined as the change in route costs when a vertex is removed from the route. The q vertices with the highest removal cost are removed. A naive approach is used where the removal costs are calculated once, before any vertices are removed, and not updated once vertices get removed. The removal costs are calculated for all $\mathcal{O}(n)$ vertices. Then the costs are ordered in decreasing order of removal costs. Resulting in a time complexity of $\mathcal{O}(n \log n)$. Then the first q vertices get selected to be removed, which takes $\mathcal{O}(q)$ time. Consequently, this neighbourhood is of complexity $\mathcal{O}(n \log nq)$.

4.3.4 Adjusted Worst Removal (AWR)

Contrary to WR, instead of only looking at removal cost in the route, it is also checked how much each vertex impacts the schedule change and preference violation costs. Then the q vertices with the highest combined costs get removed. The idea is that this results in twofold savings, both in terms of routing costs and scheduling costs. A naive approach is used where the adjusted removal costs are calculated once, before any vertices are removed, and not updated once vertices get removed. Similar to Section 4.3.3, this neighbourhood is of complexity $\mathcal{O}(n \log nq)$.

4.3.5 Greedy Insertion (GI)

The approach of this neighbourhood is similar to the PFIH defined in Section 4.2. Starting at the first removed vertex, it is checked where the vertex can be inserted with the least additional cost, where it is then inserted. Which takes $\mathcal{O}(n)$ time. This procedure is repeated until all $\mathcal{O}(q)$ removed vertices are inserted again. If a vertex cannot be inserted in an existing route, a new route is created. As it is possible to keep track of the best insertion place whilst calculating the insertion costs, there is no need to sort the costs at the end, which saves $\mathcal{O}(\log n)$ time. This logic is applied to all processes which require a single position to be returned. Therefore, this process is of time complexity $\mathcal{O}(nq)$. Assume that any insertion neighbourhood results in $\mathcal{O}(r)$ routes.

Additionally, a new local driver schedule needs to be constructed. A greedy approach is used: The list of routes is shuffled, which takes $\mathcal{O}(r)$ time, and then the first route in the list is selected to be inserted first. For each driver, it is calculated what the insertion costs are and the driver with the lowest insertion costs is selected, which takes $\mathcal{O}(p)$ time. This is repeated for all remaining routes. This process is of time complexity $\mathcal{O}(rp)$. If a route cannot be assigned to any driver, the routes are shuffled again and the process is repeated. Technically, the size of the neighbourhood is $\mathcal{O}(rp + r)$ due to the shuffling, but this simplifies to $\mathcal{O}(rp)$.

4.3.6 Demand and Failure Sorting Insertion (DFSI)

The general idea of this neighbourhood is that the most constraining vertices, e.g. with the most demand, should be assigned to the route with the most leftover space. Therefore, DFSI sorts the vertices to be inserted in decreasing order of expected demand. It also sorts the routes in decreasing order of leftover space. The sorting is of time complexity $\mathcal{O}(r \log r + n \log n)$ and since there are at most as many routes as vertices, this results in $\mathcal{O}(n \log n)$ time. A naive approach is used, where the sorting is only done initially and not updated as vertices get assigned. Then, the first vertex of the sorted list is inserted in the first feasible route of the sorted list of routes. More specifically, at the location in the route, where the route costs increase the least. If a vertex cannot be inserted in an existing route, a new route is created. This takes $\mathcal{O}(n)$ time. This procedure is repeated for each vertex to be inserted. Resulting in a time complexity of $\mathcal{O}(nq)$ for the insertion. Technically, the neighbourhood is of size $\mathcal{O}(nq + n\log n)$, but it is assumed that $q > \log n$. Therefore, the vertex insertion is of time complexity $\mathcal{O}(nq)$.

The assignment of routes to drivers is also load based. The routes are sorted in decreasing order of loads, which takes $\mathcal{O}(r\log r)$ time. Each route in the sorted list is then assigned greedily to a driver, which takes $\mathcal{O}(rp)$ time. It could be that at some point a route cannot be assigned to a driver anymore. Then the greedy procedure defined in Section 4.3.5 is used. It is assumed that $p > \log r$ Therefore, the process is of time complexity $\mathcal{O}(rp)$, since $\mathcal{O}(rp + r\log r)$ simplifies to $\mathcal{O}(rp)$.

4.3.7 Regret Insertion (RI)

As opposed to Section 4.3.5, for each vertex the difference in cost of inserting it at the best and second best position is calculated, defined as regret value. This takes $\mathcal{O}(n^2)$ time. The vertex with the highest regret value is inserted first. Then the procedure is repeated until all vertices are inserted again, so $\mathcal{O}(q)$ times. If a vertex cannot be inserted in an existing route, a new route is created. This process is of time complexity $\mathcal{O}(n^2q)$.

The assignment of routes to drivers is also performed with a regret mechanism. For all routes, the costs of assigning it to the best and second best driver are calculated, which takes $\mathcal{O}(rp)$ time. The route with the highest costs is inserted first. Then the process is repeated until all routes are assigned, which is $\mathcal{O}(r)$ times. Resulting in a complexity of $\mathcal{O}(r^2p)$. It could be that at some point a route cannot be assigned anymore. Then the greedy procedure defined in Section 4.3.5 is used. As $\mathcal{O}(r^2p) > \mathcal{O}(rp)$, the complexity is $\mathcal{O}(r^2p)$ regardless of the usage of the greedy procedure.

4.3.8 Memory Regret Insertion (MRI)

Before the removal of vertices, it was known which vertices were in which routes and which routes were assigned to which driver. Consider this as the memory. This neighbourhood uses the memory by adding regret if the memory gets violated. It implicitly assumes that the memory consisted of a good solution, and should only be violated if there is a much better option. Therefore, a similar procedure to Section 4.3.7 is used, but regret is also added if a vertex is assigned to another route than before the removal. This process is $\mathcal{O}(n^2q)$.

For the assignment of routes to drivers, a similar regret approach is used. As opposed to Section 4.3.7, regret is also added if a route is not assigned to the same driver as it was previously assigned to. It could be that at some point a route cannot be assigned anymore. Then the greedy procedure defined in Section 4.3.5 is used. This process is $\mathcal{O}(r^2p)$.

4.3.9 Similar Route Splitting (SRS)

The general idea of this neighbourhood is to split a conflicting route, e.g. a route that starts or ends too late for a specific driver, at the conflicting vertex and try to assign the conflicting part to another driver. This leads to an increase in routing costs, but could significantly reduce schedule change costs. The similar element is to not consider a single driver, but consider a subset of drivers with similar skills, but different start and end time preferences. Assignment of the split routes should be relatively straightforward, as all drivers in the subset have similar skills. Another advantage is that splitting routes is always feasible in terms of time windows and truckload, so requires no feasibility checks.

The subset of drivers is selected as follows. A random number, z, of days between $[\lceil 0.5D \rceil, \lceil 0.75D \rceil]$ is chosen. The days are shuffled and the first z days are selected. A random skill k is selected, e.g. being able to drive truck type 2. Then drivers are selected that want to work on all selected days and can drive the selected truck. This process is of time complexity $\mathcal{O}(p)$. If the subset of selected drivers contains less than $\lceil 0.05P \rceil$ drivers, the selection process is repeated. It is assumed the subset of selected drivers is of $\mathcal{O}(s)$, where $s \leq p$, but likely much smaller. Given the driver subset, create a subset of routes that make use of truck type k and start too early or end too late compared to the global driver schedule. This takes at most $\mathcal{O}(r)$ time. Resulting in a subset of routes of size $\mathcal{O}(m)$, where $m \leq r$, but likely much smaller. The selection process is of complexity $\mathcal{O}(p + r)$, but simplifies to $\mathcal{O}(r)$ as it is assumed that $r > p$.

Given the subset of violating routes, split these routes at the point of violation, and try to assign the violating part of the route to another driver in the subset with the lowest additional cost. Which takes at most $\mathcal{O}(sm)$ time. If it is not possible to assign the route to a driver in the subset, it is randomly assigned to another driver for which the route is feasible, regardless of the cost. This takes $\mathcal{O}(p)$ time. If this is not possible, the original driver schedule and routes without splitting are returned and the neighbourhood failed. The complexity in the worst case is $\mathcal{O}(pm)$, but likely much smaller. The total complexity is $\mathcal{O}(r + pm)$, but for simplicity, it is assumed that $pm > r$. The complexity in the worst case therefore is $\mathcal{O}(pm)$, in the best case it is $\mathcal{O}(sm)$ if $sm > r$.

4.3.10 Global Driver Rescheduling (GDR)

The global driver schedule is arguably the most important part of the SWVRTDSP. Therefore, a novel neighbourhood that focuses purely on the global driver schedule is introduced. The local driver schedules are considered to be fixed. Consequently, only the global driver schedule can be adjusted. The aim is to construct a global driver schedule that minimizes both C_p and C_s. Moreover, this problem can be solved for each driver separately as the drivers are independent.

One approach is to try and use a commercial solver to solve this problem to optimality. However, commercial solvers are expensive to use in practice and have highly varying run times. Therefore, a novel approach to create a global driver schedule is proposed, based on the piecewise linear cost structure of the problem. The difficulty lies in enforcing HoS regulations, as will become apparent.

