Les Cahiers du GERAD ISSN: 0711–2440

A large neighborhood search for the vehicle routing problem withmultiple time windows

H. Schaap, M. Schiffer,
M. Schneider, G. Walther

G–2019–39

June 2019

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

A large neighborhood search for the vehicle routing problem with multiple time windows

Hendrik Schaap [a]

Maximilian Schiffer [b, c]

Michael Schneider [d]

Grit Walther [e]

[a] *Chair of Operations Management, School of Business and Economics, RWTH Aachen University*

[b] *GERAD, Montréal (Québec), Canada*

[c] *TUM School of Management, Technical University of Munich*

[d] *Deutsche Post Chair – Optimization of Distribution Networks, School of Business and Economics, RWTH Aachen University*

[e] *Chair of Operations Management, School of Business and Economics, RWTH Aachen University*

hendrik.schaap@om.rwth-aachen.de
schiffer@tum.de
schneider@dpo.rwth-aachen.de
walther@om.rwth-aachen.de

June 2019
Les Cahiers du GERAD
G–2019–39

Abstract: User-centered logistics aiming at customer satisfaction are gaining importance due to growing e-commerce and home deliveries. Customer satisfaction can be strongly increased by offering narrow delivery time windows. However, there is a tradeoff for the logistics provider because user-friendly delivery time windows might decrease the operational flexibility. Against this background, we study the Vehicle Routing Problem with Multiple Time Windows (VRPMTW) that determines a set of optimal routes such that each customer is visited once within one out of several time windows. We present a large neighborhood search based metaheuristic for the VRPMTW that contains a dynamic programming component to optimally select a time window for each customer on a route, and we present computationally efficient move descriptors for all search operators. We evaluate the performance of our algorithm on the Belhaiza instance set and provide new best known solutions for 26 out of 48 instances with improvements of up to 25.3% and 1.5% on average. Furthermore, we design new benchmark instances that reflect planning tasks in user-centered last mile logistics. Based on these, we present managerial studies that show the benefit of our algorithm for practitioners and allow to derive insights on how to offer time windows to customers. We show that offering multiple time windows can be economically beneficial for the logistics service providers and increases customer flexibility simultaneously.

Keywords: Vehicle routing, multiple time windows, efficient route evaluation, dynamic programming

Acknowledgments: Michael Schneider's research on efficient neighborhood search for routing problems was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant no. SCHN 1497/1–1.

1 Introduction

With increasing e-commerce and higher shipping volumes in parcel home deliveries, user-centered last-mile logistics services gain importance. To increase customer satisfaction, logistics service providers (LSPs) often offer a selection of multiple narrow time windows to customers or commit to precisely stating and realizing a certain delivery time window. Despite limiting their own operational planning with such offers, LSPs are willing to offer these services for two main reasons. First, LSPs seek to meet the expectations of e-commerce retailers and thus to remain competitive by avoiding the shift of shipping orders to competitors or the launch of own fleets of e-commerce retailers. Second, failed deliveries cause additional costs as they usually require a second delivery attempt.

For LSPs applying operations research tools to improve their operational planning, the vehicle routing problem (VRP) with multiple time windows (VRPMTW) describes this new setting in satisfactory manner because it allows to select a service time window for each customer out of a given set of potential time windows. Only little research exists on the VRPMTW so far. Favaretto et al. (2007) introduced the VRPMTW and proposed two different objective functions, either minimizing the total route duration or the total traveled distance. The authors proposed an ant colony optimization algorithm and solved new instances that were generated by adapting the VRP instances from Fisher (1994). Belhaiza et al. (2014) proposed a hybrid variable neighborhood tabu search heuristic that improved the results of Favaretto et al. (2007). Additionally, they derived new benchmark instances based on the customer patterns of the well-known Solomon instances (Solomon 1987). Belhaiza et al. (2017) further improved the results of Belhaiza et al. (2014) with a genetic variable neighborhood search.

Against this background, the contribution of this paper to the research field of VRPMTWs, which is likely to gain importance in today's user-centered last mile logistics, is severalfold. First, we present a large neighborhood search (LNS) based metaheuristic that provides a new state of the art for the VRPMTW in terms of solution quality. As core of this algorithm, we develop an exact component to optimally assign time windows to customers. Additionally, we give rigorous proofs to its computational complexity, and we present computationally efficient move descriptors for all search operators. Second, we carry out extensive numerical studies to evaluate the performance of our algorithm. We provide a concise Mixed Integer Programming (MIP) model and validate our algorithm on small instances against the MIP. We also compare our algorithm to existing algorithms on larger instances. Herein, we identify new best known solutions for 26 out of 48 instances, deriving improvements of up to 25.3% and 1.5% on average. Third, we design new large-scale benchmark instances that reflect planning tasks in user-centered last mile logistics. Based on these, we conduct managerial studies that highlight the benefit of our algorithm to improve daily operations. Additionally, these experiments allow insights for LSPs into the design of optimal time window offers. We show that the LSPs can achieve economic benefits by offering the right time windows while simultaneously increasing customer flexibility.

The remainder of this paper is structured as follows. Section 2 provides a MIP formulation for the VRPMTW. In Section 3, we present our algorithm including the optimal time window assignment component and prove its computational complexity. Section 4 presents our experimental design, numerical, and managerial studies. Finally, Section 5 concludes this paper with a short summary of its main findings.

2 The vehicle routing problem with multiple time windows

In this section, we introduce a MIP for the VRPMTW to provide a formal basis for our studies and to allow for comparisons between our metaheuristic and optimal solutions on small-sized instances.

Let $G = (\mathcal{V}, \mathcal{A})$ be a digraph with a set of vertices $\mathcal{V}$ and a set of arcs $\mathcal{A}$. The vertex set $\mathcal{V}$ consists of a set of customer vertices $\mathcal{C} = \{1, \dots, n-1\}$ and two vertices 0 and n that represent the depot, i.e., $\mathcal{V} = \mathcal{C} \dot\cup \{0, n\}$. To keep the notation concise, we introduce $\mathcal{C}^+ = \mathcal{C} \cup \{0\}$ and $\mathcal{C}^- = \mathcal{C} \cup \{n\}$. Each vertex $i \in \mathcal{V}$ is characterized by a service time s_i, a demand p_i, and a set $\Theta_i = \left\{ \left[e_i^1, l_i^1\right], \dots, \left[e_i^{\theta i}, l_i^{\theta i}\right] \right\}$ of θ_i time windows during which a start of service is allowed. Without loss of generality, each customer's time windows are disjoint and

chronologically ordered. We define $\mathcal{A} = \{(i,j) \mid i,j \in \mathcal{C}, i \neq j\} \cup \{(0,j) \mid j \in \mathcal{C}\} \cup \{(i,0) \mid i \in \mathcal{C}\}$. For each arc $(i,j) \in \mathcal{A}$, d_{ij} denotes its travel distance, t_{ij} its travel time, and c_{ij} its travel cost. We consider a homogeneous fleet of vehicles with a freight capacity F and fixed vehicle costs c^{f}. Binary variable x_{ij} denotes whether arc (i,j) is traversed ($x_{ij} = 1$) or not ($x_{ij} = 0$). To consider travel times, τ_i denotes the time at which the service at vertex $i \in \mathcal{V}$ starts. Further, f_i denotes the remaining freight load at vertex i. Binary variable u_p^i denotes whether service at vertex $i \in \mathcal{V}$ starts during its p-th time window ($u_p^i = 1$) or not ($u_p^i = 0$). With this notation as summarized in Table 1, we state a MIP for the VRPMTW:

$$\min \quad \sum_{j \in \mathcal{C}} c^{\mathrm{f}} x_{0j} + \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij}^r \tag{1}$$

s.t.

$$\sum_{j \in \mathcal{C}^- \setminus \{i\}} x_{ij} = 1 \qquad \forall i \in \mathcal{C} \tag{2}$$

$$\sum_{i \in \mathcal{C}^+ \setminus \{j\}} x_{ij} - \sum_{i \in \mathcal{C}^- \setminus \{j\}} x_{ji} = 0 \qquad \forall j \in \mathcal{C} \tag{3}$$

$$f_j \leq f_i - p_i x_{ij} + F(1 - x_{ij}) \qquad (i,j) \in \mathcal{A} \tag{4}$$

$$0 \leq f_i \leq F \qquad i \in \mathcal{V} \tag{5}$$

$$\tau_j \geq \tau_i + t_{ij} + s_i - l_n^{\theta_n}(1 - x_{ij}) \qquad \forall (i,j) \in \mathcal{A} \tag{6}$$

$$\sum_{p \in \{1,\ldots,\theta_i\}} u_p^i = 1 \qquad \forall i \in \mathcal{V} \tag{7}$$

$$\tau_i \geq e_i^p u_p^i \qquad \forall i \in \mathcal{V}, \forall p = 1,\ldots,\theta_i \tag{8}$$

$$\tau_i - l_n^{\theta_n}\left(1 - u_p^i\right) \leq l_i^p u_p^i \qquad \forall i \in \mathcal{V}, \forall p = 1,\ldots,\theta_i \tag{9}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A} \tag{10}$$

$$u_p^i \in \{0,1\} \qquad \forall i \in \mathcal{V},\ \forall p = 1,\ldots,\theta_i \tag{11}$$

The objective (1) minimizes total costs that consist of travel costs and fixed vehicle costs. Constraints (2) secure single assignment for each customer, while constraints (3) obtain flow conservation. Constraints (4) keep track of the freight balance, and constraints (5) ensure freight feasibility. Analogously, constraints (6) keep track of all time dependencies, and constraints (7)–(9) secure time window feasibility. Constraints (10) and(11) state the binary domains.

Table 1: Decision variables and parameter definitions for the mathematical model

$\mathcal{V}$	set of vertices
$\mathcal{A}$	set of arcs
$0, n$	instances of the depot
$\mathcal{C}$	set of customers
$\mathcal{C}^+$	set of customers including depot vertex 0
$\mathcal{C}^-$	set of customers including depot vertex n
d_{ij}	travel distance from vertex i to vertex j
t_{ij}	travel time from vertex i to vertex j
c_{ij}	travel cost from vertex i to vertex j
c^{f}	fixed cost for vehicles
e_i^p	earliest time of arrival at i to start service in time window p
l_i^p	latest time of arrival at i to start service in time window p
θ_i	number of time windows at customer i
s_i	service time at vertex i
p_i	demand at vertex i
F	vehicle freight capacity
x_{ij}	binary variable, 1 if arc (i,j) is traversed, 0 otherwise
τ_i	time at which the service at customer i starts
u_p^i	binary variable, 1 if service at customer i starts during its p-th time window, 0 otherwise
f_i	remaining freight capacity at vertex i

3 Solution methodology

The VRPMTW is NP-hard because it generalizes the VRP with time windows (VRPTW). Hence, we develop a LNS-based metaheuristic to solve large problem instances. First, we present our metaheuristic in Section 3.1. Then, Section 3.2 introduces our generalized objective function including penalty terms to efficiently handle infeasible solutions, and we describe a dynamic programming routine for optimal time window assignment in Section 3.3. Finally, Section 3.4 explains how to efficiently evaluate time window violations.

3.1 Large neighborhood search

During the development of our metaheuristic, we tested a large variety of algorithmic components and operators. In the following, we detail only the components that finally remained in our algorithm. For the sake of completeness, we provide a statistical analysis that quantifies the benefit of all tested components in Appendix D.

Introduced by Shaw (1998), LNS is a metaheuristic that allows for large-scale local search moves. The exploration of these large neighborhoods bases on a two-step mechanism, the destroy and repair phase. Herein, a randomly selected destroy operator removes a subset of customers from a solution. Then, a randomly selected repair operator inserts the removed customers in a certain fashion to construct a new solution. With this basic mechanism, LNS allows to overcome local optima, and the paradigm has successfully been used to solve various VRP variants (cf., e.g., Ropke and Pisinger 2006, Kytöjoki et al. 2007, Mattos Ribeiro and Laporte 2012). For a detailed description of LNS, we refer to Pisinger and Ropke (2010).

Figure 1 shows the pseudocode of our metaheuristic. After some preprocessing steps (Section 3.1.1), we create an initial solution (Section 3.1.2) that is then improved. Our LNS stops either after $i^{\max}$ total iterations or after $i^{\max}_{\text{noi}}$ iterations without improvement. During the search, we allow infeasible solutions and use penalty terms in our objective function (Section 3.2) to explore neighborhoods that contain only few feasible solutions (cf. Cordeau et al. 2001). Hence, besides the current solution σ, we store the temporal solution (σ'), the best solution (σ^*), and the best feasible solution (σ^*_{f}) during our search. In each iteration, we apply a destroy and repair step (Section 3.1.3) to create a new temporal solution from σ. Afterwards, we apply a local search (Section 3.1.4) for intensification if the objective value of σ' is within a range of $(1+\delta)$ of the so far best objective function value $\lambda(\sigma^*)$. During our search, we forward solutions as follows. We forward the temporal solution σ' to σ and σ^* if it yields an improvement. If σ' improves σ^*, we try to generate a feasible solution out of σ' using the repair method of Vidal et al. (2014). If σ' remains infeasible afterwards, no further efforts are taken to generate a feasible solution. We forward σ' to σ^*_{f} if it is feasible

```
1:  i ← 0, i_noi ← 0
2:  σ ← initialSolution()
3:  while i ≤ i^max and i_noi ≤ i^max_noi do
4:      σ' ← destroyAndRepairSolution(σ)
5:      if λ(σ') < (1 + δ)λ(σ*) then
6:          σ' ← localSearch(σ')
7:      if λ(σ') < λ(σ) then
8:          σ ← σ'
9:          if λ(σ') < λ(σ*) then
10:             σ* ← σ'
11:             σ' ← makeFeasible(σ')
12:             if isFeasible(σ') and λ(σ') < λ(σ*_f) then
13:                 σ*_f ← σ
14:                 i_noi ← 0
15:     if   mod (i, i^r) = 0 then
16:         σ ← σ*
17:     if i = ⌊getFirstEntry(I_switch) · i^max⌋ then
18:         switchTimeWindowAssignment()
19:         removeFirstEntry(I_switch)
20:     i ← i + 1, i_noi ← i_noi + 1
```

Figure 1: Pseudocode of the LNS

and yields an improvement. We reset σ to σ^* every i^{r} iterations. To accelerate our local search, we switch between an optimal and a heuristic assignment of time windows to customers. We always start the search using the heuristic assignment method in which we consider only l^{path} different combinations of time window assignments (cf. Section 3.2). Then, we switch to the optimal assignment scheme and back every time the number of passed iterations i matches a share of the total iterations $i^{\max}$ that is stated in I_{switch}. Note that I_{switch} is a vector that allows for a varying step width between these switches.

3.1.1 Preprocessing

To speed up the search, we limit neighborhoods by identifying infeasible arcs during preprocessing. An arc (i, j) is infeasible if:

1. its demand exceeds the vehicle capacity, i.e.,

$$p_i + p_j > F \tag{12}$$

2. it is impossible to arrive at v_j before its last time window closes after visiting v_i, i.e.,

$$e_i^0 + s_i + t_{ij} > l_j^{\theta_j} \tag{13}$$

3. it is impossible to visit v_j after v_i and arrive at the depot before its last time window closes, i.e.,

$$e_i^0 + s_i + t_{ij} + s_j + t_{jn} > l_n^{\theta_n}. \tag{14}$$

To avoid visiting solutions with less vehicles than necessary to fulfill freight capacity constraints, we calculate a lower bound on the number of vehicles to fulfill all customer demands by

$$\left\lceil \frac{\sum_{v \in \mathcal{C}} p_v}{F} \right\rceil. \tag{15}$$

3.1.2 Initial solution

To create an initial solution, we modify the basic savings algorithm (cf. Clarke and Wright 1964). First, we calculate back-and-forth tours for all customers. Afterwards, we calculate all potential cost savings from routes that can be merged without using an infeasible arc and sort them in decreasing order. Then, we iteratively merge the two routes with the highest savings as long as the freight capacity constraint is not violated. We drop merge moves that are no longer feasible due to previous moves and stop once all merge moves with positive savings have been executed or dropped. Afterwards, we apply our local search (Section 3.1.4) to improve the initial solution.

3.1.3 Destroy and repair

In the destroy and repair phase, we create a new temporal solution by first randomly choosing a destroy operator that removes customers from the current solution. Then, we use a repair operator that reinserts these customers to create a new temporal solution.

