Solving an On-line Capacitated Vehicle Routing Problem with Structured Time Windows

P. Hungerländer[*], K. Maier[†], J. Pöcher[‡], A. Rendl[§] and C. Truden[¶]

25th February 2017

Abstract

The capacitated Vehicle Routing Problem with structured Time Windows (cVRPsTW) is concerned with finding optimal tours for vehicles with given capacity constraints to deliver goods to customers within assigned time windows that have a special *structure*: they are non-overlapping and can hold several customers. These special features stem from a real-world application, and we show how they can be exploited during the solving process.

In this work, we consider an *on-line* variant of the cVRPsTW, similar to the Home Delivery Problem that arises in the online shopping services of supermarket chains: customers choose a delivery time window for their order online, and the vehicle tours are updated accordingly in real time. This leads to two challenges. First, the new customers need to be inserted at a suitable place in one of the available tours. Secondly, the new customers have to be inserted in real time due to very high customer request frequency. This is why we apply a computationally cheap, two-step approach consisting of an insertion step and an improvement step. In this context, we present heuristics and two mixed-integer linear programs that are employed by the heuristics. In an experimental evaluation, we demonstrate the efficiency of our approaches on a variety of benchmark sets based on different scenarios that are motivated by our application.

1 Introduction

The online market has been a growing sector for decades, accounting for a considerable share of the global market. For instance, in EU-28, around 20 % of turnover is made through e-commerce, as reported in Eurostat (2015). The benefits of trading online are vast, for both seller and purchaser, such as lower costs or no geographical limitations. This is why customers are increasingly interested in making purchases through the internet.

In recent years this trend has also reached grocery shopping, for which traditionally customers physically go to a store or supermarket chain. Nowadays, all main supermarket chains provide online shopping services, where customers select groceries on the supermarket's website, as well as a delivery time window. Then the supermarket distributes the groceries to the customers within the time window that the customer selected.

[*]Laboratory for Information & Decision Systems, MIT, USA, philipp.hungerlaender@aau.at
[†]Department of Mathematics, Alpen-Adria Universität Klagenfurt, Austria, kerstin.maier@aau.at
[‡]Department of Mathematics, Alpen-Adria Universität Klagenfurt, Austria, joerg.poecher@aau.at
[§]Department of Mathematics, Alpen-Adria Universität Klagenfurt, Austria, andrea.rendl@aau.at
[¶]Department of Mathematics, Alpen-Adria Universität Klagenfurt, Austria, christian.truden@aau.at

This provides many benefits to the customer, such as 24-hour opening hours of the online store, quicker shopping times, the avoidance of traveling times, no carrying of heavy or bulky items, easy amendment of the purchase until shortly before delivery and facilitated access for persons with reduced mobility.

Despite the benefits for the customers, online shopping services pose three major interrelated optimization challenges to the supermarket chains:

1. **Computational complexity:** The supermarket has to operate a fleet of delivery vans and determine an efficient (optimized) delivery schedule (consisting of the van tours of its fleet) in order to service the customers within their specific time windows, which is a known, strongly NP-hard (e.g. see Madsen and Kohl (1997)) problem, the capacitated Vehicle Routing Problem with Time Windows (cVRPTW).
2. **On-line problem version:** The customers choose a delivery time window while making their online purchase, which leads suppliers to build their delivery schedule in an on-line fashion, where the tours of the delivery vans are updated and optimized as new customer orders come in.
3. **Handling large-scale instances in real-time:** All computations have to be performed as quickly as possible on large-scale instances with thousands of customers and hundreds of vans each day in order to provide a prompt online service to the customers, which is especially challenging during peak times with up to hundred customer requests per second.

In this paper, we tackle these challenges in the context of a large international supermarket chain that builds its fleet's schedule as new customer orders come in. This context is very similar to the Home Delivery Problem (HDP), introduced by Campbell and Savelsbergh (2005), with one major difference: in the HDP, the *system* (or algorithm) decides if a customer request is accepted or not, as well as the time window to which the customer is assigned to. In our problem setup, the *customer* makes these decisions.