For now, suppose HoS regulations do not exist. Then the only decision variable is the start and end time of a driver on a certain day d. Since drivers are independent, the method is illustrated for one driver. The procedure of determining the start time for a driver is similar to the procedure of determining the end time. Consequently, only the end time is elaborated on. Denote the end time of a driver on day d as et_d and the end time of a local driver schedule as et_d^ω. The preferred end time of a driver is b, as formulated in Section 3. Define S_d^{end} as the set containing b and all et_d^ω $\forall \omega \in \Omega$. Define $n_{act}^{et_d}$ as the number of active scenarios and $N_{act}^{et_d}$ as the set of active scenarios at end time et_d. $n_{act}^{et_d}$ is the number of scenarios for which $et_d^w > et_d$. The costs of end time violations can then be determined as $C_{et_d} = c_{over} \|\Omega\| * (et_d - b) + \sum_{\omega \in N_{act}^{et_d}} sc_{over} * (et_d^\omega - et_d)$. Note that $et_d \geq b$, so no negative terms can occur. The costs are twofold, the first part relates to the driver preference violation and the second part relates to the local driver schedule violations. This leads to the following observation and theorem:

Observation 4.1. *C_{et_d} is a piecewise linear cost function in et_d and consequently $N_{act}^{et_d}$.*

Theorem 4.2. *Without HoS regulations, the optimal end time on day d is equal to the end time of a scenario or the preferred end time, i.e., $et_d \in S_d^{end}$.*

Proof. Suppose that the optimal $et_d \notin S_d^{end}$, then et_d is between two points in S_d^{end}, denoted as x and y. The costs at x and y can be calculated. Due to Observation 4.1, it is known that the costs are linear between x and y; denote m as the cost slope from x to y. Therefore, if $m < 0$, the costs decrease if $et_d = y$, similarly if $m > 0$ the costs decrease if $et_d = x$. Lastly, if $m = 0$ the cost remain equal if $et_d = x$ or $et_d = y$. □

Now it is possible to determine the optimal start time and end time for each day. Resulting in costs f_d for a specific day. It is known if the driver prefers to work on day d, $o_d = 1$ if so, and how many scenarios, n_{act}, are active that day. Costs sc_{day} are incurred if a driver in some scenario ω needs to work on day d, but not in the global driver schedule. Therefore, the costs of not being scheduled are $sc_{day} * n_{act}$. If a driver is scheduled on day d, then the costs for time violations are incurred, f_d. Additionally, costs $c_{day} * \|\Omega\|$ are incurred if the day was not preferred by the driver, which leads to the following observation:

Observation 4.3. *There exists an optimal solution such that a driver works on day d if the costs of not being scheduled are higher than the costs of being scheduled. If $sc_{day} * n_{act} > \|\Omega\| * c_{day} * (1 - o_d) + f_d$ the driver works on day d.*

Applying Observation 4.3 to the costs incurred by the optimal start and end times leads to an initial global driver schedule. This schedule does not, however, conform to a nightly rest period as per HoS regulation HR.4. It is known that if a driver had 11 hours of rest on day d, then on day $d + 1$ the driver is allowed to have 9 or 11 hours of rest. If the driver had 9 hours of rest on day d, then the driver must have 11 hours of rest on day $d + 1$. This leads to a binary choice of 9 or 11 hours of rest time each day. Consequently, the following theorem can be derived:

Theorem 4.4. *The size of the set of nightly rest scenarios, n_{rs}, for a week of length D, is known and is a Fibonacci sequence.*

Proof. Suppose 11 hours of rest is denoted as 1 and 9 hours of rest is denoted as 0. 0's are not allowed consecutively but 1's are. Suppose $D = 1$, then there are $a_1 = 2$ possibilities, if $D = 2$ then $a_2 = 3$ possibilities, since the only not allowable combination is 00. Now observe that a feasible sequence of length n must either begin with 1 or 01. If it begins with a 1, then it must be followed by a feasible sequence of length $n-1$, of which there are a_{n-1}. If it begins with 01, it must be followed by a feasible sequence of length $n-2$, of which there are $a_n - 2$. So the following recursion applies: $a_n = a_{n-1} + a_{n-2}$, so n_{rs} is known. Moreover, this is the recursion of the Fibonacci Sequence. $\square$

For a week of seven days, only 13 feasible scenarios exist. Theoretically, it would be 21, but it is assumed the weeks must be scheduled independently, which is achieved by setting the rest time of the first day of the week to 11. Given one of the nightly rest scenarios, it is known which nights require 11 hours of rest and which require 9. If the initial schedule already satisfies this, no adjustments are needed.

Now assume the driver lacks r_d^{hours} of rest time on day d. This means either the end time on day d needs to be reduced or the start time on day $d+1$ increased or a combination thereof. For each point, later than st_{d+1} in S_{d+1}^{start} or earlier than et_d in S_d^{end}, the costs are calculated as well as the cost slope to the respective start or end time. This is also calculated for the points $et_d - r_d^{hours}$ and $st_{d+1} + r_d^{hours}$. Define the set with line segments from both the start and end times as L_d, where each line segment l_d has a slope m_d and length $s_d > 0$. A key observation is the following:

Observation 4.5. *Any line segment l_d in the set L_d leads to a positive rest increase of at most r_d^{hours}, i.e., $\forall l_d \in L_d : 0 < s_d \leq r_d^{hours}$.*

The goal is to have as few extra costs whilst increasing the rest time until $r_d^{hours} = 0$. Leading to the following theorem:

Theorem 4.6. *Repeatedly selecting the line segment with the lowest slope in the set L_d leads to the minimum cost increase, compared to the costs before enforcing sufficient rest time, whilst reducing r_d^{hours} to 0. If there are line segments with the same slope, the longer line segment is selected. Do note that the set L_d is updated every time a line segment is selected.*

Proof. Regardless of the line segment l_d picked, it is evident that at some point $r_d^{hours} = 0$ as $l_d > 0$ per Observation 4.5. Denote the initial set of line segments as L_d^1 and suppose $r_d^{hours} = b > 0$. By definition, the line segment with the lowest slope, m^*, leads to the lowest relative cost increase, which is either increasing the start time or reducing the end time. Suppose it is the end time, similar logic applies to the start time, and suppose the rest time is increased by s^1 hours and $r_d^{hours} = b - s^1 > 0$.

The line segments for the end time need to be recalculated from the point $et_d - s^1$, for all these line segments it applies that the slope is higher than m^1. Since if there were to be a slope, say $m_s \leq m^1$, then there would have existed a line segment with $m = a * m^1 + (1-a) * m_s$, where a is the fraction of length taken up by l^1. Which per definition would have been lower than m^1 or would have the same slope but a longer length.

The line segments for the start time are the same, except for the line segments that ended after the point $st_{d+1} + r_d^{hours}$, these are excluded. It is known that the slopes for any of the excluded line segments were higher than m_1 or, if the slope was the same, the length was shorter. Therefore, the remaining line segments per definition have slopes higher or equal to m^1. Since if there was a slope m_s lower than m^1, it would have gotten selected instead of m^1. This means that for any slope m_d in L_d^2, it holds that $m_d \geq m^1$.

Therefore, repeatedly selecting the line segment with the lowest slope leads to the minimum cost increase whilst reducing r_d^{hours} to zero. □

Based on Theorem 4.6 the lowest increase is selected iteratively until the required rest time is achieved, leading to a new start and end time. Leading to the following theorem:

Theorem 4.7. *The costs of working on day d after rest time has been enforced, c_d, are never lower than the costs before rest time has been enforced c_{before}, i.e., $c_{before} \leq c_d$.*

Proof. Suppose initial costs are c_{before} and rest time needs to be increased by $r_d^{hours} > 0$. It holds that $\forall m_d \in M_d : m_d \geq 0$ as the optimal times were determined based on Theorem 4.2. Therefore, c_d never decreases. □

After applying Theorem 4.6, Observation 4.3 is applied again, since it could be that it is cheaper to let a driver not work on day d due to Theorem 4.7. This leads to a new global driver schedule. If the work days have changed compared to the initial global driver schedule, then apply Theorem 4.6 again to the new global driver schedule. At some point, the working days do not change anymore, which means the process of finding a global driver schedule is finished. Recall that for every nightly rest scenario, an initial global driver schedule is constructed and adjusted until Theorem 4.6 is satisfied and the working days do not change.

Then the total working time is calculated, potentially resulting in overtime. A key difference between overtime and nightly rest time is that nightly rest times are a hard constraint and working hours are a soft constraint. Consequently, extra costs are added if a certain end time results in overtime. This leads to new costs for every time in S_d^{end} and slopes to the current end time. If there exist negative slopes from the current end time to a new time, then overall costs can be reduced. If not, then the current time is optimal. If this process is performed simultaneously for all start and end times for the week, this leads to the optimal solution, given the working days and nightly rest scenario.