We use the following set of destroy operators: The `worstRemove` operator (Ropke and Pisinger 2006) removes customers hierarchically according to their savings potential, starting with the highest savings. The operator stops after removing a randomly chosen percentage share of customers out of the range $[\Gamma_{\min}, \Gamma_{\max}]$. Additionally, we use a `routeRemove` operator (Hemmelmayr et al. 2012) that removes a randomly chosen route as long as more routes exist than necessary according to (15). Furthermore, we design a new `relatedRemove` operator. The `relatedRemove` operator removes randomly chosen customers in a pairwise fashion. While the first customer is always deleted, the second customer is deleted with a certain probability that correlates with the relatedness between both customers (cf. Shaw 1998). While Shaw (1998) used a purely distance based relatedness measure, we define a new measure that captures the complexity of the VRPMTW. Our

relatedness R_{ij} comprises four different components, considering spatial relatedness R^{s}_{ij}, demand relatedness R^{p}_{ij}, time window relatedness R^{t}_{ij}, and route relatedness R^{r}_{ij}. We use a parameter β for each quantity to adjust its influence.

$$R_{ij} = \beta^{\mathrm{s}} R^{\mathrm{s}}_{ij} + \beta^{p} R^{\mathrm{p}}_{ij} + \beta^{\mathrm{t}} R^{\mathrm{t}}_{ij} + \beta^{\mathrm{r}} R^{\mathrm{r}}_{ij} \tag{16}$$

$$= \beta^{\mathrm{s}} \cdot \frac{d_{ij}}{d_{\max}} + \beta^{\mathrm{p}} \cdot \frac{|p_i - p_j|}{p_{\max} - p_{\min}} + \beta^{\mathrm{t}} \cdot \omega_{ij} + \beta^{\mathrm{r}} \cdot \mathbb{1}_{ij}. \tag{17}$$

The spatial relatedness results straightforwardly from the distance d_{ij} and the maximum distance between two vertices ($d_{\max}$), while the demand relatedness results from demand of i and j as well as the maximum and minimum demand over all customers (p_{max}, p_{min}). The route relatedness is based on a boolean indicator ($\mathbb{1}_{ij}$) that tells whether two customers are on the same route. R^{t}_{ij} depends on the relative time window overlap ω_{ij} between i and j. We define this overlap for a finite union of disjoint intervals as

$$\omega_{ij} = \frac{\left| \left(\bigcup_{w \in \Theta_i} w \right) \cap \left(\bigcup_{w \in \Theta_j} w \right) \right|}{\max \left\{ \left| \bigcup_{w \in \Theta_i} w \right|, \left| \bigcup_{w \in \Theta_j} w \right| \right\}}, \tag{18}$$

denoting the cumulated length of these intervals by $|\cdot|$. We determine the number of pairs of vertices to be removed randomly within a percentage range $\left[\frac{\Gamma_{\min}}{2}, \frac{\Gamma_{\max}}{2}\right]$ of the total number of vertices.

We use a `sequentialInsertion` repair operator (Hiermann et al. 2016) and insert vertices in the order in which they have been removed randomly with a probability that is inverse to the cost of the respective insertion. Additionally, we us a `sequentialPertubatedInsertion` operator that pertubates the calculated insertion cost by up to 20%.

3.1.4 Local search

In our local search, we use a composite neighborhood that consists of four different neighborhood operators. A `relocate` operator randomly removes and inserts a vertex into the same or a different route. The `exchange` operator swaps two vertices in the same or between different routes. The `2-opt*` operator (Potvin and Rousseau 1995) removes two edges from the existing solution and introduces two new edges, connecting the beginning of one tour with the end of the other tour and vice versa. The `Or-opt` operator (Or 1976) removes a sequence of one, two, or three consecutive vertices from a route and reinserts this sequence between a pair of consecutive vertices on another route.

We use a best improvement acceptance criterion because we found during the development of our algorithm that a first improvement acceptance criterion significantly worsens the solution quality. Further, we use a variable descent neighborhood evaluation scheme: in the beginning of the search, we limit the evaluated neighborhood to the η^{c} closest vertices. If no further improvements are found, we allow search moves with vertices within the η^{m} closest neighbors, before we finally allow searching the complete neighborhood. This evaluation scheme helps to significantly reduce the computational time of the local search without deteriorating its solution quality.

3.2 Generalized objective function

We use a generalized objective function that allows to handle infeasible solutions by incorporating penalty terms for freight capacity and time window violations. We represent a solution $\sigma = \{r_1, \ldots, r_l\}$ as a set of routes r_i, and refer to a route r with $k_r - 1$ customers as an ordered set of vertices $r = \langle v_0, \ldots, v_{k_r} \rangle$ in which v_0 and v_{k_r} denote the depot vertices. With this notation, our generalized objective function is

$$\lambda^{\mathrm{gen}}(\sigma) = \sum_{r \in \sigma} \left(c^{\mathrm{f}}_r + \sum_{i=1}^{k_r} c_{v_{i-1}, v_i} \right) + \alpha^{\mathrm{fr}} FR(\sigma) + \alpha^{\mathrm{tw}} TW(\sigma) \tag{19}$$

with fixed costs per route c_r^{f}, costs for each traversed arc c_{ij}, freight capacity violation $FR(\sigma)$ and time window violation $TW(\sigma)$. To control the impact of these violations, we multiply them with α^{fr} and α^{tw} respectively to obtain the final penalty term. We multiply (resp. divide) these factors by γ every i_{pLS} local search iterations if a violation (resp. no violation) arises in these iterations. This allows to overcome local optima or to enforce a feasible solution by decreased (resp. increased) penalty terms. To limit the impact of penalty terms, we limit α^{fr} and α^{tw} to $\alpha^{\mathrm{fr}}_{\mathrm{min}}$, $\alpha^{\mathrm{tw}}_{\mathrm{min}}$, and $\alpha^{\mathrm{fr}}_{\mathrm{max}}$, $\alpha^{\mathrm{tw}}_{\mathrm{max}}$. In the following, we describe how to efficiently calculate $FR(\sigma)$ and $TW(\sigma)$.

3.2.1 Calculation of time window violations

For customers with a single time window, a time window violation occurs if a vehicle arrives at a customer after its time window. For this case, computationally efficient penalty terms exist (cf. Nagata et al. 2010, Schneider et al. 2013) that allow for an accurate calculation of violations by using the concept of time travel: after each violation, the vehicle's arrival time is shifted back to the latest feasible arrival time at a vertex to avoid overpenalization.

While calculating time window violations and using the concept of time travel in the single time window case is relatively simple and state-of-the-art, the multiple time window case adds additional complexity. When evaluating a route, each time a vehicle arrives at a customer in between two time windows, one can either decide to penalize the violation of the first time window or to proceed with using the second time window. However, using the second time window may cause (higher) violations at succeeding customers. Figure 2 shows this trade-off for a sequence of customers. While the matching of the late time window at customer C1 results in a violation at customer C2, the penalizing of the early time window at C1 however yields feasibility at C2 and results in an overall lower penalty. In case the violated time window is chosen, the time window violation also denotes the required time travel that is necessary to shift the arrival time back to the latest feasible point in time for succeeding evaluations.

In the following, we provide a notation to calculate time window violations in the multiple time window case. This notation provides the formal foundation to develop an exact component that handles the time window assignment in the best possible way in Sections 3.3 and 3.4. In the multiple time window case, one time window $[e_v^i, l_v^i],\ i \in \{1, ..., \theta_i\}$ must be assigned to every customer v to allow for the evaluation of violations. To keep track of the assigned time windows for an arbitrary (partial) route, we define a *partial time window assignment* $f^i \in \mathcal{F}^i(r) = \{1, \ldots, \theta_0\} \times \cdots \times \{1, \ldots, \theta_i\}$, which denotes a vector with all assigned time windows up to customer i on route r. We refer to the j-th entry of f^i as f^i_j. Each f^i_j denotes the index of the chosen time window. Analogously, $f = f^{i=k} \in \mathcal{F}^{i=k}(r) = \mathcal{F}(r)$ denotes a complete *time window assignment* for a given route. With this notation, we calculate the earliest start time of service $a_j^{f^i}$ for each customer j of a given (partial) time window assignment f^i (Equations (20) and (21)) and its time travel counterpart $\tilde{a}_j^{f^i}$ (22) that avoids overpenalization.

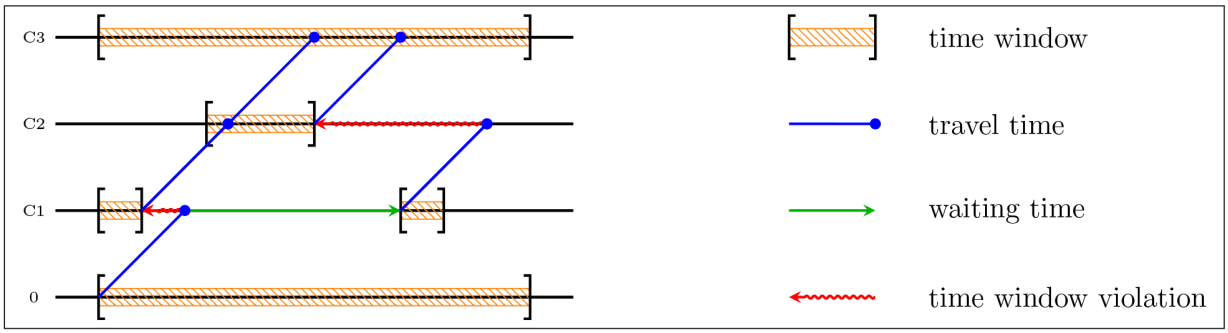


Figure 2: Trade–off between penalizing early or matching late time windows

$$a_0^{f^i} = \tilde{a}_0^{f^i} = e_0^{f_0^i} \tag{20}$$

$$a_j^{f^i} = \tilde{a}_{j-1}^{f^i} + s_{j-1} + t_{j-1,j} \qquad 1 \leq j \leq i \tag{21}$$

$$\tilde{a}_j^{f^i} = \begin{cases} \max\left\{a_j^{f^i}, e_j^{f_j^i}\right\} & \text{if } a_j^{f^i} \leq l_j^{f_j^i} \\ l_j^{f_j^i} & \text{otherwise.} \end{cases} \qquad 1 \leq j \leq i \tag{22}$$

With these definitions, the time window violation for a fixed f^i results to

$$\overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) = \sum_{j=0}^{i} \max\left\{a_j^{f^i} - l_j^{f_j^i}, 0\right\}. \tag{23}$$

For $i = k$ we introduce $TW(r, f) = \overrightarrow{\lambda}_k^{\text{tw}}(r, f^k)$ to denote the time window violation for the entire route with respect to f^k. Without a given time window assignment, determining the minimum time window violation along a route requires to solve a subsequent optimization problem of the form

$$TW(r) = \min_{f \in \mathcal{F}(r)} TW(r, f), \tag{24}$$

each time a route is evaluated. Here, the search space $\mathcal{F}(r)$ is of size $\prod_{i=0}^{k} \theta_i$ so that a simple enumeration algorithm yields an exponential complexity. To efficiently solve this problem, we introduce a dynamic programming algorithm in Sections 3.3 and 3.4.

3.2.2 Calculation of freight capacity violations

A freight capacity violation occurs if the cumulated demand of all customers on a route exceeds the vehicle capacity. For a solution σ, the freight capacity violation results from the violation of each route:

$$FR(\sigma) = \sum_{r \in \sigma} FR(r) = \sum_{r \in \sigma} \max\{-F + \sum_{i=0}^{k} p_{v_i}, 0\}.$$

By keeping track of forward and backward freight capacity violations, the freight capacity penalty can be updated in $\mathcal{O}(1)$ for any search move (cf. Kindervater and Savelsbergh 1997).

3.3 A dynamic programming routine for optimal time window sequencing

In this section, we develop a dynamic programming algorithm to efficiently determine the optimal time window assignment for a route r. We first introduce an adequate search tree representation (Section 3.3.1), before we derive a dominance criterion to reduce the computational complexity (Section 3.3.2). Then, we describe the resulting dynamic programming routine (Section 3.3.3). Finally, we prove the overall complexity of the resulting algorithm by deriving bounds on the size of the search tree (Section 3.3.4).

3.3.1 Search tree representation

We define a search tree (cf. Example 1) in which edges represent the propagation of a partial time window assignment from one to the next customer's time windows so that each path from a node to the tree's root node represents a partial time window assignment. In this tree, we label each node with its corrected time of arrival $\tilde{a}_j^{f^j}$ and the cumulated time window violation up to this customer $\overrightarrow{\lambda}_j^{\text{tw}}(r, f^j)$. We always start at the root level L_0 with a single root node, labeled with $(e_0^0, 0)$ to construct this tree. Then, we develop each subsequent level L_{i+1} by calculating the arrival time at vertex $i+1$ starting from the arrival time of each node in level L_i. For each time window at vertex $i+1$, we introduce a new node that contains the corrected arrival time and the resulting total time window violation. Note that the resulting search tree grows exponentially.

Example 1 (Exponential search tree) *We consider a route* $r = \langle 0, 1, 2, 3, 4 \rangle$ *consisting of the three customer vertices 1, 2, and 3 and two depot vertices 0 and 4. The travel time between each pair of consecutive vertices is* $t_{i,i+1} = 2$ *and the service time is* $s_i = 0$ *for all vertices. Figure 3 depicts the time windows of the vertices. Figure 4 shows the resulting fully enumerated search tree. Its size is exponential in the number of time windows per customer.*

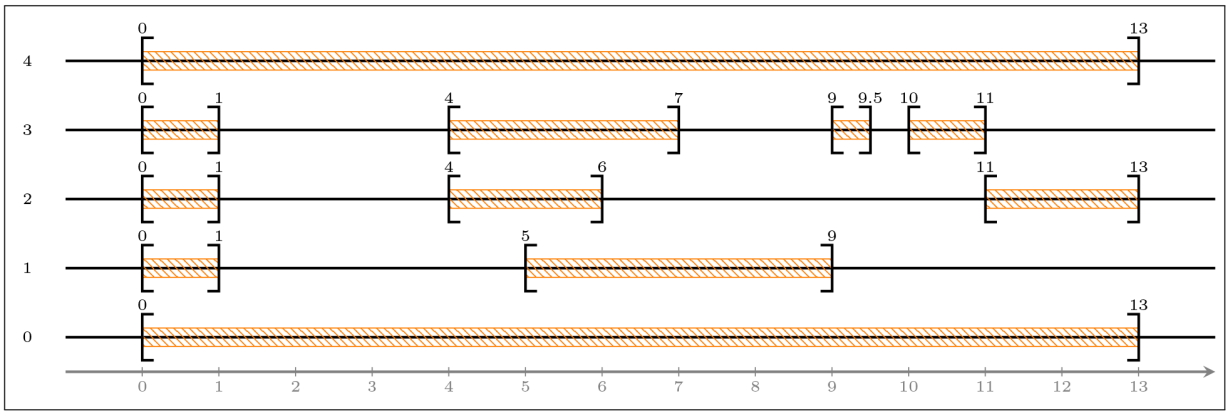


Figure 3: Time window distribution for the route in Example 1

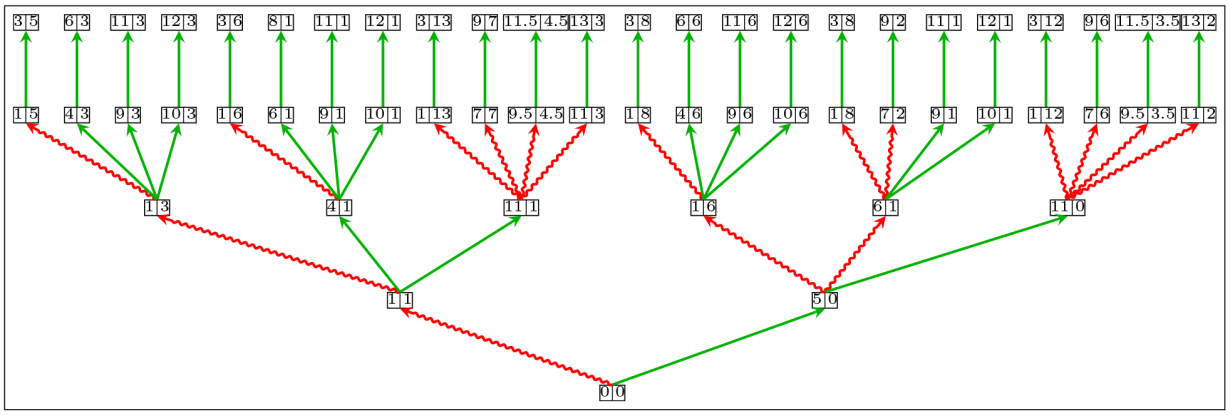


Figure 4: Exponential search tree

3.3.2 Dominance criterion

Because we need to determine optimal time window assignments in every search move, evaluating a fully enumerated exponential search tree is computationally not tractable. Hence, we proof a dominance criterion to cut redundant branches and limit the size of the search tree accordingly.