More specifically, the customer places an order on the supermarket's website, which proposes time windows (based on the customer's address) during which the order can be delivered to the customer, who then selects a time window. The system then inserts the customer's order into the schedule in an on-line fashion, i.e. the system inserts orders into the tours as new customers come in, iteratively building the whole schedule. Between these *insertion steps* we invoke *improvement steps* to reduce the duration of the schedule (i.e. the sum of the travel times of the vans). Now let us take a closer look at both the insertion and improvement step that are executed alternately.

Insertion Step: The insertion step is concerned with inserting a new customer order into the current schedule and consists of three parts:

1. The new customer receives a selection of available time windows. The larger the selection, the more likely the customer will find a suitable time window, leading to higher customer satisfaction. Note that the fuller or less efficient the schedule, the more difficult it is to provide time windows for selection.
2. The new customer selects one of the available time windows.
3. The customer's order is inserted into the current delivery schedule within its assigned time window.

Improvement Step: After the insertion step, the system improves the current schedule by applying an improvement step. This step is essential to find as many feasible time windows as possible for the following customers and to schedule as many customers as possible in total.

Within both steps lies an optimization problem. The first one is concerned with computing all time windows at which a given new customer can be feasibly inserted. The second optimization problem deals with improving the existing, incomplete schedule, where the objective is to minimize the duration of the schedule without moving customers from their assigned time window. We denote the problem of stepwise (customer by customer) building an optimized schedule in real-time as on-line capacitated Vehicle Routing Problem with structured Time Windows (cVRPsTW).

Figure 1: Illustration of a typical cVRPsTW instance with 3 tours and 9 time windows of 1 hour each.

Special Structure of Time Windows: The difference between the cVRPsTW and the standard cVRPTW is the special *structure* of the time windows: time windows can hold several customers and are non-overlapping. These features are motivated by a real-world application related to a project of our working group with a large grocer. To simplify the exposition we additionally assume that the time windows are equidistant and consecutive. We refer to Figure 1 for a corresponding illustration and to Section 3 for a formal definition of the cVRPsTW and the online cVRPsTW.

Note that the structure of the time windows is set by the supplier who provides the customer with a selection of time windows to choose from. In our online shopping application, providing these special structural features does neither impose substantial restrictions to the supplier nor to the customers. Due to the strict time limits of our application, we strive to develop fast methods and our suggested solving approaches are able to computationally exploit this additional structure of the time windows quite efficiently.

In summary in this paper we tackle the on-line version of the cVRPsTW and provide the following two main contributions:

1. We suggest various heuristic approaches for both the insertion and the improvement step of the on-line cVRPsTW. Our approaches combine mixed-integer linear programs (MILPs) for solving selected sub-problems of the cVRPsTW and local search heuristics in different ways in order to handle varying customer request frequency levels. In particular our approaches are able to exploit the special structure of the time windows that is motivated by an application emerging in the context of a large international supermarket chain.
2. We give an extensive computational evaluation for four different application scenarios, illustrating that our approaches provide high-quality results while also complying with the varying and mostly very strict time limits required.

The paper is organized as follows. In Section 2 we review relevant literature related to the on-line cVRPsTW. In Section 3 we give a formal problem description of the cVRPsTW and the online cVRPsTW and state a corresponding MILP. In Section 4 we propose local search heuristics for the on-line cVRPsTW and in Section 5 we describe MILPs for optimizing sub-problems of the on-line cVRPsTW. In Section 6 we suggest how to combine our local search heuristics and the MILPs for conducting both standard and fast insertion and improvement steps. In Section 7 we show in a computational study the applicability of our approaches to the on-line cVRPsTW on a variety of benchmark instances that are based on four different application scenarios. Section 8 concludes the paper.

2 Related Work

The standard work for Vehicle Routing Problems (VRPs) is Toth and Vigo (2014) that gives an overview of different VRP types including an extensive overview of heuristics and integer programming approaches, as well as applications and case studies. El-Sherbeny (2010) gives a compact review of exact and heuristic solving approaches for the Vehicle Routing Problem with Time Windows (VRPTW). For a recent survey on the capacitated VRPTW we refer to Baldacci et al. (2012).

The VRPTW with multiple routes is discussed by Azi et al. (2010) and Macedo et al. (2011) in the context of delivering perishable goods. In this problem variant, vehicles perform multiple routes on a

working day, mainly for two reasons:

1. The number of customers is too large to service with one route per vehicle.
2. Short routes are compulsory to ensure delivery of the goods before they perish.