Now, from all nightly rest scenarios, select the scenario with the lowest costs. This leads to a good global driver schedule, which might be optimal. Optimality is not guaranteed, as not all combinations of working days were iterated over. A design choice is to iterate over all working day combinations as this guarantees optimality. However, in the worst case, this leads to $\mathcal{O}(2^D)$

additional combinations to be iterated over. The time complexity without guaranteeing optimality is determined as follows. There are $\mathcal{O}(\|\Omega\|)$ possible start/end times for $\mathcal{O}(D)$ days and $\mathcal{O}(n_{rs})$ nightly rest scenarios. The entire process above is repeated for every driver, thus the complexity of this neighbourhood is $\mathcal{O}(pD\|\Omega\|n_{rs})$

4.3.11 Route Moving (RM)

As opposed to previous neighbourhoods, RM strictly considers the local driver schedule and moves routes from one driver to another. This helps reduce the workload of a busy driver by moving it to a driver who has time. The number of routes, m, to be moved is selected randomly in the interval $[\lceil 0.1D \rceil, \lceil 0.7D \rceil]$. If a driver does not serve m routes, no routes are selected. A naive approach is used, where for each driver the m routes are only selected once randomly. This takes $\mathcal{O}(m)$ time per driver. Now it is checked for the other $\mathcal{O}(p)$ drivers if the routes can be moved there and what the costs are. Once a driver is found for which costs are reduced, the routes are moved and the moving for this driver is finished. This process is repeated for every driver. Therefore, this neighbourhood is of time complexity $\mathcal{O}(p^2 m)$.

4.3.12 Route Swapping (RS)

RS swaps routes between drivers. The number of routes, m, to be swapped is selected randomly in the interval $[\lceil 0.1D \rceil, \lceil 0.7D \rceil]$. If a driver does not serve m routes, no routes are selected. A naive approach is used, where for each driver the m routes are only selected once randomly. This takes $\mathcal{O}(m)$ time per driver. Now all other drivers, $\mathcal{O}(p)$, are iterated over and for the other driver m routes are selected randomly. Once a driver is found for which costs are reduced, the routes are swapped and the swapping for this driver is finished. This process is repeated for every driver. Therefore, this neighbourhood is of time complexity $\mathcal{O}(p^2 m^2)$.

4.3.13 Day Swapping (DS)

DS works similarly to Section 4.3.12 but swaps entire days between drivers. All possible swaps are enumerated over and the number of swapped days, d, is selected randomly in the interval $[\lceil 0.1D \rceil, \lceil 0.7D \rceil]$. This results in a set of swapping days. A naive approach is used where the swapping days are selected once and is the same for all drivers. For each driver, all routes which occur on the selected day are selected. For simplicity assume there are $\mathcal{O}(k)$ routes per driver per day. Therefore each driver has $\mathcal{O}(kd)$ routes to swap. Now all other drivers, $\mathcal{O}(p)$, are iterated over and for the other driver also $\mathcal{O}(kd)$ routes are selected. Once a driver is found for which costs are reduced, the routes are swapped and the swapping for this driver is finished. This process is repeated for every driver. Therefore, this neighbourhood is of time complexity $\mathcal{O}(p^2 k^2 d^2)$.

4.4 Sequential Approach

In practice, often a sequential approach is used. Therefore, a sequential heuristic approach, based on the neighbourhoods from the previous section, is presented to solve the SWVRTDSP. The approach consists of the following steps:

1. Initialize ALNS with only route removal and insertion neighbourhoods and the forecast scenario(s). Note that AWR and MRI are excluded, as there are no driver schedules thus reducing AWR and MRI to WR and RI respectively

2. Run ALNS, without performing scheduling, until the termination criterion has been met and save the routes

3. Given the computed routes, initialize ALNS again with only neighbourhoods related to scheduling, so GDR, RM, RS and DS

4. Run ALNS until the termination criterion has been met and save the global driver schedule

5. Initialize ALNS with the global driver schedule and the actual scenario(s). Only include the route removal and insertion neighbourhoods, except for AWR and MRI

6. Run ALNS, without performing scheduling, until the termination criterion has been met and save the routes

7. Given the computed routes and the global driver schedule, initialize ALNS again with only neighbourhoods related to local driver scheduling, so RM, RS and DS

8. Run ALNS until the termination criterion has been met and save the global driver schedule, local driver schedule(s) and corresponding routes

4.5 Adaptive search

Windras Mara et al. (2022) state that changing the adaptive mechanism from the roulette-wheel principle could result in a more efficient ALNS algorithm. Therefore, in this report, an objective and performance-based adaptive mechanism is proposed. In each iteration, operations are only performed for one specific neighbourhood, which is selected based on probabilities derived from the weighted costs of all scenarios. Given N neighbourhoods with weights w_n, neighbourhood n is chosen with probability $w_j / \sum_{n \in N} w_n$. The weights are initiated at 1 and updated every q ($q = 75$) iterations.

Each neighbourhood performs well in certain cases; some are more suitable for reducing preference violations and some for improving distance travelled. Recall that the objective consists of three costs: C_t, C_p and C_s. Consequently, each neighbourhood has a weight, $w_n^{C_i}$ per cost type.

The performance weights are also defined per cost type defined as common:

$$p^{C_i}_{n,j+1} = p^{C_i}_{nj}(1-r) + r\frac{\sigma^{C_i}_{nj}}{\epsilon_{ij} * \tau_n} \tag{4}$$

where $\sigma^{C_i}_{nj}$ and ϵ_{nj} are the performance count for costs C_i and number of times the neighbourhood is chosen in the j^{th} sequence of q iterations respectively. The performance count is defined in terms of objective value reduction: if a new feasible global best solution is found it is increased by 16, if an acceptable solution is found that does not improve the objective value it is increased by 6. If the neighbourhood impacts two elements of the objective, e.g. C_s and C_p for GDR, and if it finds a new feasible global best solution for two objectives combined, it is increased by 30. In Section 4.6 the acceptance criteria are stated. τ_n is the time weight of the neighbourhood; if a neighbourhood is more complex a higher penalty is added, to balance results and run time. Based on the time complexities, weights are set to $\tau_{SR} = \tau_{RR} = \tau_{GI} = \tau_{DFSI} = \tau_{GDR} = \tau_{SRS} = 1$, $\tau_{WR} = \tau_{AWR} = 1.1$, $\tau_{RM} = 2$, $\tau_{RS} = 3$, $\tau_{RI} = \tau_{MRI} = \tau_{DS} = 5$. Note that the complexity weights are not exponential, since using exponential weights did not prove efficient with the above performance points for this specific report. The learning parameter r is set to 0.1 in this implementation similar to Lei et al. (2011).

The total weight per neighbourhood can now be calculated. Each cost C_i is expressed in fraction $f^{C_i}_j$ of the total cost, such that the weight of neighbourhood n in the next sequence is defined as

$$w_{n,j+1} = \max(w_{nj}(1-r) + r * \sum_{C_i \in C} p^{C_i}_{n,j+1} * f^{C_i}_{j+1}, 0.1) \tag{5}$$

Note that $w_{n,j+1} \geq 0.1$ at all times. This is a construction choice that allows neighbourhoods that are more suited for final iterations to still be selected at that point, albeit with a low probability.

4.6 Acceptance and stopping criteria

Santini et al. (2018) tested several acceptance criteria for ALNS and found RRT to be the best performing. Consequently, similar to Dueck (1993) and Lei et al. (2011), the Record-to-Record Travel (RRT) algorithm is used to define the acceptance criterion. Assume f^* is the best current objective value, called a record. If the next objective value f^{next} is lower than $f^* + \delta f^*$, δ is set to 0.01, then the solution is accepted. If f^{next} is lower than f^* then f^* is updated. The search stops if the solution quality has not improved by $\phi * f^*$ in the last $max(100, 10\|\Omega\|)$ number of iterations, where ϕ is set to $\frac{0.001}{\|\Omega\|}$ i.e. 0.1% per scenario. This parametrization in terms of the number of scenarios aims to prevent premature converging.

5 Computational Results

In this section, the results of the computational experiments are presented. First, the used instances are described in Section 5.1. Then, the impact of forecast scenarios is discussed in Section 5.2 and the impact of different solving approaches in Section 5.3. The performance of the neighbourhoods is analyzed in Section 5.4. The effects of different cost and driver pool configurations are provided in Section 5.5 and Section 5.6 respectively. Lastly, Section 5.7 compares the current practice to the best approach found in this report.

All tests are performed on a 2.8 GHz Intel Core i7-7700HQ Quad-Core processor. The algorithms were coded in Java and run with IntelliJ IDE. To limit the total run time, only the first instance of every Goel problem class is used. As, on average, an experiment takes roughly six hours per instance. Only for Section 5.7 all instances were used. An integrated solving approach is used unless otherwise specified.

5.1 Test instances

The instances used are: (i) instances from Goel & Irnich (2017) and (ii) a real-life instance for a large logistical company in the Netherlands, kindly provided by Conundra. Euclidian distances are used for all instances. The 56 instances from Goel & Irnich (2017) can be separated in six problem classes: C1, C2, R1, R2, RC1 and RC2. Each instance contains locations of 100 vertices, time windows, demand, service time, number of available vehicles and vehicle capacity. The instances are adjusted to include stochastic demand, different retailers, different trucks and drivers, including skills and preferences. Moreover, the instances are designed for six-day time windows.