In the following, we refer to a *time window assignment remainder* as $\hat{f}^i \in \{1, \dots, \theta_{i+1}\} \times \cdots \times \{1, \dots, \theta_k\} = \hat{\mathcal{F}}^i(r)$, and denote the concatenation of vectors $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ as $a \oplus b \in \mathbb{R}^{n+m}$ for $m, n \in \mathbb{N}$. Further, we use $\hat{f}^i_j$ to denote the time window assignment decision at customer j, $j > i$, i.e., the $(j-i)$th entry in $\hat{f}^i$. With this notation, we derive a dominance criterion based on Definition 1 and Lemma 1.

Definition 1 (Dominant time window assignments) *For a customer* $i \in \{0, \dots, k\}$ *on a route* r*, a partial time window assignment* $f^i \in \mathcal{F}^i(r)$ *dominates a partial time window assignment* $g^i \in \mathcal{F}^i(r)$ *if some (possibly empty) time window assignment remainder* $\hat{f}^i \in \hat{\mathcal{F}}^i(r)$ *exists such that Equation (25) holds for all* $\hat{g}^i \in \hat{\mathcal{F}}^i(r)$*.*

$$TW(r, f^i \oplus \hat{f}^i) \leq TW(r, g^i \oplus \hat{g}^i) \tag{25}$$

Based on Definition 1, Lemma 1 states our dominance criterion.

Lemma 1 *Let r be a route and let $f^i, g^i \in \mathcal{F}^i(r)$ be partial time window assignments of the customers in r up to customer $i \in \{0, \ldots, k-1\}$. If Equations (26) and (27) hold, f^i dominates g^i.*

$$\overrightarrow{\lambda}^{tw}_i(r, f^i) \leq \overrightarrow{\lambda}^{tw}_i(r, g^i) \tag{26}$$

$$\tilde{a}^{f^i}_i - \tilde{a}^{g^i}_i \leq \overrightarrow{\lambda}^{tw}_i(r, g^i) - \overrightarrow{\lambda}^{tw}_i(r, f^i) \tag{27}$$

Proof of Lemma 1. Let

$$\hat{g}^* = argmin_{\hat{g}^i \in \hat{\mathcal{F}}^i(r)} TW(r, g^i \oplus \hat{g}^i) \tag{28}$$

be a time window assignment remainder minimizing the time window violation if time windows are assigned according to g^i up to customer i. We show that extending f^i by $\hat{g}^*$, i.e., $f^* = f^i \oplus \hat{g}^*$, never yields a higher total time window violation than $g^* = g^i \oplus \hat{g}^*$. As $\hat{g}^*$ denotes an optimal time window assignment remainder for g^i, this implies dominance of f^i over g^i. We introduce θ^* as the time window assigned to customer $i+1$ by $\hat{g}^*$.

It is sufficient to prove Equations (29) and (30), i.e., that Equations (26) and (27) hold for $i+1$ with $f^i \oplus (\theta^*)$ and $g^i \oplus (\theta^*)$ instead of f^i and g^i, because either $i+1$ equals k and Equation (29) entails Equation (25), or the proof follows by induction.

$$\overrightarrow{\lambda}^{\text{tw}}_{i+1}(r, f^i \oplus (\theta^*)) \leq \overrightarrow{\lambda}^{\text{tw}}_{i+1}(r, g^i \oplus (\theta^*)) \tag{29}$$

$$\tilde{a}^{f^i \oplus (\theta^*)}_{i+1} - \tilde{a}^{g^i \oplus (\theta^*)}_{i+1} \leq \overrightarrow{\lambda}^{\text{tw}}_{i+1}(r, g^i \oplus (\theta^*)) - \overrightarrow{\lambda}^{\text{tw}}_{i+1}(r, f^i \oplus (\theta^*)) \tag{30}$$

To prove equations (29) and (30), we distinguish six cases dependent on the relative position of θ^* to the earliest possible times of arrival $a^{f^i \oplus (\theta^*)}_{i+1}$ and $a^{g^i \oplus (\theta^*)}_{i+1}$ and refer to the earlier arrival time as

$$a^{min}_{i+1} = \min \left\{ a^{f^i \oplus (\theta^*)}_{i+1}, a^{g^i \oplus (\theta^*)}_{i+1} \right\}$$

and to the later arrival time as

$$a^{max}_{i+1} = \max \left\{ a^{f^i \oplus (\theta^*)}_{i+1}, a^{g^i \oplus (\theta^*)}_{i+1} \right\}$$

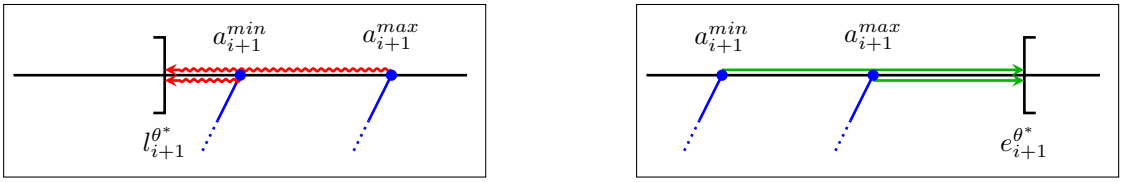
to keep the notation concise.

Case I) $a^{min}_{i+1} > l^{\theta^*}_{i+1}$ (see Figure 5a)
We correct both earliest possible times of arrival to the assigned time window's closing time because they are larger than the time window's closing time. An additional violation of $\tilde{a}^{f^i}_i + s_i + t_{i,i+1} - l^{\theta^*}_{i+1}$ resp. $\tilde{a}^{g^i}_i + s_i + t_{i,i+1} - l^{\theta^*}_{i+1}$ results so that

$$\begin{aligned}
\overrightarrow{\lambda}^{\text{tw}}_{i+1}(r, f^i \oplus (\theta^*)) &= \overrightarrow{\lambda}^{\text{tw}}_i(r, f^i) + \tilde{a}^{f^i}_i + s_i + t_{i,i+1} - l^{\theta^*}_{i+1} \\
&\overset{(27)}{\leq} \overrightarrow{\lambda}^{\text{tw}}_i(r, g^i) + \tilde{a}^{g^i}_i + s_i + t_{i,i+1} - l^{\theta^*}_{i+1} \\
&= \overrightarrow{\lambda}^{\text{tw}}_{i+1}(r, g^i \oplus (\theta^*)).
\end{aligned}$$

Thus Equation (29) holds. Because $\tilde{a}^{f^i \oplus (\theta^*)}_{i+1} = \tilde{a}^{g^i \oplus (\theta^*)}_{i+1} = l^{\theta^*}_{i+1}$ holds, Equations (29) and (30) coincide.

Case II) $a^{max}_{i+1} < e^{\theta^*}_{i+1}$ (see Figure 5b)
Both earliest possible times of arrival are smaller than the assigned time window's opening time. This implies that $\tilde{a}^{f^i \oplus (\theta^*)}_{i+1}$ and $\tilde{a}^{g^i \oplus (\theta^*)}_{i+1}$ coincide at $e^{\theta^*}_{i+1}$, and no additional violation occurs for both cases. Then, partial time window violations in Equations (29) and (30) result directly from Equations (26) and (27).



(a) Case I **(b) Case II**

Figure 5: Possible positions of a_{i+1}^{min} and a_{i+1}^{max} in relation to the time window $\left[e_{i+1}^{\theta^*}, l_{i+1}^{\theta^*}\right]$

Case III) $a_{i+1}^{min} < e_{i+1}^{\theta^*} \leq a_{i+1}^{max} \leq l_{i+1}^{\theta^*}$ (see Figure 6a)
If a_{i+1}^{min} is smaller than the opening time of θ^* and a_{i+1}^{max} lies within θ^*, no additional violation arises for either of the partial time window assignments, and Equation (29) holds. Depending on whether a_{i+1}^{min} equals $a_{i+1}^{f^i \oplus (\theta^*)}$ or $a_{i+1}^{g^i \oplus (\theta^*)}$, the left-hand side of Equation (30) either decreases compared to Equation (27) or is negative. Equation (29) implies non-negativity of the right-hand side of Equation (30), which guarantees validity of Equation (30).

Case IV) $e_{i+1}^{\theta^*} \leq a_{i+1}^{min} \leq l_{i+1}^{\theta^*} < a_{i+1}^{max}$ (see Figure 6b)
If a_{i+1}^{min} lies within θ^* and a_{i+1}^{max} is greater than the closing time of θ^*, we correct a_{i+1}^{max} to $l_{i+1}^{\theta^*}$ causing an additional violation of $\delta = a_{i+1}^{max} - l_{i+1}^{\theta^*}$. We consider two cases:

i) For $a_{i+1}^{min} = a_{i+1}^{g^i \oplus (\theta^*)}$ and $a_{i+1}^{max} = a_{i+1}^{f^i \oplus (\theta^*)}$ this yields

$$\begin{aligned} 0 \leq \tilde{a}_{i+1}^{f^i \oplus (\theta^*)} - \tilde{a}_{i+1}^{g^i \oplus (\theta^*)} &= \tilde{a}_i^{f^i} + s_i + t_{i,i+1} - \delta - \left(\tilde{a}_i^{g^i} + s_i + t_{i,i+1}\right) \\ &= \tilde{a}_i^{f^i} - \tilde{a}_i^{g^i} - \delta \\ &\overset{(27)}{\leq} \overrightarrow{\lambda}_i^{\text{tw}}(r, g^i) - \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) - \delta \\ &= \overrightarrow{\lambda}_i^{\text{tw}}(r, g^i) - \left(\overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) + \delta\right) \\ &= \overrightarrow{\lambda}_{i+1}^{\text{tw}}(r, g^i \oplus (\theta^*)) - \overrightarrow{\lambda}_{i+1}^{\text{tw}}(r, f^i \oplus (\theta^*)) \end{aligned}$$

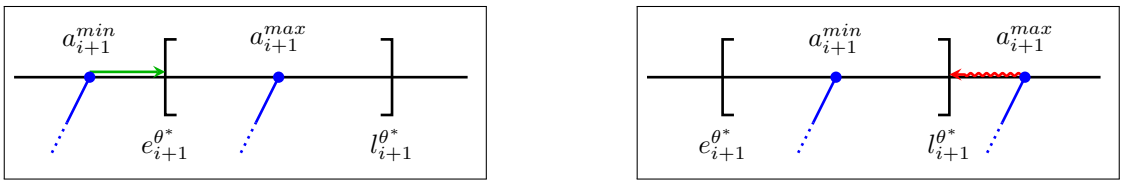
which establishes Equations (29) and (30).

ii) If $a_{i+1}^{min} = a_{i+1}^{f^i \oplus (\theta^*)}$ and $a_{i+1}^{max} = a_{i+1}^{g^i \oplus (\theta^*)}$ hold, the following inequality can be derived to prove Equation (30):

$$\begin{aligned} \tilde{a}_{i+1}^{f^i \oplus (\theta^*)} - \tilde{a}_{i+1}^{g^i \oplus (\theta^*)} &= \tilde{a}_i^{f^i} + s_i + t_{i,i+1} - \left(\tilde{a}_i^{g^i} + s_i + t_{i,i+1} - \delta\right) \\ &= \tilde{a}_i^{f^i} - \tilde{a}_i^{g^i} + \delta \\ &\overset{(27)}{\leq} \overrightarrow{\lambda}_i^{\text{tw}}(r, g^i) - \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) + \delta \\ &= \left(\overrightarrow{\lambda}_i^{\text{tw}}(r, g^i) + \delta\right) - \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) \\ &= \overrightarrow{\lambda}_{i+1}^{\text{tw}}(r, g^i \oplus (\theta^*)) - \overrightarrow{\lambda}_{i+1}^{\text{tw}}(r, f^i \oplus (\theta^*)). \end{aligned}$$

Because an additional violation arises for $a_{i+1}^{g^i \oplus (\theta^*)}$, Equation (31) proves the validity of Equation (30)

$$\overrightarrow{\lambda}_{i+1}^{\text{tw}}(r, f^i \oplus (\theta^*)) = \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) \leq \overrightarrow{\lambda}_i^{\text{tw}}(r, g^i) \leq \overrightarrow{\lambda}_i^{\text{tw}}(r, g^i) + \delta = \overrightarrow{\lambda}_{i+1}^{\text{tw}}(r, g^i \oplus (\theta^*)). \tag{31}$$



(a) Case III **(b) Case IV**

Figure 6: Possible positions of a_{i+1}^{min} and a_{i+1}^{max} in relation to the time window $\left[e_{i+1}^{\theta^*}, l_{i+1}^{\theta^*}\right]$

Case V) $a_{i+1}^{min} < e_{i+1}^{\theta^*} \leq l_{i+1}^{\theta^*} < a_{i+1}^{max}$ (see Figure 7a)
If a_{i+1}^{min} is smaller than the opening time and a_{i+1}^{max} is greater than the closing time of θ^*, we obtain Equations (29) and (30) by combining the reasoning of III and IV, i.e., first adjusting a_{i+1}^{min} to $e_{i+1}^{\theta^*}$ and afterwards a_{i+1}^{max} to $l_{i+1}^{\theta^*}$.

Case VI) $e_{i+1}^{\theta^*} \leq a_{i+1}^{min} \leq a_{i+1}^{max} \leq l_{i+1}^{\theta^*}$ (cf. Figure 7b)
If a_{i+1}^{min} and a_{i+1}^{max} lie within θ^*, no violation results and no waiting time is needed. We derive the validity of Equations (29) and (30) directly from Equations (26) and (27).

This concludes the proof. □

Corollary 1 *Let r be a route and let $f^i, g^i \in \mathcal{F}^i(r)$ be partial time window assignments of the customers in r up to customer $i \in \{0, \ldots, k-1\}$. If $\overrightarrow{\lambda}_i^{tw}(r, f^i) \leq \overrightarrow{\lambda}_i^{tw}(r, g^i)$ and $\tilde{a}_i^{f^i} = \tilde{a}_i^{g^i}$ hold, f^i dominates g^i.*

Corollary 1 states a special case of Lemma 1 as many corrected earliest possible arrival times coincide because all earliest possible arrival times that lie in between two time windows are corrected either to the opening time of a later time window or to the closing time of an earlier time window.

To further reduce the size of the search tree, we proof that waiting for opening times of time windows longer than necessary leads to dominated time window assignments by means of the following Lemmata 2 and 3 and Corollaries 2 and 3. The same holds for traveling back in time further than necessary.