Azi et al. (2010) propose a column-generation based approach, where the master problem is a set covering model. In this model each column represents a vehicle's workday that is a sequence of routes. The pricing problem generates feasible workdays, and it is formulated as a shortest path problem with resource constraints. Macedo et al. (2011) present a pseudo-polynomial network flow model where the variables represent feasible vehicle routes. Their method outperforms the branch-and-price approach of Azi et al. (2010) on Solomon benchmarks, however, both approaches are only applicable to small instances (< 40 customers). The VRPTW with multiple routes is related to the cVRPsTW, since in our problem setup, there are also more customers than the vehicles can service in one tour. We overcome this issue by introducing several shifts per day and per vehicle. The main differences to the on-line cVRPsTW are that:

- The VRPTW with multiple routes is an off-line problem, where all customer orders with their assigned time windows are part of the input.
- In the VRPTW with multiple routes, a special time window structure is not considered (and hence exploited).

Another related VRP family is that of the dynamic VRPTW, where customer requests arrive *dynamically*, i.e. in real time, while the vehicles are operational. Typical applications are (emergency) patient transport or food delivery. Pillac et al. (2013) give a comprehensive review of state-of-the-art approaches and introduce the notion of *degree of dynamism*. Different approaches have been proposed to deal with the uncertainty of incoming requests. Bent and Van Hentenryck (2004) propose a multiple scenario approach that continuously generates routing plans for scenarios including known and future, stochastic requests. Hong (2012) propose a Large Neighbourhood Search approach (LNS) where the key feature is the dynamic decomposition of the dynamic VRPTW into a series of static VRPTWs. Whenever a new request arrives, the problem is solved as a static VRPTW via LNS, where the new request is part of the large neighbourhood. For a survey on dynamic stochastic VRPs we refer to Ritzinger et al. (2016). Though related, the dynamic VRPTW differs considerably from the cVRPsTW since:

- The dynamic VRPTW augments the schedule on-the-fly, when vehicles are already on their way, while in the cVRPsTW, the schedule is always completed before the vans start their tours.
- In the dynamic VRPTW a special time window structure is not considered (and hence exploited).
- In the cVRPsTW we do not consider stochastic requests.

The home delivery problem (HDP), introduced by Campbell and Savelsbergh (2005), is the most closely-related problem, where delivery requests arrive dynamically, and the system has to decide two things:

1. If a new request is accepted or not.
2. In which time window the new request should be scheduled.

The HDP is based on a similar application as our use case, with an important distinction: in our application, the system offers a selection of available time windows, and the customers decide upon acceptance and the time window.

Campbell and Savelsbergh (2005) propose a two-step insertion heuristic for the HDP: in the first step, they employ a construction heuristic, where, starting from an empty schedule, each already accepted request is inserted into the schedule, beginning with the "heaviest" requests first. The second step evaluates if the new request can be inserted into the constructed schedule in one of its acceptable time windows. Furthermore, the authors approximate the expected profit of accepting an incoming request. In their experimental evaluation, the heuristic provides good results, however, only on instances with up to 100

customers, which is much smaller than the instances we consider in our application (100 to 2000 customers). Furthermore, they do not consider (and hence exploit) a special time window structure and they also examine a different objective function.

Agatz et al. (2011) study the time slot management in the HDP which deals with the questions of which time slots to offer for each zip code so as to minimize expected delivery costs while meeting the service requirements. They propose a continuous approximation and an integer programming approach, each with different (opposing) strengths and weaknesses, and compare them to another.

Agatz et al. (2008) present issues and solving approaches for the *attended* HDP where customers select the time window during which delivery shall take place, in a very similar setup to our problem. Their particular focus is on e-grocers that sell groceries online and discuss the tactical planning issues related to the design of a time slot schedule, i.e. deciding what and how many time slots to offer the customers. Furthermore, they discuss dynamic time slotting as well as using penalties and incentives to smooth customer demands. This work is related to our application, however, it covers tactical considerations rather than operational difficulties and corresponding optimization approaches.

Yang et al. (2016) present a framework for the dynamic pricing of delivery time slots within the HDP. The authors estimate a multinomial logit customer choice model from historic data that is used to establish dynamic pricing policies. In a simulation study they show that considering the impact of future expected orders in the estimation of delivery cost produces higher profits. In order to check feasibility of insertion during the booking process, they propose a simple insertion heuristic that is very similar to our insertion heuristic. It also uses a similar concept of earliest and latest arrival times. However, they apply the insertion heuristic randomized on the set of (yet un-inserted) orders, and in a different context.