Therefore, the instances are rescaled to have at most a time window of a day. The following procedure is used for adaptation: the travel times and time windows are divided by six. Four retailers are introduced, where each customer has a 25% probability of belonging to a specific retailer r. Additionally, customers belonging to the last two retailers can be served by the same truck. Each day 100 trucks are available, and each truck k has a 25% probability of being able to visit a certain retailer. The truck capacity is equal for all trucks and set to the capacity in the instances divided by 3. This is a conscious choice as it results in shorter routes, which fit more to the application of this report. The mean demand for customer i on day d, d_i^d is taken uniformly random in the interval $[0, 2d_i]$, where d_i is the demand specified in the instance.

A driver pool of 100 drivers is available. Each driver has a 75% probability of wanting to work on day d and a 75% probability of being able to drive truck k. A driver has a 40% probability of wanting to start at hour 0 and a 20% probability of wanting to start at hour 6, 12 or 18. Given the start time t_{start}, the preferred end time is generated uniformly random, on the interval $[t_{start} + 4, 24]$. The preferred weekly hours are drawn from a Poisson distribution with the mean equal to 40 hours.

The real-life instance contains 671 customers per day with time windows, service times, locations and expected demand. There are three different retailers and seven distinctive truck types, as some customers can not be served by trucks of other retailers, with the same capacity. There are 710 drivers available for the week. The driver pool is a real driver pool with certain shifts, preferred working days and hours per driver.

The following is set and saved for each instance: 100 different forecast demand scenarios, and 100 actual demand scenarios. Where all 200 demand scenarios are generated independently via $Poi(d_i^d)$ for each customer i on each day d. The 100 actual demand realizations serve to create a reliable estimate of average performance. Costs are set as: $c_{tardiness} = c_{over} = \alpha$, $c_{day} = 500 * c_{tardiness} = 500\alpha$, $sc_{tardiness} = \beta$, $sc_{day} = 500 * sc_{tardiness} = 500\beta$, $c_{ij} = \gamma * t_{ij}$, with minutes as time unit. This cost configuration is defined as (α, β, γ), which is set to (1, 2, 1) initially. This configuration is used as scheduling costs and routing costs are of roughly equal magnitude. Moreover, the costs for C_s are higher than for C_p, as it assumed that the inconvenience of a last-minute schedule change is higher compared to one communicated in advance.

5.2 Impact of number of forecast scenarios

In this section, the impact of an increasing number of forecast scenarios on the solution quality is tested. Table 1 shows the mean realized objective value over the 100 actual scenarios for a varying number of forecast scenarios.

Table 1: Mean realized objective value based on a certain number of forecast scenarios

Instance	μ-1	μ-2	μ-3	μ-5	μ-10	μ-25	μ-50	μ-75	μ-100
C101	94,078	93,602	91,982	94,672	90,803	94,897	91,162	**90,210**	89,735
C201	77,060	76,183	74,210	76,114	**69,165**	74,611	74,655	75,891	73,364
R101	71,352	69,438	72,294	72,590	67,526	97,870	71,508	**67,341**	79,224
R201	58,143	60,031	59,918	60,041	61,278	59,885	59,150	**57,958**	65,033
RC101	92,170	88,864	89,850	84,462	87,278	87,401	78,628	**76,380**	81,958
RC201	64,828	69,106	69,546	**63,470**	64,397	65,685	64,055	78,060	64,560
Average	76,272	76,204	76,300	75,225	73,408	80,058	**73,193**	74,307	75,645

Note. The first column shows the instance. The other columns show the mean objective value (μ) for the actual scenarios for a certain number of forecast scenarios, which is indicated at the top as μ-number. Lastly, at the bottom row, the average objective value over the instances is calculated. Per row, the lowest mean is highlighted in bold.

The mean objective value can be interpreted as the realized costs for the actual scenarios based on the global driver schedule, which was created using the forecast scenarios. So a lower objective value means fewer costs and thus equals a better solution. It can be observed that there is no significant change in the objective values when using 1, 2 or 3 forecast scenarios in Table 1. However, using more scenarios, generally speaking, seems to reduce the objective value. For 10 and 50 scenarios a reduction in objective value of roughly 4% is achieved. At the same time, using 25 scenarios would increase the objective value by roughly 5%. This is mainly caused by instance R101, as for this instance the objective value is disproportionately high compared to the other number of forecast scenarios. A possible reason is that the ALNS terminated prematurely for this instance. Additionally, it can be observed that for most instances 75 forecast scenarios lead to the lowest objective value, but for other instances, i.e. RC201 and C201, fewer forecast scenarios would be preferred.

Performing the same experiment for a cost configuration of (1, 2, 20) shows there is no consistency in the optimal number of forecast scenarios. The full results of this experiment can be found in Table 7 in Appendix A. This variability could be explained by the stochastic nature of the forecast and actual scenarios. It could be that the set of forecast scenarios simply includes many improbable or extreme scenarios. On the other hand, the issue could also lie in assessing the mean objective value. It could be that 100 scenarios are not sufficient to realistically capture the stochasticity and thus result in an unreliable estimate of the realized costs. Regardless, it holds that increasing the number of forecast scenarios generally leads to lower objective values.

For further experiments, the number of forecast experiments is set to 10. This number of forecast scenarios reduces the mean objective value for both cost configurations, with reductions of 3.75% for (1,2,1) and 1.64% for (1, 2, 20) compared to a single forecast scenario.

5.3 Effects of different solving approaches

In this section, the effect of different solving approaches on the realized solution quality is elaborated on. There are three possible approaches: sequential (seq), integrated (int) and sequential followed by integrated (si). These approaches can be used both for creating the global driver schedule, based on forecast scenarios, and realized solutions, based on actual scenarios, leading to nine possible solving approaches. Table 2 shows the average results of this experiment for the different approaches.

Table 2: Average results of different solving approaches in terms of objective value and run time

App-f	App-a	μ-a	σ-a	Rt-f (s)	Rt-a (s)	C_p	C_s-a	C_t-a
int	int	77,429	6,020	74	2,546	18,707	3,240	55,401
int	seq	72,323	7,876	74	386	18,707	15,818	37,712
int	si	**72,195**	7,899	74	547	18,707	15,660	37,741
seq	int	83,446	6,241	12	2,718	18,150	7,138	58,071
seq	seq	75,907	8,084	12	505	18,150	19,854	37,809
seq	si	73,234	6,950	12	644	18,150	17,188	37,809
si	int	77,598	6,359	21	2,785	18,150	3,186	56,178
si	seq	74,123	7,781	21	450	18,150	18,086	37,801
si	si	74,084	7,766	21	550	18,150	18,038	37,810

Note. Columns related to forecast scenarios are indicated by -f, and columns related to actual scenarios by -a. The first and second column show the approach (App) used. Columns three and four show the mean realized objective value (μ) and standard deviation over the scenarios (σ). Columns five and six show the run time (Rt), in seconds. The last three columns show the average objective value divided into preference violation costs (C_p), schedule change costs (C_s) and routing costs (C_t). The lowest mean objective value is highlighted in bold. For future reference, note that C_p, C_s and C_t might not add up to μ due to internal rounding.

Table 2 shows that using an integrated approach to create the global driver schedule results in the lowest objective values compared to the other two approaches. However, sequential followed by integrated does find the best objective values for the actual scenarios. Additionally, it seems the standard deviation generally increases as the objective decreases. The improvement over the sequential approach found by the integrated approach ranges from 0.05% to 3.52%. In a similar experiment, based on 25 forecast scenarios instead of 10, improvements of up to 22% were found. The full results can be found in Table 8 in Appendix A.

Interestingly, for 25 forecast scenarios, using the sequential approach results in the best results. An explanation is that the initial solution from Section 4.2 might not be sufficient to find good results. Sequential followed by integrated can be seen as providing the solution from the sequential approach as the initial solution for the integrated approach. This approach is roughly 5 times faster than the integrated approach with the initial solution from Section 4.2. From Table 2, it can also be observed that a purely integrated approach struggles with the routing costs but excels at scheduling costs. Whereas, for the sequential approach, it is the exact opposite, which seems to indicate that there is a trade-off between scheduling costs and routing costs. This could be explained by the duration of the routes, as this is relevant for the scheduling but not for the routing costs.

Not too much emphasis should be placed on run time, as in practice only the run time required to create the global driver schedule and the run time for the actual scenario are relevant. For any approach, this means that, on average, it would take at most two minutes to solve.

5.4 Performance of neighbourhoods

In this section, the neighbourhood performance is analyzed, with a specific focus on the problem-specific neighbourhoods. Table 3 shows the average performance of all neighbourhoods over the six Goel instances.

Table 3: Overview of average neighbourhood performance using forecast scenarios

Neighbourhood	$\sigma_{tot}^{C_p}$	$\sigma_{tot}^{C_s}$	$\sigma_{tot}^{C_t}$	σ_{tot}	ϵ_{tot}	τ	p_{avg}	Rt (ms)
RR	0	82	2,362	2,444	527	1	4.64	0.01
SR	0	79	2,951	3,030	573	1	5.29	0.01
WR	0	152	3,279	3,431	554	1.1	5.63	0.19
AWR	0	110	3,119	3,230	567	1.1	5.18	0.23
GI	0	270	4,449	4,719	651	1	7.25	1.14
DFSI	0	34	1,894	1,928	517	1	3.73	1.01
RI	0	51	2,675	2,726	428	5	1.27	65.89
MRI	0	68	2,694	2,762	465	5	1.19	78.55
SRS	0	517	623	1,140	379	1	3.01	0.02
GDR	2,723	0	0	2,723	676	1	4.03	0.13
RM	0	816	0	816	313	2	1.31	1.78
RS	0	651	0	651	317	3	0.68	0.87
DS	0	922	0	922	297	5	0.62	10.14

Note. The first column shows the neighbourhood name. The next three columns show the total performance count, σ_{tot}, for C_p, C_s and C_t respectively, which are summed in column five. The next three columns show the total number of times a neighbourhood is chosen (ϵ_{tot}), time weight (τ) and average performance ($p_{avg} = \frac{\sigma_{tot}}{\epsilon_{tot} * \tau}$). Lastly, the average run time (Rt) for each neighbourhood is reported in milliseconds. Recall that the performance count is increased if a certain neighbourhood improves the global solution or finds an acceptable solution.