Lemma 2 *If for a given f^i the earliest possible time of arrival at customer $i+1$ is greater than e_{i+1}^j, $f^i \oplus (j)$ dominates $f^i \oplus (j')$ for all $j' \in \{1, \ldots, j-1\}$.*

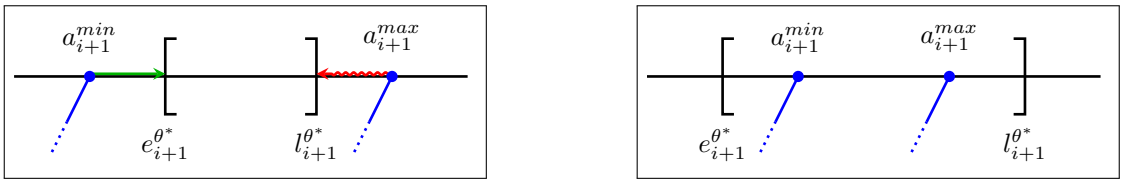
Proof of Lemma 2. Let $\delta = \max\left\{\tilde{a}_i^{f^i} + s_i + t_{i,i+1} - l_{i+1}^j, 0\right\}$ and $\delta' = \tilde{a}_i^{f^i} + s_i + t_{i,i+1} - l_{i+1}^{j'}$ be the time window violations to shift $\tilde{a}_i^{f^i} + s_i + t_{i,i+1}$ to l_{i+1}^j resp. $l_{i+1}^{j'}$ (see Figure 8a). Then,

$$\overrightarrow{\lambda}_{i+1}^{\text{tw}}(r, f^i \oplus (j')) = \overrightarrow{\lambda}_{i+1}^{\text{tw}}(r, f^i \oplus (j)) + \delta' - \delta \tag{32}$$

and

$$\tilde{a}_{i+1}^{f^i \oplus (j')} = \tilde{a}_{i+1}^{f^i \oplus (j)} - (\delta' - \delta) \tag{33}$$

hold and Equation (32) implies Equation (26). Combining Equations (32) and (33) implies Equation (27)



(a) Case V **(b) Case VI**

Figure 7: Possible positions of a_{i+1}^{min} and a_{i+1}^{max} in relation to the time window $\left[e_{i+1}^{\theta^*}, l_{i+1}^{\theta^*}\right]$

so that according to Lemma 1, $f^i \oplus (j)$ dominates $f^i \oplus (j')$. □

Lemma 3 *If for a given f^i the earliest possible time of arrival at customer $i+1$ is smaller than l^j_{i+1}, then $f^i \oplus (j)$ dominates $f^i \oplus (j')$ for all $j' \in \{j+1, \ldots, \theta_{i+1}\}$.*

Proof of Lemma 3. The time of arrival $\tilde{a}_i^{f^i} + s_i + t_{i,i+1}$ is shifted to e^j_{i+1} resp. $e^{j'}_{i+1}$ (see Figure 8b). Violations for $f^i \oplus (j)$ and $f^i \oplus (j')$ are identical. Because $\tilde{a}_{i+1}^{f^i \oplus (j)}$ is smaller than $\tilde{a}_{i+1}^{f^i \oplus (j')}$, the validity of Equations (26) and (27) is trivial. Applying Lemma 1 concludes the proof. □

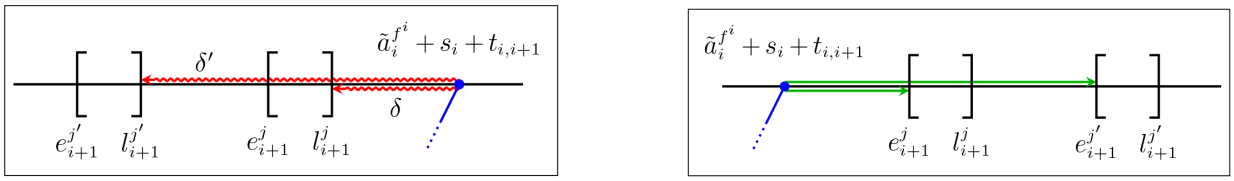
We combine Lemmata 2 and 3 to obtain Corollaries 2 and 3.

Corollary 2 *If the earliest possible time of arrival at customer $i+1$ with respect to f^i lies within a time window $\left[e^j_{i+1}, l^j_{i+1}\right]$, then $f^i \oplus (j)$ dominates $f^i \oplus (j')$ for all $j' \in \{1, \ldots, \theta_{i+1}\} \setminus \{j\}$.*

Corollary 3 *If the earliest possible time of arrival at customer $i+1$ with respect to f^i lies in between two time windows $\left[e^j_{i+1}, l^j_{i+1}\right]$ and $\left[e^{j+1}_{i+1}, l^{j+1}_{i+1}\right]$, then $f^i \oplus (j)$ dominates $f^i \oplus (j')$ for all $j' \in \{1, \ldots, j-1\}$, and $f^i \oplus (j+1)$ dominates $f^i \oplus (j'')$ for all $j'' \in \{j+1, \ldots, \theta_{i+1}\}$.*

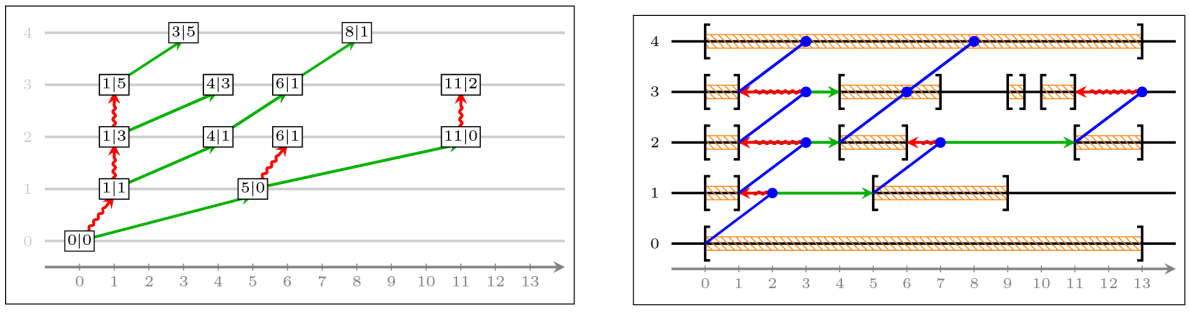
With these dominance rules, we can significantly reduce the search tree as illustrated in Example 2.

Example 2 (Example 1 revisited) *Revisiting Example 1, we can now cut dominated branches leaving us with the search tree depicted in Figure 9. This reduces the size of the example search tree from 57 (see Figure 4) to 13 nodes.*



(a) (b)

Figure 8: Violation and waiting time for two different time windows



(a) Dominance-free search tree (b) Search tree in the instances

Figure 9: The dominance-free search tree and its visualization in the route

3.3.3 Dynamic programming routine

The dominance criterion from Section 3.3.2 allows to design an efficient dynamic program that determines an optimal time window assignment for a given route $r = \langle v_0, \dots, v_k \rangle$ with time windows $[e_i^p, l_i^p], p \in \{1, \dots, \theta_i\}$. Figure 10 shows the pseudocode overview of our dynamic program while we detail pseudocodes for additional subroutines in Appendix C. After initializing the root level L_0 with the depot time window with a violation of 0, we propagate all nodes from the previous level L_{i-1} to the new level L_i by propagating the respective arrival times (l. 5). If the new arrival time is larger than the last time window's closing time, we create a new label for the closing time of the affected time window (l. 6–9). Otherwise, the time of arrival is either enclosed in a time window and we create a new label for the time of arrival (l. 12–15), or there is a subsequent time window for which we create a new label for its opening time window (l. 17–19). If there is also a previous time window, we create a label for the closing time of this time window as well (l. 21–23). For determining the subsequent or the enclosing time window, we use the algorithm depicted in Figure 16. We insert every non-dominated label into the list of labels for the considered customer using a constant time procedure as detailed in Figure 17 (Appendix C). For the heuristic variant of the algorithm, we only consider the l^{path} elements in L_{i-1} with the smallest time window violation.

3.3.4 Computational complexity

In this section, we prove that the worst-case complexity of our optimal time window sequencing routine is in $\mathcal{O}(\theta_{max} \log(\theta_{max}) \cdot k^2)$. We prove this complexity step-wise. First, we derive a bound on the size of our search tree, before we discuss the complexity of propagating a single node in the search tree and derive the overall complexity for determining an optimal time window assignment.

Size of the search Tree: In the creation of L_i of the search tree, we apply the dominance criterion from Section 3.3.2 such that L_i only contains non-dominated nodes. Then, Lemma 4 denotes the maximum size of L_i.

1: $L_0 \leftarrow [(e_0^0, (0), 0)]$
2: **for** $i := 1$ to k **do**
3: $\quad L_i := [\,]$
4: $\quad$ **for** $\left(\tilde{a}_{i-1}^{f^{i-1}}, f^{i-1}, \lambda_{i-1}\right) \in L_{i-1}$ **do**
5: $\quad\quad a_i := \tilde{a}_{i-1}^{f^{i-1}} + s_{i-1} + t_{i-1,i}$
6: $\quad\quad$ **if** $a_i \geq l_i^{\theta_i}$ **then**
7: $\quad\quad\quad f^i := f^{i-1} \oplus (\theta_i)$
8: $\quad\quad\quad \lambda_i := \lambda_{i-1} + a_i - l_i^{\theta_i}$
9: $\quad\quad\quad smartAppend\left(L_i, (l_i^{\theta_i}, f^i, \lambda_i)\right)$
10: $\quad\quad$ **else**
11: $\quad\quad\quad [e_i^p, l_i^p] := findSubsequentOrEnclosingTimeWindow\left(a_i, \left\{[e_i^1, l_i^1], \dots, \left[e_i^{\theta_i}, l_i^{\theta_i}\right]\right\}\right)$
12: $\quad\quad\quad$ **if** $a_i \in [e_i^p, l_i^p]$ **then**
13: $\quad\quad\quad\quad f^i := f^{i-1} \oplus (\theta)$
14: $\quad\quad\quad\quad \lambda_i := \lambda_{i-1}$
15: $\quad\quad\quad\quad smartAppend\left(L_i, (a_i, f^i, \lambda_i)\right)$
16: $\quad\quad\quad$ **else**
17: $\quad\quad\quad\quad f_1^i := f^{i-1} \oplus (p)$
18: $\quad\quad\quad\quad \lambda_i^1 := \lambda_{i-1}$
19: $\quad\quad\quad\quad smartAppend\left(L_i, \left(e_i^p, f_1^i, \lambda_i^1\right)\right)$
20: $\quad\quad\quad\quad$ **if** $p - 1 \geq 0$ **then**
21: $\quad\quad\quad\quad\quad f_2^i := f^{i-1} \oplus (p-1)$
22: $\quad\quad\quad\quad\quad \lambda_i^2 := \lambda_{i-1} + a_i - l_i^{p-1}$
23: $\quad\quad\quad\quad\quad smartAppend\left(L_i, \left(l_i^{p-1}, f_2^i, \lambda_i^2\right)\right)$
24: Find $(\tilde{a}^*, f^*, \lambda^*) := \underset{(\tilde{a}, f, \lambda) \in L_n}{argmin} \lambda$

Figure 10: Optimal time window assignment

Lemma 4 *The size of L_i is bounded by*

$$|L_i| \leq \min\left\{2\,|L_{i-1}|\,, |L_{i-1}| + \theta_i - 1\right\} \leq 1 + i \cdot (\theta_{max} - 1)\,. \tag{34}$$

Proof of Lemma 4. Corollaries 2 and 3 imply that every node in L_{i-1} yields either zero, one or two nodes in L_i depending on whether the earliest possible time of arrival is in a time window or in between two time windows and on whether the resulting paths are dominated so that

$$|L_i| \leq 2\,|L_{i-1}|\,. \tag{35}$$

All earliest possible times of arrival at customer i that are in between the same two time windows $\left[e_i^j, l_i^j\right]$ and $\left[e_i^{j+1}, l_i^{j+1}\right]$ are corrected to either l_i^j or e_i^{j+1}. According to Corollary 1, only two of these arrival times (l_i^j, e_i^{j+1}) are non-dominated. Hence, at most $\theta_i - 1$ nodes in L_{i-1} (one for each gap between two time windows) account for two nodes in L_i such that

$$|L_i| \leq |L_{i-1}| + \theta_i - 1. \tag{36}$$

Combining Equations (35) and (36) yields the first inequality in Equation (34). We obtain the second inequality by utilizing $\theta_i \leq \theta_{max}$ and inductively applying Equation (36) i times. □

With Lemma 5, we derive a bound on the total number of nodes in the search tree.

Lemma 5 *The number of nodes in the search tree does not exceed* $(\theta_{max} - 1)\frac{k(k+1)}{2} + k + 1$.

Proof of Lemma 5. The number of nodes in the search tree is the cumulated number of nodes per level, i.e., $\sum_{i=0}^{k} |L_i|$. Applying Lemma 4 with $|L_0| = 1$ yields

$$\sum_{i=0}^{k} |L_i| = |L_0| + \sum_{i=1}^{k} |L_i| \leq |L_0| + \sum_{i=1}^{k} \left(|L_0| + i \cdot (\theta_{max} - 1)\right). \tag{37}$$

$$\sum_{i=0}^{k} |L_i| \leq (k+1) \cdot |L_0| + (\theta_{max} - 1) \sum_{i=1}^{k} i = k + 1 + (\theta_{max} - 1)\frac{k(k+1)}{2}. \tag{38}$$

This concludes the proof. □

Node propagation: To keep this paper concise, we provide pseudocodes for the following discussion in Appendix C. In general, the complexity for propagating a node depends on the complexity of i) identifying non-dominated branches and of ii) checking whether these branches dominate other branches with a different parent node. Given Corollaries 2 and 3, we can use a binary search (see Figure 16) to find the time window(s) that enclose a node's earliest time of arrival such that the complexity of identifying non-dominated branches results to $\mathcal{O}(\log(\theta_{\max}))$. Applying a dominance check is possible in constant time (see Figure 17) such that the overall complexity of propagating nodes remains in $\mathcal{O}(\log(\theta_{\max}))$.

Overall complexity: Synthesizing Lemma 5 with the node propagation complexity yields Lemma 6 which denotes the overall complexity of our optimal time window sequencing.

Lemma 6 *An optimal time window assignment can be found in* $\mathcal{O}(\theta_{max} \log(\theta_{max}) \cdot k^2)$.

Proof of Lemma 6. According to Lemma 5 the number of nodes in the search tree is bounded by

$$(\theta_{max} - 1)\frac{k(k+1)}{2} + k + 1 \in \mathcal{O}(\theta_{max} \cdot k^2). \tag{39}$$

Recall that the complexity to evaluate a single node is in $\mathcal{O}(\log(\theta_{max}))$. By multiplying the total number of nodes with the runtime for evaluating a single node, we obtain a total runtime of

$$\log(\theta_{max}) \cdot \left((\theta_{max}-1)^{\frac{k(k+1)}{2}} + k + 1\right) \in \mathcal{O}(\theta_{max} \log(\theta_{max}) \cdot k^2). \tag{40}$$

This concludes the proof. □

3.4 Time window violation update

Analogously to the logic of backward violation terms in the single time window case (cf. Nagata et al. 2010), we introduce backward violation terms for the multiple time window case to speed up the evaluation of search moves. We introduce a latest possible start time of the service $z_j^{\hat{f}^i}$ at customer j such that the backward violation terms result as follows:

$$z_n^{\hat{f}^i} = \tilde{z}_n^{\hat{f}^i} = l_n^{\hat{f}^i_{\theta_n}} \tag{41}$$

$$z_j^{\hat{f}^i} = \tilde{z}_{j+1}^{\hat{f}^i} - s_j - t_{j,j+1} \qquad i \le j \le n \tag{42}$$

$$\tilde{z}_j^{\hat{f}^i} = \begin{cases} \min\left\{z_j^{\hat{f}^i}, l_j^{\hat{f}^i_j}\right\} & \text{if } z_j^{\hat{f}^i} \ge e_j^{\hat{f}^i_j} \\ e_j^{\hat{f}^i_j} & \text{otherwise} \end{cases} \qquad i+1 \le j \le n. \tag{43}$$

With this notation, the *backward time window violation* on r with respect to $\hat{f}^i$ results to

$$\overleftarrow{\lambda}_i^{\text{tw}}(r, \hat{f}^i) = \sum_{j=i+1}^{k} \max\left\{e_j^{\hat{f}^i_j} - z_j^{\hat{f}^i}, 0\right\}. \tag{44}$$

With (23) and (44) we can calculate the overall time window violation for a route $r = \langle 0, \dots, v_i, v_{i+1}, \dots, n\rangle$ by

$$TW(r, f^i \oplus \hat{f}^i) = \overrightarrow{\lambda}_i^{\text{tw}}(r_1, f^i) + \overleftarrow{\lambda}_i^{\text{tw}}(r_2, \hat{f}^i) + \max\left\{\tilde{a}_i^{f^i} - z_i^{\hat{f}^i}, 0\right\}, \tag{45}$$

when r is generated from concatenating two partial routes $r_1 = \langle 0, \dots, v_i\rangle$ and $r_2 = \langle v_{i+1}, \dots, n\rangle$ with a fixed time window assignment f^i (for r_1) and a fixed time window assignment remainder $\hat{f}^i$ (for r_2) .

Similar to the forward notation, we introduce a backwards search tree with $\hat{L}_i$ being its i-th level such that Equation 45 generalizes to

$$TW(r) = \min_{\substack{(\tilde{a}_i^{f^i}, f^i, \overrightarrow{\lambda}_i^{\text{tw}}) \in L_i \\ (\tilde{z}_i^{\hat{f}^i}, \hat{f}^i, \overleftarrow{\lambda}_i^{\text{tw}}) \in \hat{L}_i}} \overrightarrow{\lambda}_i^{\text{tw}} + \overleftarrow{\lambda}_i^{\text{tw}} + \max\left\{\tilde{a}_i^{f^i} - z_i^{\hat{f}^i}, 0\right\}. \tag{46}$$

Note that all theoretical concepts and results developed for the forward evaluation hold for the backward evaluation as well because both can be transformed into each other by defining a reversed instance (see Appendix A).

According to Lemma 4 we have up to $1+i\cdot(\theta_{max}-1)$ non-dominated time window assignments at level L_i as well as $1+(k-i)\cdot(\theta_{max}-1)$ non-dominated time window assignment remainders in $\hat{L}_i$. Hence, a trivial algorithm that solves (46) by updating the time window violation via pairwise comparison yields a complexity of $\mathcal{O}(k^2\theta_{max}^2)$ depending on the number of customers k on route r. To allow for more efficient search move evaluations, we prove that $\mathcal{O}(k\theta_{max})$ comparisons are sufficient to evaluate a concatenation correctly.

Lemma 7 *Let f^i be a non-dominated time window assignment, and let $\hat{f}^i$, $\hat{g}^i$ be two non-dominated time window assignment remainders. In this case, if Equation (47) holds, Equation (48) follows.*

$$\tilde{z}_{i+1}^{\hat{g}^i} - s_i - t_{i,i+1} \le \tilde{z}_{i+1}^{\hat{f}^i} - s_i - t_{i,i+1} \le \tilde{a}_i^{f^i} \tag{47}$$

$$TW(r, f^i \oplus \hat{f}^i) < TW(r, f^i \oplus \hat{g}^i) \tag{48}$$

Proof of Lemma 7. With (47) we simplify Equation (45)

$$TW(r, f^i \oplus \hat{f}^i) = \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) + \overleftarrow{\lambda}_i^{\text{tw}}(r, \hat{f}^i) + \max\left\{0, \tilde{a}_i^{f^i} - \left(\tilde{z}_{i+1}^{\hat{f}^i} - s_i - t_{i,i+1}\right)\right\} \tag{49}$$

to

$$= \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) + \overleftarrow{\lambda}_i^{\text{tw}}(r, \hat{f}^i) + \tilde{a}_i^{f^i} - \tilde{z}_{i+1}^{\hat{f}^i} + s_i + t_{i,i+1}$$

Because $\hat{g}^i$ does not dominate $\hat{f}^i$ and $\tilde{z}_{i+1}^{\hat{f}^i} \geq \tilde{z}_{i+1}^{\hat{g}^i}$, we estimate

$$(49) < \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) + \left(\overleftarrow{\lambda}_i^{\text{tw}}(r, \hat{g}^i) + \tilde{z}_{i+1}^{\hat{f}^i} - \tilde{z}_{i+1}^{\hat{g}^i}\right) + \tilde{a}_i^{f^i} - \tilde{z}_{i+1}^{\hat{f}^i} + s_i + t_{i,i+1} \tag{50}$$

$$= \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) + \overleftarrow{\lambda}_i^{\text{tw}}(r, \hat{g}^i) + \tilde{a}_i^{f^i} - \left(\tilde{z}_{i+1}^{\hat{g}^i} - s_i - t_{i,i+1}\right) \tag{51}$$

from Equation (55) (see Appendix A). Because Equation (51) equals $TW(r, f^i \oplus \hat{g}^i)$ this concludes the proof:

$$= \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) + \overleftarrow{\lambda}_i^{\text{tw}}(r, \hat{g}^i) + \max\left\{0, \tilde{a}_i^{f^i} - \left(\tilde{z}_{i+1}^{\hat{g}^i} - s_i - t_{i,i+1}\right)\right\}$$
$$= TW(r, f^i \oplus \hat{g}^i).$$

□

Lemma 8 *Let f^i be a non-dominated time window assignment, and let $\hat{f}^i$, $\hat{g}^i$ be two non-dominated time window assignment remainders . In this case, if Equation (52) holds, (53) follows.*

$$\tilde{a}_i^{f^i} \leq \tilde{z}_{i+1}^{\hat{f}^i} - s_i - t_{i,i+1} \leq \tilde{z}_{i+1}^{\hat{g}^i} - s_i - t_{i,i+1}, \tag{52}$$

$$TW(r, f^i \oplus \hat{f}^i) \leq TW(r, f^i \oplus \hat{g}^i) \tag{53}$$

Proof of Lemma 8. In analogy to Lemma 7, see Appendix B. □

Lemmata 7 and 8 imply that for each partial time window assignment in L_i only the two neighboring partial time window remainders in $\hat{L}_i$ must be considered to update the time window violation of a route and vice versa. In the worst case, the corrected earliest arrival times and the latest possible start times at customer i alternate such that $1 + i \cdot (\theta_{max} - 1) + 1 + (k - i) \cdot (\theta_{max} - 1) \in \mathcal{O}(k\theta_{max})$ comparisons are necessary. Figure 11 shows a pseudo code of this evaluation scheme.

Note that these results also hold for search operators which insert a vertex instead of concatenating two partial routes because we can replace each vertex insertion by combining a forward propagation and a partial route concatenation (see Appendix E, Table 6).

Input: $L_i = [f_0^i, \ldots, f_l^i]$ with $\tilde{a}_i^{f_0^i} < \cdots < \tilde{a}_i^{f_l^i}$ and level $\hat{L}_i = \left[\hat{f}_0^i, \ldots, \hat{f}_{\hat{l}}^i\right]$ with $z_i^{\hat{f}_0^i} < \cdots < z_i^{\hat{f}_{\hat{l}}^i}$
Initialization:
$g \coloneqq f_0^i$, $\hat{g} \coloneqq \hat{f}_0^i$, $\lambda^* \coloneqq \infty$
Iteration:
while true **do**
  $\lambda \coloneqq \overrightarrow{\lambda}_i^{\text{tw}}(r, g) + \overleftarrow{\lambda}_i^{\text{tw}}(r, \hat{g}) + \max\left\{0, \tilde{a}_i^g - \left(\tilde{z}_{i+1}^{\hat{g}} - s_i - t_{i,i+1}\right)\right\}$
  if $\lambda < \lambda^*$ **then**
    $\lambda^* \coloneqq \lambda$
  if $\tilde{a}_i^g == z_i^{\hat{g}}$ **then**
    $g \coloneqq successor(g)$, $\hat{g} \coloneqq successor(\hat{g})$
  else if $\tilde{a}_i^g < z_i^{\hat{g}}$ **then**
    $g \coloneqq successor(g)$
  else
    $\hat{g} \coloneqq successor(\hat{g})$
  if $g == f_l^i$ or $\hat{g} == \hat{f}_{\hat{l}}^i$ **then**
    break
Output: λ^*

Figure 11: Linear time window update

4 Results

We implemented the LNS as a single thread code in C++ and ran all experiments on a desktop computer with an Intel Core i7 3.7 GHz and 32 GB RAM, running Ubuntu 16.04. For future research, we provide all benchmark instances including solutions at https://rwth-aachen.sciebo.de/s/KzIKxP5nmxtzMDc. Because the single components of our algorithm depend on various parameters, we first describe our parameter fitting in Section 4.1. Section 4.2 details computational studies to evaluate the performance of our algorithm. Finally, we present new benchmark instances that reflect planning tasks in user-centered last mile logistics in Section 4.3. Based on these, we discuss results that show the benefit of our algorithm and allow to gain managerial insights on strategies to offer multiple time windows in practice.

4.1 Parameter fitting

We base our parameter fitting on the method described in Ropke and Pisinger (2006). Starting with a parameter setting that we found while developing the algorithm, we vary a single parameter while leaving all other parameters fixed. We evaluate each setting based on its solution quality according to the best objective value out of 20 runs on a set of tuning instances (cm101, cm201, rm101, rm201, rcm101, rcm201) from the instance set of Belhaiza et al. (2014). After fixing a parameter to its best performing setting, we proceed with the next one until all parameters are fixed. Table 2 details our final parameter setting for the minimum and maximum percentage of removed customers in destroy operators ($\Gamma_{\min}, \Gamma_{\max}$), the number of paths considered in the heuristic propagation of the time window violation (l^{path}), the iterations (relative to $i^{\max}$) after which the time window violation propagation method swaps between optimal and heuristic evaluations (I_{switch}), the weight of the spatial, freight, time window, and route relatedness component in R_{ij} ($\beta_S, \beta_F, \beta_T, \beta_R$), the number of closest relatives to be considered in a removal step (l^{relative}), the number of insertion positions considered in a repair step (l^{insert}), the initial value and the lower and upper bound for penalty weights($\alpha^{\text{fr}}_{\min}, \alpha^{\text{fr}}_{\text{start}}, \alpha^{\text{fr}}_{\max}, \alpha^{\text{tw}}_{\min}, \alpha^{\text{tw}}_{\text{start}}, \alpha^{\text{tw}}_{\max}$), the penalty update factor (γ), the number of local search iterations after which penalty weights are updated (i_{pLS}), the maximum number of iterations without improvement ($i^{\max}_{\text{noi}}$), the number of iterations before the current solution is set back to the best known solution (i^{r}), the deviation factor for the local search corridor (δ), the percentage of customers contained in the descending neighborhoods ($\eta^{\text{c}}, \eta^{\text{m}}$), and the maximum number of iterations ($i^{\max}$). We highlight the final setting in bold and state for each parameter the deviation $\overline{\Delta\lambda}$ between the best and the other settings.

4.2 Computational studies

We present results on 48 benchmark instances introduced by Belhaiza et al. (2014). These instances are based on the VRPTW instances from Solomon (1987). While customer locations, demands and service times

Table 2: Parameter setting for the LNS

$(\Gamma_{\min}, \Gamma_{\max})$	(0.1, 0.15)	**(0.1, 0.20)**	(0.1, 0.25)	γ	**1.05**	1.15	1.3
$\overline{\Delta\lambda}$	0.01	0	0.04	$\overline{\Delta\lambda}$	0	0.06	0.07
l^{path}	2	**3**	5	i_{pLS}	**1**	3	5
$\overline{\Delta\lambda}$	0.06	0	0.04	$\overline{\Delta\lambda}$	0	0.02	0.04
I_{switch}	[0.5]	**[0.3, 0.5, 0.7]**	[0.4, 0.5, 0.9]	$i^{\max}_{\text{noi}}$	300	525	**750**
$\overline{\Delta\lambda}$	0	0.02	0.05	$\overline{\Delta\lambda}$	0.44	0.25	0
$(\beta_S, \beta_F, \beta_T, \beta_R)$	(6, 3, 5, 8)	**(6, 3, 5, 5)**	(8, 3, 6, 8)	i^{r}	**75**	150	225
$\overline{\Delta\lambda}$	0	0	0.07	$\overline{\Delta\lambda}$	0	0.19	0.35
l^{relative}	1	3	**5**	δ	1.15	1.3	**2**
$\overline{\Delta\lambda}$	0.11	0.23	0	$\overline{\Delta\lambda}$	0.07	0.03	0
l^{insert}	1	**3**	5	$(\eta^{\text{c}}, \eta^{\text{m}})$	(0.05, 0.2)	**(0.1, 0.3)**	(0.1, 0.5)
$\overline{\Delta\lambda}$	0.1	0	0.14	$\overline{\Delta\lambda}$	0.09	0	0.03
$\left(\alpha^{\text{fr/tw}}_{\min}, \alpha^{\text{fr/tw}}_{\text{start}}, \alpha^{\text{fr/tw}}_{\max}\right)$	(0.1, 10, 1000)	(0.5, 50, 5000)	**(0.1, 10^2, 10^5)**	$i^{\max}$	1000	2000	**3000**
$\overline{\Delta\lambda}$	0.23	0.31	0	$\overline{\Delta\lambda}$	0.27	0.06	0

The table shows the results on the tested parameter settings. The final setting used for numerical studies is printed in bold. We fitted the parameters in the order displayed in the table, beginning at its upper left.

as well as the vehicles' capacities remain as in their original counterparts, the instances have multiple time windows based on different distributions. Table 7 (see Appendix E) details these time window distributions. To compare our algorithm to the MIP from Section 2, we created additional small instances by taking the first 10, 15, 20 and 25 customers of the instances cm101, cm105, cm201, cm205, rm101, rm105, rm201, rm205, rcm101, rcm105, rcm201, and rcm205.

Table 3 compares our LNS to our MIP formulation which was run with Gurobi 8.1.0 on the same desktop computer. For the MIP, we state the objective value λ and its computational time t. We used a time limit of 7200 seconds such that instances that show a runtime of 7200 seconds might not be solved to optimality yet. For our LNS, we state the best λ^b and average λ^a objective value out of ten runs, the deviation between λ^b and the MIP solution Δ^b, as well as between λ^a and the MIP solution Δ^a, and the average runtime t^a. As can be seen, our LNS provides stable results, matching or improving the solution for all instances within runtimes of a few seconds, while the MIP already fails to solve even some instances with 15 customers to optimality.

Table 4 compares the results of our algorithm to the results of Belhaiza et al. (2014) and Belhaiza et al. (2017) on larger instances. We state for each instance its so far best known solution (BKS); and for each algorithm the number of vehicles in the best solution r^b, the solution value of the best (λ^b) and the average solution (λ^a) out of ten runs, the gap between the so far BKS and the best (Δ^b) or average (Δ^a) solution, and the average computational time t^a. As can be seen, our algorithms vastly outperforms the algorithms of Belhaiza et al. (2014) and Belhaiza et al. (2017) in terms of solution quality. We find new BKSs for 26 out of 48 instances. On average, we improve the solution quality over all instances by 1.5%. For some instances that show challenging time window configurations, we find improvements of 23.6% or 25.3%. For 5 instances, we do not only reduce the objective function value but also the number of necessary vehicles. However, the superiority in solution quality comes with a price in terms of computational time because the exact evaluation of time window assignments allows to find better solutions (especially on difficult instances) but bears an

Table 3: Comparison of MIP implementation and LNS

		MIP		LNS							MIP		LNS				
Instance	$\|\mathcal{C}\|$	λ	t	λ^b	Δ^b	λ^a	Δ^a	t^a	Instance	$\|\mathcal{C}\|$	λ	t	λ^b	Δ^b	λ^a	Δ^a	t^a
cm101	10	485.3	1345	485.3	0	485.3	0	< 1	cm101	20	825.9	7200	801.7	**-2.9**	801.7	**-2.9**	3
cm105	10	486.0	114	486.0	0	486.0	0	< 1	cm105	20	801.3	7200	801.1	**-0.3**	801.1	**-0.3**	2
cm201	10	833.1	< 1	833.1	0	833.1	0	< 1	cm201	20	1599.0	7200	1599.0	0	1599.0	0	2
cm205	10	834.9	< 1	834.9	0	834.9	0	< 1	cm205	20	1614.9	7200	948.7	**-41.3**	948.7	**-41.3**	2
rcm101	10	566.0	2	566.0	0	566.0	0	< 1	rcm101	20	883.7	168	883.7	0	883.7	0	3
rcm105	10	566.8	< 1	566.8	0	566.8	0	< 1	rcm105	20	885.3	376	885.2	0	885.2	0	2
rcm201	10	1137.8	6	1137.8	0	1137.8	0	< 1	rcm201	20	1219.4	66	1219.4	0	1219.4	0	2
rcm205	10	1137.8	4	1137.8	0	1137.8	0	< 1	rcm205	20	1219.4	1482	1219.4	0	1219.4	0	1
rm101	10	598.2	63	598.2	0	598.2	0	< 1	rm101	20	922.4	7200	922.4	0	922.8	0	3
rm105	10	595.4	161	595.4	0	597.4	0	< 1	rm105	20	906.3	7200	906.2	0	908.2	0.2	1
rm201	10	1173.0	< 1	1173.0	0	1173.0	0	< 1	rm201	20	1262.3	5	1262.3	0	1262.3	0	3
rm205	10	1173.0	< 1	1173.0	0	1173.0	0	< 1	rm205	20	1262.3	11	1262.3	0	1262.3	0	1
cm101	15	537.8	38	537.8	0	537.8	0	1	cm101	25	1054.9	7200	825.2	**-21.8**	825.2	**-21.8**	5
cm105	15	538.1	70	538.1	0	538.1	0	< 1	cm105	25	1026.6	7200	823.6	**-19.8**	823.6	**-19.8**	3
cm201	15	870.6	8	870.6	0	870.6	0	< 1	cm201	25	1626.9	7200	1626.9	0	1626.9	0	4
cm205	15	883.2	41	883.2	0	883.2	0	< 1	cm205	25	1619.7	7200	1619.7	0	1619.7	0	3
rcm101	15	586.9	5	586.9	0	586.9	0	< 1	rcm101	25	895.0	725	895.0	0	895.0	0	4
rcm105	15	588.5	4	588.5	0	588.5	0	< 1	rcm105	25	898.4	1246	898.3	0	898.3	0	3
rcm201	15	1153.6	641	1153.6	0	1153.6	0	< 1	rcm201	25	1226.1	7200	1226.1	0	1226.1	0	3
rcm205	15	1153.6	1094	1153.6	0	1153.6	0	< 1	rcm205	25	1226.1	7200	1226.1	0	1226.1	0	3
rm101	15	878.2	7200	878.2	0	878.2	0	< 1	rm101	25	1185.8	7200	1185.8	0	1185.8	0	4
rm105	15	661.5	7200	661.5	0	661.5	0	< 1	rm105	25	960.1	7200	960.1	0	960.1	0	4
rm201	15	1229.2	< 1	1229.2	0	1229.2	0	< 1	rm201	25	1316.9	14	1316.9	0	1316.9	0	4
rm205	15	1229.2	< 1	1229.2	0	1229.2	0	< 1	rm205	25	1313.3	19	1313.3	0	1313.3	0	2

Abbreviations hold as follows: $|\mathcal{C}|$ - number of customers considered, λ - objective value found by MIP, t - runtime unsing MIP in seconds, λ^b - best objective value out of ten runs, Δ^b - gap between solution found using MIP and λ^b in percent, λ^a - average objective value out of ten runs, Δ^a - gap between solution found using MIP and λ^a in percent,t^a - average runtime out of ten runs in seconds.