Table 3 shows that MRI, RS and DS seem to be performing relatively poorly compared to the other neighbourhoods, in terms of average performance. Do note, that this is also influenced by the time weight of the neighbourhood, e.g. MRI would have an average performance of 6 with a time weight of one. The average run times are mostly in line with the time complexities described in Section 4.3. However, there are two outlying pairs: MRI/RI and RM/RS. MRI and RI have the same time complexity ($\mathcal{O}(n^2q + r^2p)$), but MRI takes roughly 20% as long to run. This could be explained by the additional regret calculations needed for MRI. This also shows the danger in purely relying on time complexities, as a similar time complexity can still result in a significant difference in run time. The difference in run time between RM ($\mathcal{O}(p^2m)$) and RS ($\mathcal{O}(p^2m^2)$) is directly contradicting the time complexities. One explanation is that, theoretically, there might be significantly more possible combinations for RS, but due to the driver's skills this is not the case leading to less run time than expected. Another explanation is that RM is programmed inefficiently.

From Table 3, MRI and AWR seem to perform slightly worse compared to RI and WR respectively. It could indicate that they are potentially interchangeable and one could be removed. A similar argument can be made for RS and DS. Therefore, each of these neighbourhoods is excluded once. Even though SRS is performing well, this neighbourhood is also excluded to further test if problem-specific neighbourhoods are necessary. Table 4 shows the results of excluding these neighbourhoods, RM is also added for comparison.

Table 4: Average results of excluding certain neighbourhoods in terms of objective value

Excluded	μ-a	σ-a	Rt-f (s)	Rt-a (s)	C_p	C_s-a	C_t-a	$\Delta\mu$ (%)
None	77,429	6,020	74	2,546	18,707	3,240	55,401	0.00
WR	83,633	4,871	69	2,303	17,990	4,200	61,360	8.01
AWR	77,281	5,635	83	2,623	18,431	3,042	55,727	-0.19
RI	79,964	5,694	57	1,669	17,740	5,414	56,726	3.27
MRI	76,463	5,946	48	1,660	18,754	1,229	56,406	-1.25
SRS	75,681	6,998	80	3,222	17,851	2,285	55,469	-2.26
RM	80,546	6,892	62	2,632	17,524	8,448	54,481	4.03
RS	77,620	6,432	90	2,825	18,120	3,092	56,325	0.25
DS	77,540	6,949	59	3,682	19,279	5,750	52,431	0.14

Note. Columns related to forecast scenarios are indicated by -f or by -a for actual scenarios. The first column shows the excluded neighbourhood. Columns two and three show the mean realized objective value (μ) and standard deviation over the scenarios (σ). Columns four and five show the run time (Rt), in seconds. The next three columns show the average objective value divided into preference violation (C_p), schedule change (C_s) and routing costs (C_t). Lastly, the relative objective change, $\Delta\mu$, in % is shown compared to no exclusion.

Table 4 shows that excluding WR leads to significantly worse results but excluding AWR results in a negligible change in terms of objective value. Therefore, it seems that WR is essential and AWR is not. Excluding RI leads to a significant increase in objective value, whereas excluding MRI leads to a lower objective value. Showing that MRI potentially diversifies the search too much. Excluding SRS leads to a better objective value, but also to an increase in run time of roughly 25%. Showing that SRS potentially intensifies the search too much, due to which the run time also decreases. Lastly, it can be observed that excluding RM leads to significantly worse results, whereas excluding RS or DS leads to a similar objective value. A key difference is that excluding DS leads to an increase in the run time of almost 50%, whereas excluding RS only slightly increases the run time. Therefore, it seems RS and DS serve the same function, but DS does so more quickly.

These experiments have shown that not all neighbourhoods perform well, either in terms of run time compared to their complexity or improving the solution. Consequently, more neighbourhoods do not necessarily help ALNS achieve a better solution. Neighbourhoods that could be excluded are AWR, MRI and RS, as WR, RI and DS are better counterparts respectively. Optionally, SRS could be removed, as this would improve the solution quality, but increase the required run time.

5.5 Impact of cost prioritization

In this section, the effects of different cost configurations are tested. Figure 1 shows the average total objective value and the breakdown into the three different costs for certain cost configurations. First, for the base cost configuration of (1, 2, 1). Then the preference violation, scheduling violation and routing costs are each prioritized in turn, where costs are set to 10. Lastly, the total scheduling costs are prioritized. Note that the ALNS is run for the cost configurations shown in the figure, but for comparison, the costs are scaled back to the base cost configuration in the figure.

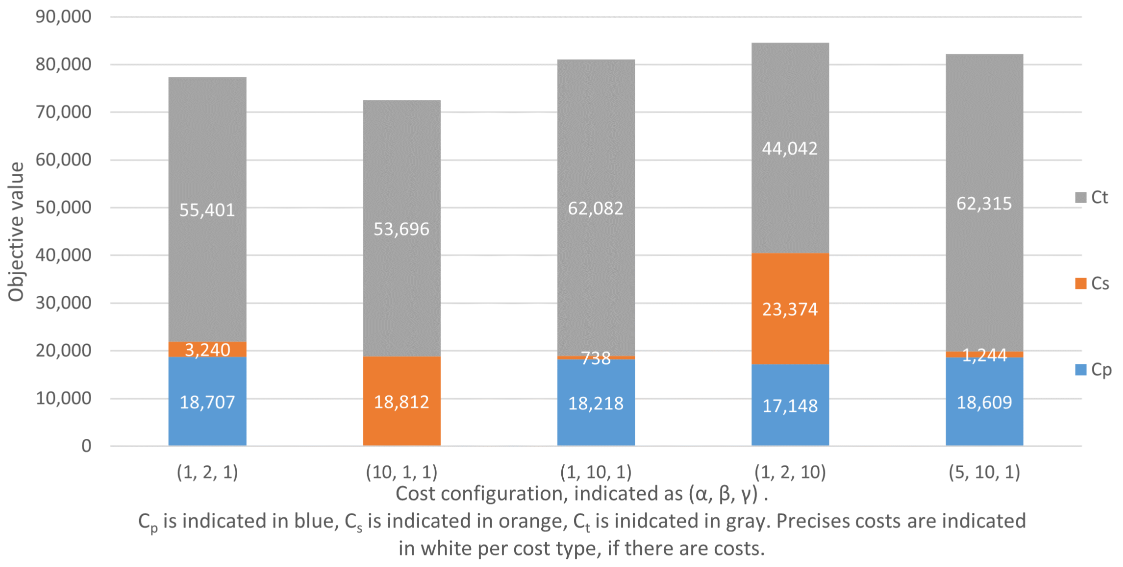

Figure 1: Scaled average objective value breakdown for different cost configurations

First, it can be observed that heavily prioritizing one specific cost generally comes with an increase in the other costs. Prioritizing preference violation costs leads to much higher schedule change costs and vice versa. Prioritizing the routing costs leads to much higher total scheduling costs, specifically the schedule change costs. Second, it seems that prioritizing both the preference and schedule change costs does not significantly reduce scheduling costs compared to the base cost scenario in Figure 1. This shows that it is not possible, for these instances, to create a global driver schedule and local driver schedules which overlap and do not violate driver preferences. Third and last, if the preference violation costs are prioritized, then the total costs are the lowest.

5.6 Impact of driver pool characteristics

In this section, the effects of driver skills and availability on the scheduling costs are explored. Four driver pool configurations are used: balanced (75%, 75%), skilled and available (90%, 90%), skilled (90%, 60%) and available (60%, 90%). The percentages indicate the probability of being able to drive truck k and working on day d respectively. Table 5 shows the average results of this experiment.

Table 5: Influence of driver skills and availability on scheduling violations

Config	μ-a	C_p	Day	Early	Late	Over	P_{act}	C_s-a	Day	Early	Late	C_t-a
75%, 75%	75,099	17,926	0.67	9,171	8,189	233	67	1,131	0.88	75	49	56,042
90%, 90%	70,245	14,117	0	7,344	6,679	94	65	591	0.43	28	51	55,537
90%, 60%	70,730	14,185	0	7,279	6,669	237	64	639	0.49	31	46	55,906
60%, 90%	69,045	12,756	0	6,688	5,823	245	60	458	0.35	22	34	55,831

Note. Columns related to forecast scenarios are indicated by -f or by -a for actual scenarios. The first column shows which driver pool configuration is used. The second column shows the mean objective value (μ). Column three shows the preference violation costs (C_p). Columns four to seven show the breakdown of C_p into non-preferred working days, starting early, ending late and working overtime respectively. Column eight shows how many drivers are used (P_{act}). Column nine shows the schedule change costs (C_s), which can be divided into different working days, earlier start and later ending times respectively in columns ten to twelve. The last column shows the routing costs (C_t).