Table 4: Results on benchmark instances RM1, RM2, CM1, CM2, RCM1 and RCM2

		Belhaiza et al. (2014)					Belhaiza et al. (2017)			**Schaap et al.**							
Instance	BKS	λ^b	r^b	t^b	λ^a	Δ^a	λ^b	r^b	t^b	λ^b	r^b	Δ^b	t^b	λ^a	r^a	Δ^a	t^a
cm101	3089.2	3089.2	10	102	3102.4	0.4	3101.2	10	62	3126.7	10	1.2	327	3172.6	10.2	2.7	259
cm102	3339.7	3426.9	12	97	3426.9	2.6	3339.7	11	86	3467.8	12	3.8	307	3481.7	12.0	4.3	309
cm103	3520.5	3532.7	12	91	3572.7	1.5	3520.5	12	99	3436.8	11	**-2.4**	260	3471.9	11.0	**-1.4**	257
cm104	4048.0	4051.3	14	69	4058.0	0.2	4048	14	79	3870.8	13	**-4.4**	209	3893.5	13.0	**-3.8**	215
cm105	3010.2	3060.6	11	67	3077.3	2.2	3010.2	10	62	3018.8	10	0.3	213	3061.7	10.0	1.7	191
cm106	2982.2	2992.4	10	65	3020.2	1.3	2982.2	10	69	2982.9	10	0.0	193	2999.8	10.0	0.6	172
cm107	3256.5	3256.5	11	38	3292.3	1.1	3256.5	11	63	3077.9	10	**-5.5**	102	3080.9	10.0	**-5.4**	107
cm108	2967.3	2968.7	10	32	2973.1	0.2	2967.3	10	70	2965.6	10	**-0.1**	65	2968.6	10.0	0.0	107
cm201	4390.3	4436.6	5	92	4452.5	1.4	4390.3	5	96	4392.2	5	0.0	276	4419.6	5.0	0.7	260
cm202	4990.8	4998.8	6	90	5024.9	0.7	4990.8	6	73	4996.5	6	0.1	185	5005.6	6.0	0.3	194
cm203	4445.8	4445.8	5	94	4484.6	0.9	4446.7	5	64	4445.5	5	**0.0**	230	4463.8	5.0	0.4	219
cm204	4332.2	4335.2	5	92	4372.4	0.9	4332.2	5	79	4321.5	5	**-0.2**	224	4335.7	5.0	0.1	220
cm205	3819.1	3863.5	4	99	3883.2	1.7	3819.1	4	99	3818.7	4	**0.0**	240	3838.4	4.0	0.5	241
cm206	3709.4	3722	4	84	3743.2	0.9	3709.4	4	97	3699.0	4	**-0.3**	221	3713.1	4.0	0.1	192
cm207	3933.1	3968.4	4	60	3977.8	1.1	3933.1	4	84	3921.9	4	**-0.3**	179	3941.7	4.0	0.2	186
cm208	3730.5	3771.1	4	65	3793.2	1.7	3730.5	4	98	3716.1	4	**-0.4**	209	3738.3	4.0	0.2	214
rcm101	3062.0	3062	10	77	3086.6	0.8	3063.5	10	94	3067.7	10	0.2	137	3070.8	10.0	0.3	152
rcm102	3127.7	3132.2	10	60	3142.3	0.5	3127.7	10	94	3110.8	10	**-0.5**	150	3157.2	10.1	0.9	130
rcm103	3131.8	3152.9	10	75	3163.8	1.0	3131.8	10	62	3124.9	10	**-0.2**	138	3138.9	10.0	0.2	129
rcm104	3111.8	3119.6	10	64	3134.6	0.7	3111.8	10	60	3111.2	10	**0.0**	135	3124.1	10.0	0.4	129
rcm105	3185.9	3187.9	10	65	3210.7	0.8	3185.9	10	84	3172.8	10	**-0.4**	94	3203.8	10.1	0.6	99
rcm106	3193.4	3218.9	10	55	3218.9	0.8	3193.4	10	96	3173.1	10	**-0.6**	118	3191.4	10.0	**-0.1**	116
rcm107	3487.7	3488.9	11	35	3514.0	0.8	3487.7	11	77	3487.7	11	**0.0**	44	3497.3	11.0	0.3	67
rcm108	3532.6	3592.7	11	36	3592.7	1.7	3532.6	11	82	3532.5	11	**0.0**	75	3544.7	11.0	0.3	71
rcm201	2778.7	2804	2	69	2827.8	1.8	2778.7	2	88	2701.7	2	**-2.8**	367	2736.4	2.0	**-1.5**	313
rcm202	2815.9	2836.9	2	79	2846.8	1.1	2815.9	2	61	2722.3	2	**-3.3**	241	2749.1	2.0	**-2.4**	294
rcm203	2721.9	2721.9	2	92	2725.4	0.1	2722	2	60	2707.3	2	**-0.5**	201	2736.2	2.0	0.5	271
rcm204	2698.4	2726.5	2	76	2743.1	1.7	2698.4	2	78	2696.5	2	**-0.1**	191	2703.9	2.0	0.2	244
rcm205	2754.5	2754.5	2	72	2775.7	0.8	2754.5	2	76	2721.2	2	**-1.2**	241	2733.3	2.0	**-0.8**	204
rcm206	2769.6	2812.7	2	22	2830.6	2.2	2769.6	2	93	2725.9	2	**-1.6**	200	2764.1	2.0	**-0.2**	211
rcm207	3749.8	3749.8	3	68	3786.8	1.0	3749.8	3	90	2863.4	2	**-23.6**	127	3585.4	2.8	**-4.4**	32
rcm208	2742.7	2791.4	2	21	2817.2	2.7	2742.7	2	72	2722.7	2	**-0.7**	148	2755.1	2.0	0.5	176
rm101	2970.2	2977.2	10	66	3005.0	1.2	2970.2	10	81	2977.2	10	0.2	157	2991.9	10.0	0.7	154
rm102	2725.3	2759.4	9	58	2759.4	1.3	2725.3	9	90	2730.0	9	0.2	139	2897.9	9.8	6.3	29
rm103	2681.5	2692.5	9	47	2710.5	1.1	2681.5	9	97	2693.4	9	0.4	153	2702.8	9.0	0.8	143
rm104	2691.8	2696.6	9	74	2719.2	1.0	2691.8	9	83	2697.6	9	0.2	135	2701.4	9.0	0.4	129
rm105	2687.2	2688.8	9	84	2711.0	0.9	2687.2	9	87	2689.7	9	0.1	124	2699.9	9.0	0.5	130
rm106	2691.9	2691.9	9	75	2691.9	0.0	2700.9	9	95	2707.1	9	0.6	124	2717.5	9.0	1.0	118
rm107	2685.1	2690.8	9	44	2714.7	1.1	2685.1	9	92	2686.0	9	0.0	122	2700.4	9.0	0.6	117
rm108	2716.0	2729.1	9	53	2729.1	0.5	2716	9	91	2727.6	9	0.4	110	2774.8	9.2	2.2	89
rm201	3686.4	3711.4	3	43	3720.5	0.9	3686.4	3	71	2753.3	2	**-25.3**	385	2780.6	2.0	**-24.6**	357
rm202	2681.0	2698.1	2	40	2717.3	1.4	2681	2	79	2691.3	2	0.4	361	2703.6	2.0	0.8	330
rm203	2673.6	2686.1	2	47	2702.0	1.1	2673.6	2	66	2684.4	2	0.4	228	2693.6	2.0	0.7	285
rm204	2664.9	2680.5	2	48	2691.2	1.0	2664.9	2	83	2678.4	2	0.5	251	2683.4	2.0	0.7	243
rm205	2651.3	2671	2	44	2688.2	1.4	2651.3	2	92	2671.0	2	0.7	163	2678.8	2.0	1.0	178
rm206	2672.8	2686.3	2	32	2704.9	1.2	2672.8	2	61	2688.3	2	0.6	192	2697.2	2.0	0.9	216
rm207	2657.3	2678.2	2	52	2696.2	1.5	2657.3	2	69	2682.4	2	0.9	215	2693.8	2.0	1.4	212
rm208	2663.6	2673.9	2	43	2690.7	1.0	2663.6	2	88	2678.6	2	0.6	154	2686.0	2.0	0.8	190
Tot. Avrg.	3179.8	3197.6	6.58	64	3215.1	1.1	3180.3	6.54	81	3132.0	6.46	**-1.5**	189	3168.4	6.50	**-0.36**	185

Abbreviations hold as follows: BKS - previous best known solution, r^b - number of vehicles in best solution out of ten runs, t^b - runtime to best solution in seconds, λ^b - best objective value out of ten runs, λ^a - average objective value out of ten runs, t^a - average runtime out of ten runs in seconds, r^a - average number of vehicles in ten runs, Δ^b - gap between the previous BKS and λ^b in percent, Δ^a - average gap between the average solution in Belhaiza et al. (2014) and λ^a in percent.

inherently higher computational complexity. Nevertheless, our computational times are still sufficient to allow for a day-ahead computation of routes in practice, which is a common application for the VRPMTW. Here, practitioners would certainly value a superior solution quality that gains improvements of up to 25% over a decreased computational time as long as computational times remain tractable for operations.

4.3 Managerial studies

Especially in e-commerce and grocery home deliveries, customer-driven logistics services are gaining importance. As a new service, customers can often choose one or multiple potential delivery time windows to receive their deliveries. In this section, we aim at deriving insights into the impact of different time window offering strategies, and whether or not a competitive algorithm helps to stabilize the operations for different customer demands and preferences. In the following, we create new instances (Section 4.3.1) to conduct these analyses and discuss their results in Section 4.3.2.

4.3.1 Experimental design

For our studies, we use the c100, r100, and rc100 VRPTW instances of Solomon (1987) as base instances because they cover different customer patterns. We consider a global service time window of eight hours in which customers must be served, and we choose a ten hour planning horizon for the depot. Within this general setting, we generate instances that reflect the following characteristics:

LSP strategies to offer time windows: In our basic scenario, the LSP offers only a single eight-hour time window (1x8h), i.e., the customer can only select the delivery day and has to be available the complete day. In all other scenarios, the LSP offers the customer to choose a specific time window or a selection of time windows with a shorter width. Specifically, we consider scenarios in which customers choose one out of two four-hour time windows (1x4h), two out of four two-hour time windows (2x2h), one out of four two-hour time windows (1x2h), four out of eight one-hour time windows (4x1h), three of eight one-hour (3x1h), two out of eight one-hour (2x1h), and one out of eight one hour (1x1h) time windows.

Customer time window selection behaviour: To account for different customer preferences, we create instances in which customers select time windows either *i*) uniformly distributed, or twice more likely *ii*) in the first 4 hours (morning delivery), *iii*) in the last 4 hours (afternoon delivery), *iv*) in the first or last quarter (early-or-late delivery), or *v)* after the first and before the last quarter (midday delivery).

The 1x8h scenario represents our baseline to assess the different time window offering strategies. It is not affected by the customer selection behavior because all customers have to remain available for deliveries the whole day. We consider a full factorial design in all influence factors such that we have 34 instance types for each customer pattern and 102 different instance types in total. To avoid a statistical bias, we create 20 instances for each instance type but the baseline, which is not affected by randomness. Thus, in total we obtain 1983 instances for our studies.

4.3.2 Results

In the following, we describe the results of the experiments described in 4.3.1. To keep this paper concise but secure scientific rigor, we provide extensive results that give numerical proof to our findings at `https://rwth-aachen.sciebo.de/s/KzIKxP5nmxtzMDc`.

Figures 12–14 detail the average cost increase for each time window offering strategy and customer time window selection behavior compared to our baseline scenario (1x8h) for spatially random (Figure 12), randomly clustered (Figure 13) and clustered (Figure 14) customer distributions. As can be seen, the cost increases vary significantly between 0.17% and 8.99% with respect to the time window offering strategy and the spatial customer distribution, but appear to be nearly robust on customer selection behaviors. From these results, we synthesize the following three managerial insights:

Spatially clustered customer distributions reduce the cost increase that result from multiple time window offering: As can be seen, for any time window offering strategy and customer time window selection behavior, the average cost increase for clustered customer distributions is up to 3.86% lower than for random spatial customer distributions. This shows that having a clustered customer set helps to reduce the cost increase of offering multiple time windows. In real-world applications, LSPs can aim at such a customer distribution by strategically increasing their market share in certain areas or by exchanging demands with competitors.

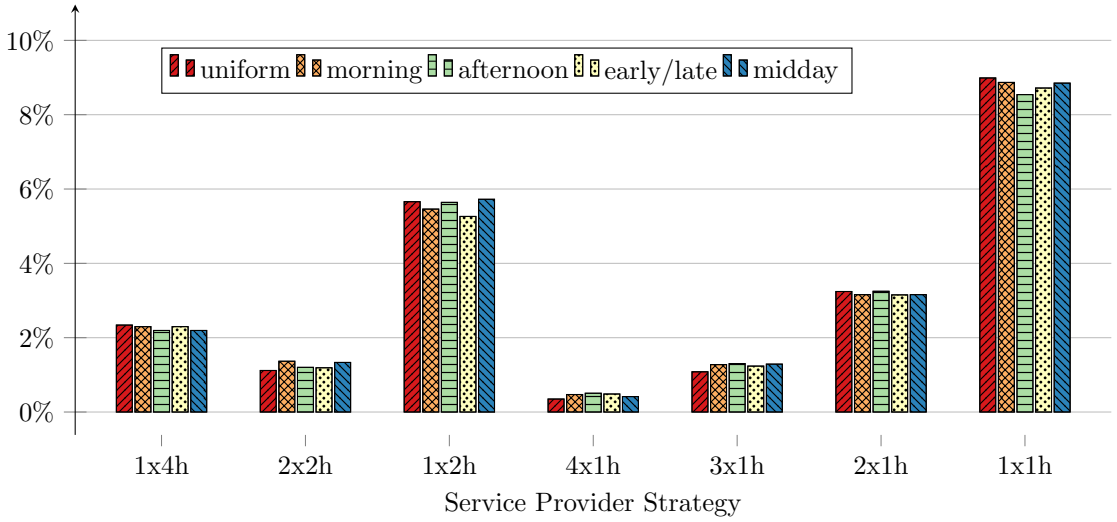


Figure 12: Average cost increase for different time window offering strategies and customer selection scenarios compared to the baselien scenario (1x8h) on random customer distributions

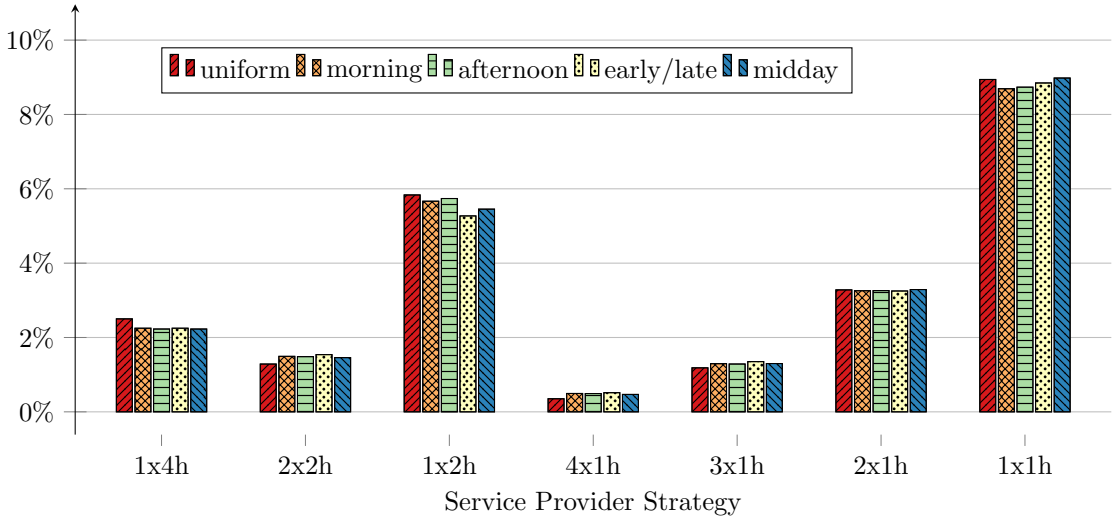


Figure 13: Average cost increase for different time window offering strategies and customer selection scenarios compared to the baselien scenario (1x8h) on randomly clustered customer distributions

The time window offering strategies show a robust hierarchy with respect to the customer distribution and the customer selection behavior: The customer selection strategy causes only minimal changes in the cost increase. Furthermore, for all spatial customer distributions, the largest cost increase results for the (1x1h) strategy and the relation between cost increases for other strategies remains consistent, too. Apparently, allowing the customer to choose a single time window limits the operational flexibility of the LSP the most and results in the highest cost increase. This cost increase is the lower, the wider the time window is.

Enforcing a customer preference selection of more than one time window helps to decrease additional costs significantly, independent of the time window width: On the contrary, enforcing multiple delivery time windows decreases the additional costs for the LSP significantly. Herein, offering a 2x2h strategy appears to be a good trade-off between cost increase and customer satisfaction. As customers can design two 2x2h time windows out of the 4x1h strategy, an LSP could even use a 4x1h strategy to reduce its costs even further.

Figure 15 exemplary shows a box-whisker-plot for the cost increase for each time window offering strategy and customer selection behavior for the instances with a random customer distribution. The characteristics for box-whisker plots for randomly clustered or clustered instances show similar characteristics and scale only in its amplitude. As can be seen, the single time window selection strategies which perform worst with respect to average cost increase also show the largest spread that may result in even higher cost increases for certain realizations. Contrary, the cost increase for the other strategies remains nearly stable with only a small spread in its realizations.

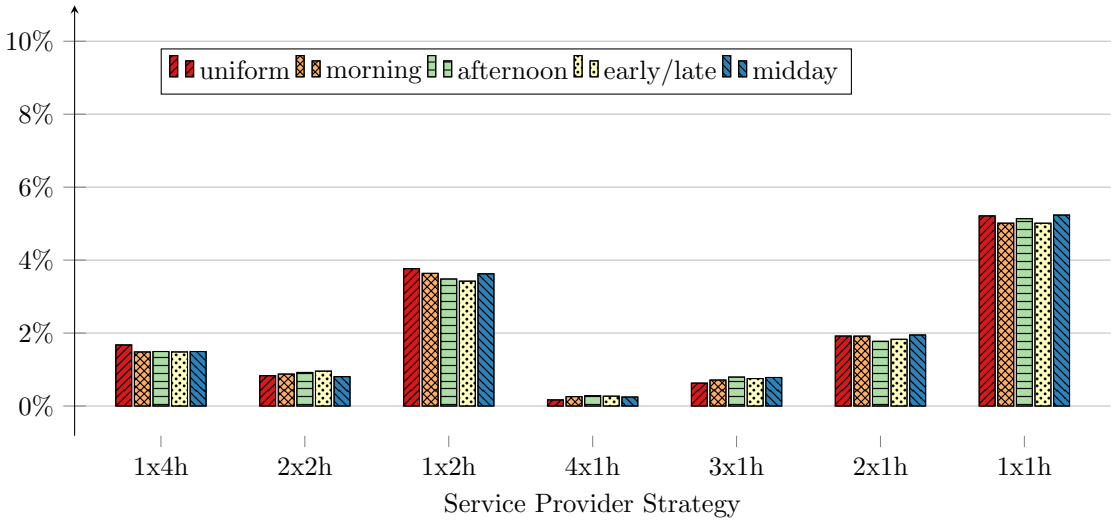


Figure 14: Average cost increase for different time window offering strategies and customer selection scenarios compared to the baselien scenario (1x8h) on clustered customer distributions

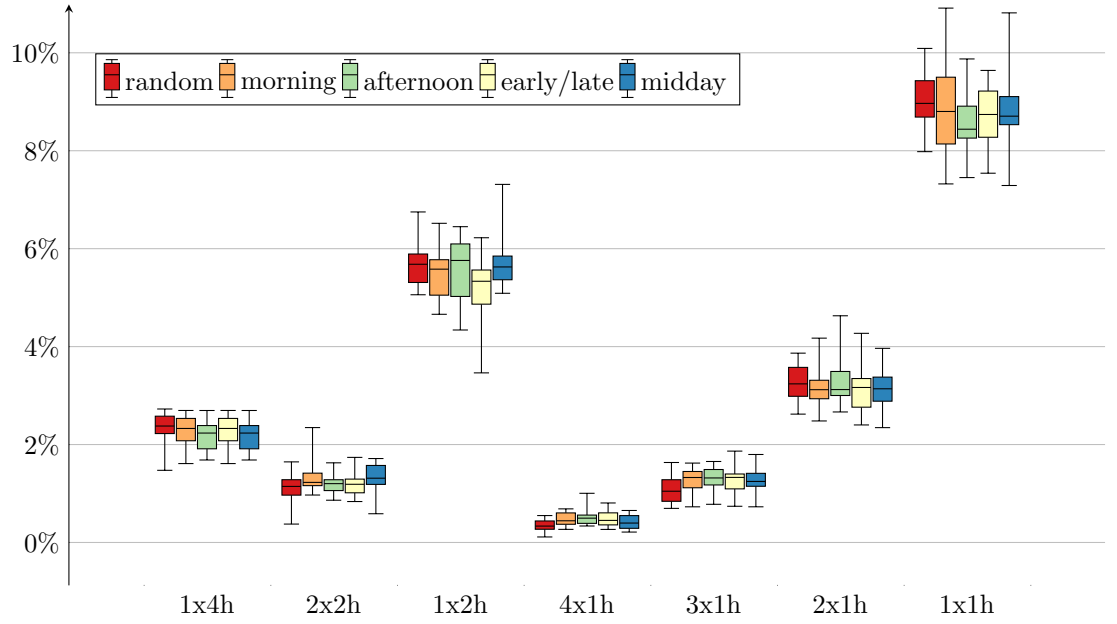


Figure 15: Box-whisker plots of increase of cost for R instances when compared to service provider strategy 1

5 Conclusion

In this paper, we presented a LNS based metaheuristic for the VRPMTW. Enhanced with a dynamic programming component that allows to efficiently assign delivery time windows to customers when evaluating a route, this algorithm presents a new state of the art in terms of solution quality for the VRPMTW. Besides a rigorous mathematical proof of this component and its computational complexity, we presented 26 new best known solutions for the instance set of Belhaiza et al. (2014), some improving the so far best known solutions up to 25.3%. Besides, we derived new benchmark sets to evaluate the impact of different time window offering strategies in the course of customer-centered logistics. We showed that time window offering strategies that leave a minimum choice between delivery time windows to the LSP may reduce the cost increase significantly while keeping the customer satisfaction at a high level.

Appendix A Graph transformation for backward-forward evaluation

In the following we describe how backward time window violations can be calculated by evaluating forward time window violations on a reversed route. This extends the theoretical results presented in Section 3 to the backward time window violation. For the sake of simplicity, we consider only a single time window and omit notational details from the multiple time window case.

The calculation of the backward time window violation of a route $r = \langle v_0, \dots, v_k \rangle$ coincides with the calculation of the forward time window violation of a reversed route r' that is defined as follows:

$$\begin{aligned}
\text{Route: } & r' = \langle v'_0, \dots, v'_k \rangle \coloneqq \langle v_k, \dots, v_0 \rangle \\
\text{Time windows: } & \left[e'_{v'_i}, l'_{v'_i}\right] \coloneqq \left[-l_{v_{k-i}}, -e_{v_{k-i}}\right] && \forall i = 0, \dots, k, j = 1, \dots, \theta_{v_i} \\
\text{Sevice time: } & s'_{v'_i} \coloneqq s_{v_{k-i-1}} && \forall i = 0, \dots, k-1 \\
& s'_{v'_k} \coloneqq s_{v_0}
\end{aligned}$$

This transformation inverts the route direction, mirrors its time windows, and shifts service times to the succeeding customer. For the forward violation calculation this yields

$$\begin{aligned}
& a_{v'_0} = \tilde{a}_{v'_0} = e'_{v'_0} = -l_{v_k} \\
& a_{v'_j} = \tilde{a}_{v'_{j-1}} + s'_{v'_{j-1}} + t_{v'_{j-1}, v'_j} = \tilde{a}_{v_{k-j+1}} + s_{v_{k-j}} + t_{v_{k-j+1}, v_{k-j}} && 1 \le j \le k \\
& \tilde{a}_{v'_j} = \begin{cases} \max\left\{a_{v'_j}, e_{v'_j}\right\} & \text{if } a_{v'_j} \le l_{v'_j} \\ l_{v'_j} & \text{otherwise} \end{cases} = \begin{cases} \max\left\{a_{v_{k-j}}, -l_{v_{k-j}}\right\} & \text{if } a_{v_{k-j}} \le -e_{v_{k-j}} \\ -e_{v_{k-j}} & \text{otherwise.} \end{cases} && 1 \le j \le k
\end{aligned}$$

To transform results back to the original route, we replace any $a_{v'_j}$ by $-z_{v_{k-j}}$ and any $\tilde{a}_{v'_j}$ by $-\tilde{z}_{v_{k-j}}$ respectively.

$$\begin{aligned}
& -z_{v_k} = -\tilde{z}_{v_k} = -l_{v_k} \\
& -z_{v_{k-j}} = -\tilde{z}_{v_{k-j+1}} + s_{v_{k-j}} + t_{v_{k-j+1}, v_{k-j}} && 1 \le j \le k \\
& -\tilde{z}_{v_{k-j}} = \begin{cases} \max\left\{-z_{v_{k-j}}, -l_{v_{k-j}}\right\} & \text{if } -z_{v_{k-j}} \le -e_{v_{k-j}} \\ -e_{v_{k-j}} & \text{otherwise.} \end{cases} && 1 \le j \le k
\end{aligned}$$

Additionally, we switch the order of enumeration from $k-j$ to j and multiply every equation by -1 to attain the formulation of the backward time window violation.

$$\begin{aligned}
& z_{v_k} = \tilde{z}_{v_k} = l_{v_k} \\
& z_{v_j} = \tilde{z}_{v_{j+1}} - s_{v_j} - t_{v_{j+1}, v_j} && 0 \le j \le k-1
\end{aligned}$$

$$\tilde{z}_{v_j} = \begin{cases} \min\left\{z_{v_j}, l_{v_j}\right\} & \text{if } z_{v_j} \geq e_{v_j} \\ e_{v_j} & \text{otherwise.} \end{cases} \qquad 0 \leq j \leq k-1$$

In the following lemma, we state the dominance criterion for the backward violation remainder. Note that the transformation described above induces a switch of the roles of $\tilde{z}_i^{g^i}$ and $\tilde{z}_i^{f^i}$ in Equation (55).

Lemma 9 *Let r be a route and let $\hat{f}^i, \hat{g}^i \in \hat{\mathcal{F}}^i(r)$ be partial time window assignments on r up to customer $i \in \{0, \ldots, k-1\}$. If Equations (54) and (55) hold, $\hat{f}^i$ dominates $\hat{g}^i$.*

$$\overrightarrow{\lambda}_i^{tw}(r, f^i) \leq \overrightarrow{\lambda}_i^{tw}(r, g^i) \tag{54}$$

$$\tilde{z}_i^{g^i} - \tilde{z}_i^{f^i} \leq \overrightarrow{\lambda}_i^{tw}(r, g^i) - \overrightarrow{\lambda}_i^{tw}(r, f^i) \tag{55}$$

Appendix B Proof of Lemma 8

Proof of Lemma 8. Adapted from the single time window case (cf. Schneider et al. (2013))

$$TW(r, f^i \oplus \hat{f}^i) = \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) + \overleftarrow{\lambda}_i^{\text{tw}}(r, \hat{f}^i) + \max\left\{0, \tilde{a}_i^{f^i} - \left(\tilde{z}_{i+1}^{\hat{f}^i} - s_i - t_{i,i+1}\right)\right\}$$

holds and can be simplified to

$$TW(r, f^i \oplus \hat{f}^i) = \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) + \overleftarrow{\lambda}_i^{\text{tw}}(r, \hat{f}^i)$$

because of Equation (52). Because $\hat{g}^i$ does not dominate $\hat{f}^i$ and because Equation(52) implies $\tilde{z}_{i+1}^{\hat{f}^i} \geq \tilde{z}_{i+1}^{\hat{g}^i}$, we conclude

$$TW(r, f^i \oplus \hat{f}^i) < \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) + \overleftarrow{\lambda}_i^{\text{tw}}(r, \hat{g}^i). \tag{56}$$

Because Equation (56) equals $TW(r, f^i \oplus \hat{g}^i)$ this concludes the proof:

$$\begin{aligned} TW(r, f^i \oplus \hat{f}^i) &= \overrightarrow{\lambda}_i^{\text{tw}}(r, f^i) + \overleftarrow{\lambda}_i^{\text{tw}}(r, \hat{g}^i) + \max\left\{0, \tilde{a}_i^{f^i} - \left(\tilde{z}_{i+1}^{\hat{g}^i} - s_i - t_{i,i+1}\right)\right\} \\ &= TW(r, f^i \oplus \hat{g}^i). \end{aligned}$$

□

Appendix C Pseudo codes

In this section, we detail pseudo codes for subroutines of our algorithm. Figure 16 details a binary search to identify a customer's earliest time window which closes after the arrival time at this customer. If no time window closes after the arrival time, the first time window that opens after the arrival time is returned. Figure 17 details the pseudo code to integrate a tree label into a list of non-dominated labels in constant time. Newly generated labels are only inserted if their time window assignment is not dominated by the time window assignment of any previously added label. Labels are removed from the list of labels if their time window assignment is dominated by the new label's time window assignment.