From Table 5 it becomes apparent that, for the tested instances, driver availability is more important than driver skills. However, it also shows that a balanced driver pool results in worse scheduling costs than either an available or skilled driver pool. A driver pool that is both skilled and available performs worse than an available driver pool. There was no sign of premature converging of the ALNS, so it could simply be due to the randomness in the generation of driver pools. Alternatively, ALNS remains a heuristic, so there is no guarantee that it finds equally good solutions every time.

A closer look at the breakdown of the scheduling costs in Table 5 reveals that almost all drivers work on their preferred working days and that this is not violated significantly for the actual scenarios. Additionally, the overtime of the drivers is also limited to, in the worst case, four hours for one driver. However, the working time preferences of the drivers get violated significantly over the week. With drivers, on average, starting roughly 100 minutes earlier and later than they would have preferred during the week. On the plus side, for the actual scenarios drivers, on average, do not need to work much earlier or later than they were planned originally.

The large amount of working time preference violations in the schedule is likely caused by the generation of the drivers in combination with the instance characteristics. Recall that 40% of the drivers are available from the start of the day. Moreover, the end time is at least four hours later than the start time but generated uniformly random, causing relatively few drivers to be available until the end of the day and potentially only being available for four hours a day. The combination of relatively few drivers wanting to start early and end late seems like a plausible reason for the time preference violations. From Table 5 it can be observed that fewer drivers are required as the availability of drivers increases. This means the drivers who are available for a long time each day, can be used on more days than before. Consequently, leading to fewer drivers required and a reduction in early starting or late ending.

5.7 Comparison with current practice

In practice, a single-scenario sequential approach is typically used, where the routes are assigned manually. Similar to Wagenvoort et al. (2022), the manual assignment is mimicked using the greedy schedule insertion heuristic from Section 4.3.5. The global driver schedule is then equal to this greedily assigned schedule. For the actual scenarios, a similar approach is used. This entire procedure is referred to as 1-practice, 1 indicates the number of forecast scenarios. In this section, the 1-practice approach is compared to the 1-seq approach, see Section 4.4, and the 10-int-si approach. These three approaches are tested on all instances. The average results can be found in Table 6.

Table 6: Comparison of 1-practice, 1-sequential and 10-int-si approaches

Instance	Approach	μ-a	σ-a	Rt-f (s)	Rt-a (s)	C_p	C_s-a	C_t-a	$\Delta\mu$-a (%)
C1$_{avg}$	1-practice	84,116	16,067	2	215	22,100	19,661	42,355	0.00
$\|I\| = 9$	1-seq	64,161	8,743	6	750	16,010	5,796	42,355	-23.72
	10-int-si	68,307	8,583	144	945	15,724	10,226	42,357	-18.79
C2$_{avg}$	1-practice	72,352	2,935	3	107	29,027	16,280	27,046	0.00
$\|I\| = 8$	1-seq	60,846	4,367	3	230	20,079	13,722	27,045	-15.90
	10-int-si	58,907	2,266	21	405	21,373	10,376	27,159	-18.58
R1$_{avg}$	1-practice	61,459	12,240	2	204	12,747	13,848	34,864	0.00
$\|I\| = 12$	1-seq	53,187	6,872	5	768	9,899	8,426	34,863	-13.46
	10-int-si	50,531	5,918	83	903	9,210	6,444	34,878	-17.78
R2$_{avg}$	1-practice	57,269	3,185	4	114	11,856	19,093	26,321	0.00
$\|I\| = 11$	1-seq	51,754	2,009	3	215	14,197	11,237	26,320	-9.63
	10-int-si	48,833	2,200	15	559	14,389	7,938	26,506	-14.73
RC1$_{avg}$	1-practice	77,402	11,954	2	263	10,405	20,390	46,607	0.00
$\|I\| = 8$	1-seq	64,790	5,331	5	955	12,655	5,531	46,605	-16.29
	10-int-si	68,215	5,038	148	1,084	13,316	8,289	46,610	-11.87
RC2$_{avg}$	1-practice	68,126	5,287	3	119	16,277	20,037	31,812	0.00
$\|I\| = 8$	1-seq	60,127	4,028	3	244	20,561	7,757	31,809	-11.74
	10-int-si	58,383	3,410	18	466	20,312	6,154	31,918	-14.30
Goel$_{avg}$	1-practice	70,121	8,612	3	170	17,068	18,218	34,834	0.00
	1-seq	59,144	5,225	4	527	15,567	8,745	34,833	-15.65
	10-int-si	58,863	4,569	72	727	15,721	8,238	34,904	-16.06
Real-life	1-practice	553,653	107,836	279	5,157	162,892	102,400	288,361	0.00
$\|I\| = 1$	1-seq	435,969	42,520	916	11,266	128,831	18,777	288,361	-21.26
	10-int-si	426,630	30,089	2,542	16,916	128,677	7,998	289,955	-22.94

Note. Columns related to forecast scenarios are indicated by -f or by -a for actual scenarios. The first two columns indicate the instance type, with the number of instances $\|I\|$ if applicable, and approach respectively. Columns three and four show the mean realized objective value (μ) and standard deviation over the scenarios (σ). Columns five and six show the run time (Rt), in seconds. The next three columns show the average objective value divided into preference violation costs (C_p), schedule change costs (C_s) and routing costs (C_t). Lastly, the relative objective change, $\Delta\mu$, in % is shown compared to 1-practice.

The 1-seq approach is used to show how much practice could potentially improve, whilst still using a single-scenario sequential approach. The 10-int-si approach was found to be most suitable, for the tested instances in Section 5.2 and Section 5.3, and is used to show the potential benefits of a multi-scenario integrated approach. Note that Goel$_{avg}$ is averaged over the problem classes to prevent the dominance of problem classes with more instances. For any approach, the same routes are used for the actual scenarios, so the objective value differences are mainly caused by the scheduling costs. To limit the run time for the real-life instance, a maximum number of iterations is set to 250, 1000 and 2500 for one forecast, ten forecast and the actual scenarios respectively.

Experiments in verifying the routing costs using Cplex and the VRPTW formulation, see Appendix B, from Kallehauge et al. (2005) resulted in large optimality gaps for Goel instances. Therefore, the commercial solver, Optiflow, from Conundra was used for the real-life instance. The best routing costs found by the sequential approach were roughly 272,000, which are within 1% of the routing costs found by Optiflow when no limit is set on the number of iterations.

From Table 6, it can be observed that 1-practice yields the worst results in terms of mean objective value and standard deviation for any type of instance. Interestingly, the performance of the 1-seq and 10-int-si approaches varies significantly for different instances. This could be caused by the garbage in - garbage out principle (Kilkenny & Robinson, 2018), a poor quality scenario will lead to a poor quality solution, which a single-scenario approach is particularly prone to. It can also be observed that 10-int-si generally increases routing costs, to reduce scheduling costs.

For four out of six Goel problem classes, 10-int-si outperforms 1-seq by 2.56 percentage points (pp) up to 5.10pp. However, 1-seq outperforms 10-int-si by 4.93pp and 4.42pp for problem classes C1 and RC1 respectively. On average, for the Goel instances, the 10-int-si performs 0.40pp better than 1-seq at the cost of a roughly 50% total run time increase. Even so, the standard deviation is roughly 10% lower for the 10-int-si approach compared to 1-seq. Showing that the global driver schedule of the 10-int-si approach is more robust than that of the 1-seq approach.

For the real-life instance, a similar trend is visible. The objective value is reduced by 21.26% for 1-seq, compared to 1-practice. Yet, 10-int-si finds a better solution in terms of objective value. Furthermore, the standard deviation is over 25% lower compared to 1-seq. Confirming that 10-int-si results in a more robust global driver schedule. Moreover, these results were found with a limit on the maximum number of iterations. Letting ALNS converge naturally should result in even better results. Even so, routing costs are within 10% of the best-found routing costs.

Table 6 shows that the scheduling costs can be reduced significantly, compared to 1-practice, regardless of the approach. This shows route swapping and moving neighbourhoods reduce the schedule change costs significantly. Moreover, results show that, for similar preference violation costs, schedule change costs are, on average, significantly reduced for 10-int-si compared to 1-seq. This shows GDR works as intended and effectively creates a robust global driver schedule. Lastly, for the real-life instance, it would take 10-int-si roughly 45 minutes to create a global driver schedule. This seems acceptable, considering the size of the instance.

6 Conclusion

In this report, the VRTDSP with time windows was extended to include stochastic demand and a one-week planning horizon, resulting in the SWVRTDSP. An adaptive large neighbourhood search algorithm was developed to solve for instances with up to 671 customers per day. Additionally, driver schedules conform to (EC) No. 561/2006 and are based on drivers' working preferences. Furthermore, a novel approach, GDR, to effectively and efficiently construct a global driver schedule based on a set of forecast scenario(s) was presented. The global driver schedule was evaluated using a sample average approach with 100 actual scenarios. When 10 forecast scenarios were used, it was found that an integrated approach results in the best global driver schedule, compared to a sequential approach. For the actual scenarios, the best practice is to first solve sequentially and provide this solution as an initial solution for the integrated ALNS.

Furthermore, multiple experiments were performed regarding neighbourhood performance, cost priority and drivers. It was shown that most problem-specific neighbourhoods did not perform better than commonly used neighbourhoods, except for GDR. Additionally, it was shown that, regardless of the cost priority, scheduling costs remain. However, it is possible to reduce either the preference violation costs or schedule change costs to almost zero. Interestingly, there is a trade-off between routing costs and scheduling costs, reducing one increases the other. Regarding the drivers, availability seems to be more important than driver skills. A driver pool with highly available drivers results in 10% fewer drivers being required, whilst also reducing the scheduling costs significantly.

Finally, experiments for the SWVRTDSP were performed that compared a multi-scenario integrated approach to a single-scenario approach commonly used in practice and a more elaborate single-scenario approach. Results show that the objective value can be reduced by roughly 10% up to 24% for the elaborated single-scenario approach, compared to practice. However, the multi-scenario integrated approach, on average, resulted in even lower objective values and a significantly lower standard deviation, for both the instances from literature and the real-life instance. Showing that GDR is effective at creating robust global driver schedules. To conclude, a multi-scenario integrated approach results in the most robust driver schedules and the lowest objective values on average, with an objective reduction of 22.94% for the real-life instance, compared to practice.

In the future, multiple research directions following the findings of this report can be explored. First, the required break time in routes should be explored, as this was not explicitly checked for in this report. Second, the stochasticity could be fully captured in a probabilistic model, similar to Lei et al. (2011). Third, exact approaches could be explored, either to construct routes, create schedules or both. Fourth, more research into the inner workings of ALNS could be performed. Fifth and last, a heuristic method that includes more scenarios or is capable of solving larger instances more efficiently could be explored.

References

Boujlil, M., & Lissane Elhaq, S. (2020, 12). The vehicle routing problem with Time Window and Stochastic Demands(VRPTW-SD): Review. *2020 13th International Colloquium of Logistics and Supply Chain Management, LOGISTIQUA 2020*. doi: 10.1109/LOGISTIQUA49782.2020.9353927

Bredstrom, D., & Ronnqvist, M. (2007, 2). A Branch and Price Algorithm for the Combined Vehicle Routing and Scheduling Problem With Synchronization Constraints. *SSRN Electronic Journal*. Retrieved from https://papers.ssrn.com/abstract=971726 doi: 10.2139/SSRN.971726

Castillo-Salazar, J. A., Landa-Silva, D., Qu, R., Castillo-Salazar, J. A., Landa-Silva, D., Qu, R., ... Qu, R. (2016). Workforce scheduling and routing problems: literature survey and computational study. *Ann Oper Res*, *239*, 39–67. doi: 10.1007/s10479-014-1687-2

Chang, M. S. (2011). A vehicle routing problem with time windows and stochastic demands. *http://dx.doi.org/10.1080/02533839.2005.9671048*, *28*(5), 783–794. Retrieved from https://www.tandfonline.com/doi/abs/10.1080/02533839.2005.9671048 doi: 10.1080/02533839.2005.9671048

Dueck, G. (1993). New optimization heuristics; The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, *104*(1), 86–92. doi: 10.1006/JCPH.1993.1010

European Parliament. (2006). *Regulation (EC) No 561/2006.* Retrieved from https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex:32006R0561

Feillet, D. (2010, 6). A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR*, *8*(4), 407–424. Retrieved from https://link.springer.com/article/10.1007/s10288-010-0130-z doi: 10.1007/S10288-010-0130-Z/METRICS

Goel, A. (2008, 4). Vehicle Scheduling and Routing with Drivers' Working Hours. *https://doi.org/10.1287/trsc.1070.0226*, *43*(1), 17–26. Retrieved from https://pubsonline.informs.org/doi/abs/10.1287/trsc.1070.0226 doi: 10.1287/TRSC.1070.0226

Goel, A. (2010). Truck driver scheduling in the European union. *Transportation Science*, *44*(4), 429–441. Retrieved from http://pubsonline.informs.org441.https://doi.org/10.1287/trsc.1100.0330http://www.informs.org doi: 10.1287/TRSC.1100.0330

Goel, A., & Irnich, S. (2017). An exact method for vehicle routing and truck driver scheduling problems. *Transportation Science*, *51*(2), 737–754. Retrieved from http://pubsonline.informs.org754.https://doi.org/10.1287/trsc.2016.0678http://www.informs.org doi: 10.1287/TRSC.2016.0678

IRU. (2022). Driver Shortage Global Report 2022: Summary.

Jia, S., Deng, L., Zhao, Q., Chen, Y., Jia, S., Deng, L., ... Chen, Y. (2023). An adaptive large neighborhood search heuristic for multi-commodity two-echelon vehicle routing problem with satellite synchronization. *Journal of Industrial and Management Optimization*, *19*(2), 1187–1210. Retrieved from /en/article/doi/10.3934/jimo.2021225/en/article/doi/10.3934/jimo.2021225?viewType=HTML doi: 10.3934/JIMO.2021225

Kallehauge, B., Larsen, J., Madsen, O. B., & Solomon, M. M. (2005). Vehicle routing problem with time windows. *Column Generation*, 67–98. Retrieved from https://link-springer-com.tudelft.idm.oclc.org/chapter/10.1007/0-387-25486-2_3 doi: 10.1007/0-387-25486-2{_}3/COVER

Kilkenny, M. F., & Robinson, K. M. (2018, 5). Data quality: "Garbage in – garbage out". *https://doi.org/10.1177/1833358318774357*, *47*(3), 103–105. Retrieved from https://journals.sagepub.com/doi/full/10.1177/1833358318774357?casa_token=1thm2Jf1PEAAAAAA%3A3n8jumH35AzNxfk6GFdkrXbkTMgBKGQWMmJOIR7esaFU1nZaOaSMoEKRUlsZvfVMSPk7qKOvCO5u doi: 10.1177/1833358318774357

Laporte, G., & Nobert, Y. (1987, 1). Exact Algorithms for the Vehicle Routing Problem. *North-Holland Mathematics Studies*, *132*(C), 147–184. doi: 10.1016/S0304-0208(08)73235-3

Lau, H. C. (1996, 1). On the complexity of manpower shift scheduling. *Computers & Operations Research*, *23*(1), 93–102. doi: 10.1016/0305-0548(94)00094-O

Lei, H., Laporte, G., & Guo, B. (2011, 12). The capacitated vehicle routing problem with stochastic demands and time windows. *Computers & Operations Research*, *38*(12), 1775–1783. doi: 10.1016/J.COR.2011.02.007

Lourenço, H. R., Martin, O. C., & Stützle, T. (2003, 2). Iterated Local Search. *Handbook of Metaheuristics*, 320–353. Retrieved from https://link-springer-com.tudelft.idm.oclc.org/chapter/10.1007/0-306-48056-5_11 doi: 10.1007/0-306-48056-5{_}11

Marzouk, M., & Kamoun, H. (2020). Nurse to patient assignment through an analogy with the bin packing problem: Case of a Tunisian hospital. *https://doi.org/10.1080/01605682.2020.1727300*, *72*(8), 1808–1821. Retrieved from https://www.tandfonline.com/doi/abs/10.1080/01605682.2020.1727300 doi: 10.1080/01605682.2020.1727300

Mor, A., Archetti, C., Jabali, O., Simonetto, A., & Speranza, M. G. (2022, 1). The Bi-objective Long-haul Transportation Problem on a Road Network. *Omega*, *106*, 102522. doi: 10.1016/J.OMEGA.2021.102522

Özarık, S. S., Veelenturf, L. P., Woensel, T. V., & Laporte, G. (2021, 4). Optimizing e-commerce last-mile vehicle routing and scheduling under uncertain customer presence. *Transportation Research Part E: Logistics and Transportation Review*, *148*, 102263. doi: 10.1016/J.TRE.2021.102263

Pereira, D. L., Alves, J. C., & Moreira, M. C. d. O. (2020, 6). A multiperiod workforce scheduling and routing problem with dependent tasks. *Computers & Operations Research*, *118*, 104930. doi: 10.1016/J.COR.2020.104930

Perumal, S. S., Dollevoet, T., Huisman, D., Lusby, R. M., Larsen, J., & Riis, M. (2021, 8). Solution approaches for integrated vehicle and crew scheduling with electric buses. *Computers & Operations Research*, *132*, 105268. doi: 10.1016/J.COR.2021.105268

Ropke, S., & Pisinger, D. (2006, 11). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *https://doi.org/10.1287/trsc.1050.0135*, *40*(4), 455–472. Retrieved from https://pubsonline.informs.org/doi/abs/10.1287/trsc.1050.0135 doi: 10.1287/TRSC.1050.0135

Sacramento, D., Pisinger, D., & Ropke, S. (2019, 5). An adaptive large neighborhood search metaheuristic for the vehicle routing problem with drones. *Transportation Research Part C: Emerging Technologies*, *102*, 289–315. doi: 10.1016/J.TRC.2019.02.018