Input: Time of arrival a, sorted list of time windows $\Theta = \left[\left[e^1, l^1\right], \ldots, \left[e^\theta, l^\theta\right]\right]$
Initialization:
$low \coloneqq 1$
$high \coloneqq \theta$
$l^0 \coloneqq -\infty$
Iteration:
while True do
    $mid = low + \lfloor \frac{high - low}{2} \rfloor$
    if $a \leq l^{mid-1}$ **then**
        $high \coloneqq mid - 1$
    else if $a > l^{mid}$ **then**
        $low \coloneqq mid + 1$
    else
        break
Output: $\left[e^{mid}, l^{mid}\right]$

Figure 16: Pseudo code for the findSubsequentOrEnclosingTimeWindow subroutine

Input:
List $L = \left[\left(\tilde{a}_0, f_0, \lambda_0\right), \ldots, \left(\tilde{a}_{|L|}, f_{|L|}, \lambda_{|L|}\right)\right]$ with $\tilde{a}_0 < \cdots < \tilde{a}_{|L|}$, element $(\tilde{a}^*, f^*, \lambda^*)$
Iteration:
$i \coloneqq |L| - 1$
if $\tilde{a}_i > \tilde{a}^*$ **then**
    $i \coloneqq i - 1$
if f^i dominates f^* **then**
    Discard f^*
else if f^* dominates f^i **then**
    $L\left[i\right] \coloneqq (\tilde{a}^*, f^*, \lambda^*)$
else
    Insert $(\tilde{a}^*, f^*, \lambda^*)$ into L between i and $i + 1$
Output: L

Figure 17: Pseudo code for the smartAppend subroutine

Appendix D Algorithmic component analysis

To evaluate the contribution of single components of the algorithm on the solution quality, we conducted an algorithmic component analysis. We present the results in Table 5. In the analysis, we tentatively removed every single component of our algorithm and assessed the impact of the removal, i.e., whether or not a benchmark value was improved or impaired. As benchmark value we took the average solution value of the best solutions out of ten runs for the instances used for parameter fitting (cf. Section 4.1). We provide the deviation $\overline{\Delta\lambda}$ between the reference value and the value derived with the respective component configuration. If the solution quality decreased ($\overline{\Delta\lambda} > 0$), we kept this component in our algorithm, since it contributes to solution quality. If removing a component resulted in better results ($\overline{\Delta\lambda} > 0$) or the results were equal but the computation time decreased, we removed this component from our algorithm. We tested all components in certain steps (from now on referred to as blocks), investigating the basic algorithmic components (A) first. Afterwards, we tested the local search operators (B) and the variable descent neighborhood component (C) (cf. Section 3.1.4). Finally, we tested the destroy operators (D) and the repair operators (E) (cf. Section 3.1.3). Results for the single blocks are reported below:

A: We tested the basic algorithmic components, i.e., the local search component (A1), the restart component (A2), the time window evaluation mechanism, i.e., switching between heuristic and optimal time window evaluation (A3), an adaptive destroy and repair operator selection where we adjust the probability for chosing destroy and repair operators depending on their success in finding improving solutions (A4), and the best-improvement acceptance criterion (by replacing it with a first-improvement acceptance criterion) (A5). We found that the adaptive component (A4) impairs the solution quality and all other components contribute to the quality of the solution.

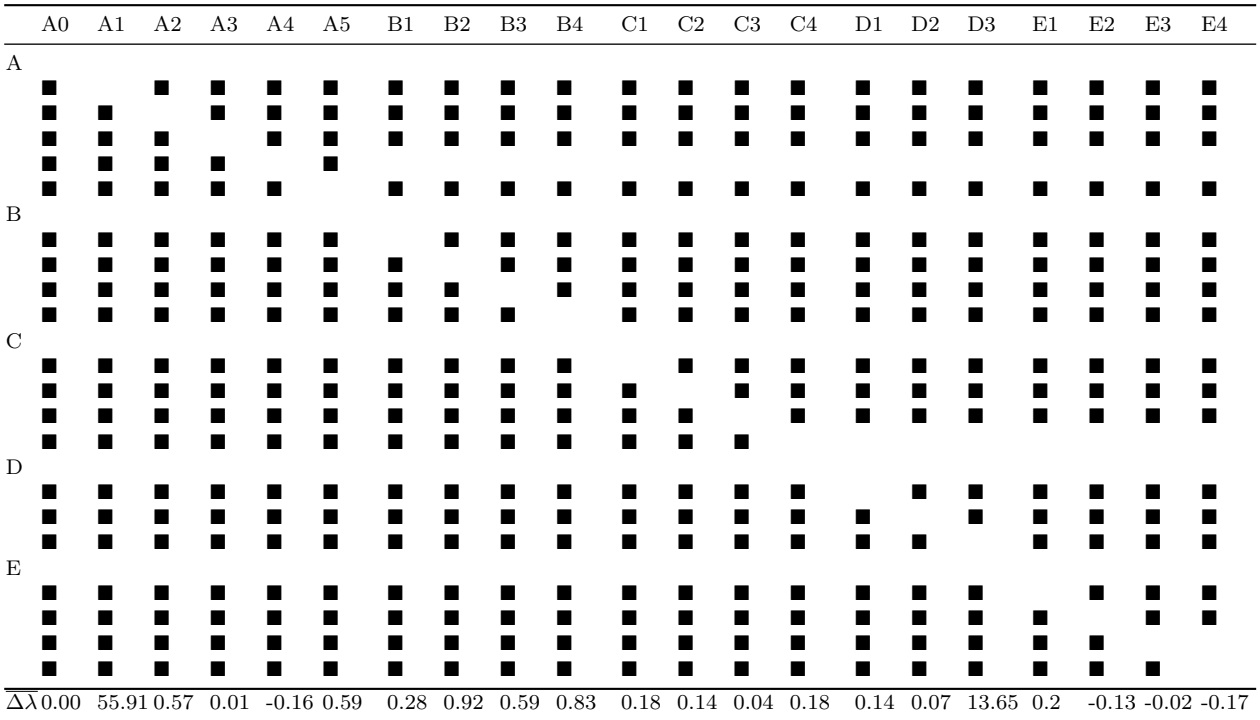
B: We tested the local search operators, i.e., the relocate operator (B1), the exchange operator (B2), the 2-opt* operator (B3), and the Or-opt operator (B4). All four local search operators contribute to the quality of the solution.

C: We tested the variable descent neighborhood component for all operators, i.e., for the relocate operator (C1), for the exchange operator (C2), for the 2-opt* operator (C3), and for the Or-opt operator (C4). All four variable descent neighborhood components contribute to the quality of the solution.

D: We tested the destroy operators, i.e., the worstRemove operator (D1), the relatedRemove operator (D2), and the routeRemove operator (D3). All three components contribute to the quality of the solution.

E: We tested the repair operators, i.e., the sequentialInsertion operator (E1), the sequentialPertubatedInsertion operator (E2), and the randomInsertion operator. Because the sequentialPertubatedInsertion and the randomInsertion operator impair the soltuon quality, we ran additional tests with both components removed (E4). This improves the solution quality even further so we only kept the sequentialInsertion operator.

Table 5: Algorithmic component analysis



	A0	A1	A2	A3	A4	A5	B1	B2	B3	B4	C1	C2	C3	C4	D1	D2	D3	E1	E2	E3	E4
$\overline{\Delta\lambda}$	0.00	55.91	0.57	0.01	-0.16	0.59	0.28	0.92	0.59	0.83	0.18	0.14	0.04	0.18	0.14	0.07	13.65	0.2	-0.13	-0.02	-0.17

Appendix E Supplementary tables

In the following, we present additional tables.

Table 6 depicts how the time window violation can be updated for different local search operators by combining propagation and concatenation.

Table 7 details these time window distributions, denoting the minimum and maximum number of time windows ($\underline{p}$, $\overline{p}$), temporal distance between two consecutive time windows ($\underline{d}$, $\overline{d}$), and the minimum and maximum time window width ($\underline{w}$, $\overline{w}$).

Table 6: Time window violation update in local search

Operator	Route(s) before and after move	Recalculation steps
relocate (Intra route)	$\langle 0,\dots,v_{x-1},\boldsymbol{v_x},\dots,v_y,v_{y+1},\dots,n\rangle$ ↓ $\langle 0,\dots,v_{x-1},\dots,v_y,\boldsymbol{v_x},v_{y+1},\dots,n\rangle$	• forward propagation from v_{x-1} to v_x • concatenate $\langle 0,\dots,\boldsymbol{v_x}\rangle$ and $\langle v_{y+1},\dots,n\rangle$
relocate (Inter route)	$\langle 0,\dots,v_{x-1},\boldsymbol{v_x},v_{x+1},\dots,n\rangle$ $\langle 0,\dots,v_{y-1},v_y,\dots,n\rangle$ ↓ $\langle 0,\dots,v_{x-1},v_{x+1},\dots,n\rangle$ $\langle 0,\dots,v_{y-1},\boldsymbol{v_x},v_y,\dots,n\rangle$	• concatenate $\langle 0,\dots,v_{x-1}\rangle$ and $\langle v_{x+1},\dots,n\rangle$ • forward propagation from v_{y-1} to v_x • concatenate $\langle 0,\dots,\boldsymbol{v_x}\rangle$ and $\langle v_y,\dots,n\rangle$
exchange (Intra route)	$\langle 0,\dots,v_{x-1},\boldsymbol{v_x},\dots,\boldsymbol{v_y},v_{y+1},\dots,n\rangle$ ↓ $\langle 0,\dots,v_{x-1},\boldsymbol{v_y},\dots,\boldsymbol{v_x},v_{y+1},\dots,n\rangle$	• forward propagation from v_{x-1} to v_x • concatenate $\langle 0,\dots,\boldsymbol{v_x}\rangle$ and $\langle v_{y+1},\dots,n\rangle$
exchange (Inter route)	$\langle 0,\dots,v_{x-1},\boldsymbol{v_x},v_{x+1},\dots,n\rangle$ $\langle 0,\dots,v_{y-1},\boldsymbol{v_y},v_{y+1},\dots,n\rangle$ ↓ $\langle 0,\dots,v_{x-1},\boldsymbol{v_y},v_{x+1},\dots,n\rangle$ $\langle 0,\dots,v_{y-1},\boldsymbol{v_x},v_{y+1},\dots,n\rangle$	• forward propagation from v_{x-1} to $\boldsymbol{v_y}$ • concatenate $\langle 0,\dots,\boldsymbol{v_y}\rangle$ and $\langle v_{x+1},\dots,n\rangle$ • forward propagation from v_{y-1} to $\boldsymbol{v_x}$ • concatenate $\langle 0,\dots,\boldsymbol{v_x}\rangle$ and $\langle v_{y+1},\dots,n\rangle$
2-opt*	$\langle 0,\dots,v_{x-1},\boldsymbol{v_x},\boldsymbol{v_{x+1}},v_{x+2},\dots,n\rangle$ $\langle 0,\dots,v_{y-1},\boldsymbol{v_y},\boldsymbol{v_{y+1}},v_{y+2},\dots,n\rangle$ ↓ $\langle 0,\dots,v_{x-1},\boldsymbol{v_x},\boldsymbol{v_{y+1}},v_{y+2},\dots,n\rangle$ $\langle 0,\dots,v_{y-1},\boldsymbol{v_y},\boldsymbol{v_{x+1}},v_{x+2},\dots,n\rangle$	• concatenate $\langle 0,\dots,\boldsymbol{v_x}\rangle$ and $\langle v_{y+1},\dots,n\rangle$ • concatenate $\langle 0,\dots,\boldsymbol{v_y}\rangle$ and $\langle v_{x+1},\dots,n\rangle$
Or-opt* (sequence length r)	$\langle 0,\dots,v_{x-1},\boldsymbol{v_x},\dots,\boldsymbol{v_{x+r}},v_{x+r+1},\dots,n\rangle$ $\langle 0,\dots,v_y,v_{y+1},\dots,n\rangle$ ↓ $\langle 0,\dots,v_{x-1},v_{x+r+1},\dots,n\rangle$ $\langle 0,\dots,v_y,\boldsymbol{v_x},\dots,\boldsymbol{v_{x+r}},v_{y+1},\dots,n\rangle$	• concatenate $\langle 0,\dots,v_{x-1}\rangle$ and $\langle v_{x+r+1},\dots,n\rangle$ • forward propagation from v_y to $\boldsymbol{v_{x+r}}$ • concatenate $\langle 0,\dots,\boldsymbol{v_{x+r}}\rangle$ and $\langle v_{y+1},\dots,n\rangle$

Table 7: Distribution of time windows in instances

Inst.	$\underline{p}$	$\overline{p}$	$\underline{d}$	$\overline{d}$	$\underline{w}$	$\overline{w}$	Inst.	$\underline{p}$	$\overline{p}$	$\underline{d}$	$\overline{d}$	$\underline{w}$	$\overline{w}$	Inst.	$\underline{p}$	$\overline{p}$	$\underline{d}$	$\overline{d}$	$\underline{w}$	$\overline{w}$
rm101	5	9	10	10	10	30	cm101	5	10	10	50	50	100	rcm101	5	10	10	30	10	30
rm102	5	7	10	30	10	30	cm102	5	7	10	70	50	100	rcm102	5	7	10	30	10	50
rm103	4	7	10	50	10	30	cm103	3	7	10	100	50	100	rcm103	3	7	10	50	10	50
rm104	3	6	10	70	10	30	cm104	3	5	10	100	50	100	rcm104	3	5	10	50	10	50
rm105	2	6	10	100	10	30	cm105	2	5	50	200	50	100	rcm105	2	5	10	70	10	70
rm106	2	3	50	100	30	50	cm106	2	4	50	200	100	200	rcm106	2	4	30	70	30	70
rm107	1	3	50	150	30	50	cm107	1	3	100	300	100	200	rcm107	1	3	30	100	30	70
rm108	1	2	100	200	50	100	cm108	1	3	100	500	100	500	rcm108	1	3	30	100	30	100
rm201	5	8	50	100	50	100	cm201	5	10	100	150	50	100	rcm201	5	10	100	150	50	100
rm202	3	5	50	300	50	100	cm202	5	7	100	200	50	100	rcm202	5	7	100	200	50	100
rm203	2	5	50	500	50	100	cm203	3	7	100	300	50	100	rcm203	3	7	100	300	50	100
rm204	2	4	50	700	50	100	cm204	3	5	100	500	50	100	rcm204	3	5	100	500	50	100
rm205	1	4	50	1000	50	100	cm205	2	5	200	500	100	200	rcm205	2	5	200	500	100	200
rm206	1	3	100	1000	100	200	cm206	2	4	200	700	100	200	rcm206	2	4	200	700	100	200
rm207	1	3	200	1000	100	200	cm207	1	3	200	1000	100	300	rcm207	1	3	200	1000	100	300
rm208	1	5	500	1000	100	200	cm208	1	3	500	1000	100	500	rcm208	1	3	500	1000	100	500

Abbreviations hold as follows: $\underline{p}$ - minimum number of time windows per customer, $\overline{p}$ - maximum number of time windows per customer, $\underline{d}$ - minimum distance between time windows, $\overline{d}$ - maximum distance between time windows, $\underline{w}$ - minimum width of time windows, $\overline{w}$ - maximum width of time windows.

References

Belhaiza S, Hansen P, Laporte G (2014) A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows. Computers & Operations Research 52:269–281.

Belhaiza S, M'Hallah R, Brahim GB (2017) A new hybrid genetic variable neighborhood search heuristic for the vehicle routing problem with multiple time windows. IEEE Congress on Evolutionary Computation (CEC) (San Sebastian, Spain).

Clarke G, Wright JW (1964) Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12(4):568–581.

Cordeau JF, Laporte G, Mercier A (2001) A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational Research Society 52(8):928–936.

Favaretto D, Moretti E, Pellegrini P (2007) Ant colony system for a VRP with multiple time windows and multiple visits. Journal of Interdisciplinary Mathematics 10(2):263–284.

Fisher ML (1994) Optimal solution of vehicle routing problems using minimum k-trees. Operations Research 42(4):626–642.

Hemmelmayr VC, Cordeau JF, Crainic TG (2012) An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. Computers & Operations Research 39(12):3215–3228.

Hiermann G, Puchinger J, Ropke S, Hartl RF (2016) The electric fleet size and mix vehicle routing problem with time windows and recharging stations. European Journal of Operational Research 252(3):995–1018.

Kindervater GAP, Savelsbergh MWP (1997) Vehicle routing: handling edge exchanges. Aarts E, Lenstra JK, eds., Local Search in Combinatorial Optimization (Boston, MA: John Wiley & Sons Ldt.), first edition.

Kytöjoki J, Nuortio T, Bräysy O, Gendreau M (2007) An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. Computers & Operations Research 34(9):2743–2757.

Mattos Ribeiro G, Laporte G (2012) An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. Computers & Operations Research 39(3):728–735.

Nagata Y, Bräysy O, Dullaert W (2010) A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. Computers & Operations Research 37(4):724–737.

Or I (1976) Traveling salesman-type problems and their relation to the logistics of regional blood banking. Ph.D. thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, USA.

Pisinger D, Ropke S (2010) Large neighborhood search. Gendreau M, Potvin JY, eds., Handbook of Metaheuristics (Boston, MA: Springer US), second edition.

Potvin JY, Rousseau JM (1995) An exchange heuristic for routing problems with time windows. Journal of the Operational Research Society 46(12):1433–1446.

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

Schneider M, Sand B, Stenger A (2013) A note on the time travel approach for handling time windows in vehicle routing problems. Computers & Operations Research 40(10):2564–2568.

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. Principles and Practice of Constraint Programming — CP98 (Berlin, Heidelberg).

Solomon MM (1987) Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35(2):254–265.

Vidal T, Crainic TG, Gendreau M, Prins C (2014) A unified solution framework for multi-attribute vehicle routing problems. European Journal of Operational Research 234(3):658–673.