Salani, M., & Vacca, I. (2011, 9). Branch and price for the vehicle routing problem with discrete split deliveries and time windows. *European Journal of Operational Research*, *213*(3), 470–477. Retrieved from https://ideas.repec.org/a/eee/ejores/v213y2011i3p470-477.htmlhttps://ideas.repec.org//a/eee/ejores/v213y2011i3p470-477.html doi: 10.1016/j.ejor.2011.03.023

Santini, A., Ropke, S., Lars, , Hvattum, M., Magnus, L., & No, H. H. (2018). A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. *J Heuristics*, *24*, 783–815. Retrieved from https://doi.org/10.1007/s10732-018-9377-x doi: 10.1007/s10732-018-9377-x

Sartori, C. S., Smet, P., & Vanden Berghe, G. (2022, 4). Scheduling truck drivers with interdependent routes under European Union regulations. *European Journal of Operational Research*, *298*(1), 76–88. doi: 10.1016/J.EJOR.2021.06.019

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *1520*, 417–431. Retrieved from https://link-springer-com.tudelft.idm.oclc.org/chapter/10.1007/3-540-49481-2_30 doi: 10.1007/3-540-49481-2{_}30/COVER

Solomon, M. M. (1987, 4). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *https://doi.org/10.1287/opre.35.2.254*, *35*(2), 254–265. Retrieved from https://pubsonline.informs.org/doi/abs/10.1287/opre.35.2.254 doi: 10.1287/OPRE.35.2.254

Spliet, R., & Desaulniers, G. (2015, 7). The discrete time window assignment vehicle routing problem. *European Journal of Operational Research*, *244*(2), 379–391. doi: 10.1016/J.EJOR.2015.01.020

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., & Subramanian, A. (2017, 3). New benchmark instances for the Capacitated Vehicle Routing Problem. *European Journal of Operational Research*, *257*(3), 845–858. doi: 10.1016/J.EJOR.2016.08.012

Van Den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., & De Boeck, L. (2013, 5). Personnel scheduling: A literature review. *European Journal of Operational Research*, *226*(3), 367–385. doi: 10.1016/J.EJOR.2012.11.029

Verweij, B., Ahmed, S., Kleywegt, A. J., Nemhauser, G., & Shapiro, A. (2003, 2). The sample average approximation method applied to stochastic routing problems: A computational study. *Computational Optimization and Applications*, *24*(2-3), 289–333. Retrieved from https://www.researchgate.net/publication/225929534_The_Sample_Average_Approximation_Method_Applied_to_Stochastic_Routing_Problems_A_Computational_Study doi: 10.1023/A:1021814225969

Wagenvoort, M., Bouman, P., van Ee, M., Lamballais Tessensohn, T., & Postek, K. (2022). An Exact and Heuristic Approach for the Ship-to-Shore Problem.

Wang, Y., Ma, X. L., Lao, Y. T., Yu, H. Y., & Liu, Y. (2014). A two-stage heuristic method for vehicle routing problem with split deliveries and pickups. *Journal of Zhejiang University: Science C*, *15*(3), 200–210. Retrieved from https://www.researchgate.net/publication/220371726_Heuristic_methods_for_vehicle_routing_problem_with_time_windows doi: 10.1631/JZUS.C1300177

Wen, M., Krapper, E., Larsen, J., & Stidsen, T. K. (2011, 12). A multilevel variable neighborhood search heuristic for a practical vehicle routing and driver scheduling problem. *Networks*, *58*(4), 311–322. Retrieved from https://onlinelibrary-wiley-com.tudelft.idm.oclc.org/doi/10.1002/net.20470 doi: 10.1002/NET.20470

Windras Mara, S. T., Norcahyo, R., Jodiawan, P., Lusiantoro, L., & Rifai, A. P. (2022, 10). A survey of adaptive large neighborhood search algorithms and applications. *Computers and Operations Research*, *146*. doi: 10.1016/J.COR.2022.105903

Witteman, M., Deng, Q., & Santos, B. F. (2021, 10). A bin packing approach to solve the aircraft maintenance task allocation problem. *European Journal of Operational Research*, *294*(1), 365–376. doi: 10.1016/J.EJOR.2021.01.027

Zhang, J., Lam, W. H., & Chen, B. Y. (2016, 2). On-time delivery probabilistic models for the vehicle routing problem with stochastic demands and time windows. *European Journal of Operational Research*, *249*(1), 144–154. doi: 10.1016/J.EJOR.2015.08.050

A Appendix A

Table 7: Mean realized objective value based on a certain number of forecast scenarios for a cost configuration of (1, 2, 20)

Instance	μ-1	μ-2	μ-3	μ-5	μ-10	μ-25	μ-50	μ-75	μ-100
C101	1,069	1,038	1,038	1,027	**1,024**	1,043	1,041	1,052	1,038
C201	627	623	**615**	625	630	627	638	634	622
R101	864	852	**844**	858	846	842	862	869	854
R201	727	723	723	726	722	**715**	719	723	721
RC101	1,056	**1,040**	1,060	1,042	1,060	1,052	1,042	1,057	1,050
RC201	852	835	848	859	828	**816**	856	834	838
Average	866	852	855	856	852	**849**	860	861	854

Note. The first column shows the instance. The other columns show the mean objective value (μ) for the actual scenarios for a certain number of forecast scenarios, which is indicated at the top as μ-number. Note that the objective has been divided by 10^3 for readability. Lastly, at the bottom row, the average objective value over the instances is calculated. Per row the lowest mean is highlighted in bold.

Table 8: Average results of different solving approaches for a cost configuration of (1, 2, 1) and 25 forecast scenarios

App-f	App-a	μ-a	σ-a	Rt-f (s)	Rt-a (s)	C_p	C_s-a	C_t-a
int	int	97,531	8,205	159	2,726	27,363	16,038	54,057
int	seq	98,696	16,792	159	259	27,363	27,065	44,193
int	si	97,453	16,029	159	1,068	27,363	25,769	44,247
seq	int	92,549	12,329	26	2,251	5,345	34,432	52,689
seq	seq	87,929	23,693	26	314	5,345	37,158	45,340
seq	si	**74,452**	9,693	26	1,002	5,345	23,714	45,312
si	int	91,209	11,351	79	2,005	6,313	31,511	53,303
si	seq	102,128	27,455	79	329	6,313	50,440	45,278
si	si	79,047	15,838	79	1,539	6,313	27,159	45,486

Note. Columns related to forecast scenarios are indicated by -f, columns related to actual scenarios by -a. The first and second column show the approach (App) used. Columns three and four show the mean realized objective value (μ) and standard deviation over the scenarios (σ). Columns five and six show the run time (Rt), in seconds. The last three columns show the average objective value divided into preference violation costs (C_p), schedule change costs (C_s) and routing costs (C_t). The lowest mean objective value is highlighted in bold. Recall that C_p, C_s and C_t might not add up to μ due to internal rounding.

B Appendix B

Here the VRPTW is formulated as a multi commodity network flow problem. The model contains two sets of decision variables x and s. For each arc (i,j), where $i \neq j,\ i \neq n+1,\ j \neq 0$, and each vehicle k, x_{ijk} is defined. If a vehicle k drives directly from vertex i to vertex j, $x_{ijk} = 1$, 0 otherwise. The decision variable s_{ik} is defined for each vertex i and each vehicle k and denotes the time vehicle k starts to service customer i. In case vehicle k does not service customer i, s_{ik} has no meaning. The mathematical formulation is as follows:

$$\min \sum_{k\in K}\sum_{i\in V}\sum_{j\in V} c_{ij}x_{ijk} \quad s.t., \tag{6}$$

$$\sum_{k\in K} v_{ik} \sum_{j\in V} x_{ijk} = 1 \quad \forall i \in V' \tag{7}$$

$$\sum_{j\in V} x_{0jk} = 1 \quad \forall k \in K \tag{8}$$

$$\sum_{i\in V} x_{i,n+1,k} = 1 \quad \forall k \in K \tag{9}$$

$$\sum_{i\in V'} z_i \sum_{j\in V} x_{ijk} \leq Q^k \quad \forall k \in K \tag{10}$$

$$\sum_{i\in V} x_{ihk} - \sum_{j\in V} x_{hjk} = 0 \quad \forall h \in V',\ \forall k \in K \tag{11}$$

$$s_{ik} + t_{ij} - M_{ij}(1 - x_{ijk}) \leq s_{jk} \quad \forall i,j \in V,\ \forall k \in K \tag{12}$$

$$e_i \leq s_{ik} \leq l_i \quad \forall i \in V,\ \forall k \in K \tag{13}$$

$$x_{ijk} \in \{0,1\} \quad \forall i,j \in V,\ \forall k \in K \tag{14}$$

The objective function (6) minimizes the total travel costs. Constraint (7) ensures each customer is visited exactly once by the correct truck. Constraints (8) and (9) ensure each vehicle leaves and enters the depot respectively. The capacity of a vehicle is enforced in constraint (10). Constraint (11) is the flow conservation constraint, that makes sure a truck leaves a customer after arriving. Constraint (12) describes the relationship in departure time from a customer and its immediate successor, where constraint (13) enforces the window at a customer. The integrality is described in (14). Note that M_{ij} can be decreased to $\max(l_i + t_{ij} - e_j), (i,j) \in A$.