Gendreau et al. (1998) propose a generalized insertion heuristic for the Traveling Salesperson Problem with Time Windows (TSPTW). During the first phase a feasible solution is constructed using simple insertions and a backtracking procedure. The off-line setting allows to insert customer orders in a sensible sequence (which is not possible in an on-line environment) by applying a criterion for ranking the difficulty of the insertion. In the second phase the objective value is improved by removing and reinserting customers.

Finally, Ioannou et al. (2001) consider a real world problem similar to our application, but for the traditional cVRPTW (without dynamic requests). They present a look-ahead construction heuristic where the objective is to minimize the number of vehicles used. They give a mathematical model of the overall problem and present a heuristic which employs specialized customer selection and route insertion heuristics. Even though their approach can tackle instances with up to 1000 customers, their methods are not fast enough for the strict run time requirements of our application.

3 Problem Description and Modeling

The capacitated Vehicle Routing Problem with structured Time Windows (cVRPsTW) arises when delivering goods to customers who choose the delivery time window. The objective of the cVRPsTW is to find a feasible schedule with minimal travel time. The particularity of the cVRPsTW is the structure of the time windows, of which we assume the following two characteristics:

1. several customers fit into a single time window,
2. time windows are non-overlapping.

We exploit these characteristics in our solving approaches for the on-line cVRPsTW. To simplify the exposition we additionally assume that the time windows are equidistant and consecutive. In Figure 1 we illustrate a schedule for the cVRPsTW with 3 tours and 9 time windows that each has a duration of 1 hour.

In the following subsection we formally define the off-line cVRPsTW, where all customer orders with their assigned time windows are part of the input. In Subsection 3.2 we state a corresponding mixed-integer linear program (MILP) and finally in Subsection 3.3 we formally define the online cVRPsTW, where the time windows are assigned and the schedule is built stepwise (customer by customer) dynamically over

time through alternating insertion and improvement steps. Afterwards, in Sections 4 and 5 we suggest local search heuristics and MILPs for solving the on-line cVRPsTW.

3.1 Notation and Input Parameters

The off-line cVRPsTW corresponds to the cVRPTW with specially structured time windows. As input parameters we are given:

1. A set of *time windows* $\mathscr{W} = \{w_1, \dots, w_q\}$, where each time window $w \in \mathscr{W}$ is defined through its *start time* s_w and its *end time* e_w.
2. A *schedule* $\mathscr{S} = \{\mathscr{A}, \mathscr{B}, \dots\}$, consisting of $|\mathscr{S}| = m$ tours with assigned capacities $C_i,\ i \in \mathscr{S}$, that correspond to the *capacity* of the van that operates tour i.
3. A set of *customers* $\mathscr{C}$, with corresponding *order weight* function $c : \mathscr{C} \to \mathbb{R}^+$, *service time* function $s : \mathscr{C} \to \mathbb{R}^+$, and *travel time* function $t : \mathscr{C} \times \mathscr{C} \to \mathbb{R}^+$, where we set the travel time from a customer a to itself to 0, i.e. $t(a,a) = 0,\ a \in \mathscr{C}$.

The customer-related definitions from above can be summarized by considering a weighted graph $G = (V = \mathscr{C}, E = \mathscr{C} \times \mathscr{C})$ with two weight functions (c and s) defined for each vertex and one weight function (t) defined for each edge. In our application, we typically deal with asymmetric travel time functions for which the triangle inequalities hold, i.e. $t(u,v) \leq t(u,w) + t(w,v)$ for all $u, v, w \in \mathscr{C}$. However note that all our solving approaches presented in Sections 4 and 5 work with arbitrary asymmetric travel time functions.

The function $w : \mathscr{C} \to \mathscr{W}$ assigns a time window to each customer during which the delivery van has to arrive at the customer. We speak of *structured* time windows, if the number of customers $|\mathscr{C}| = p$ is much larger than the number of time windows $|\mathscr{W}| = q$, i.e. $p >> q$, and therefore typically several customers are assigned to the same time window. Further we assume all structured time windows to be non-overlapping, i.e. $e_w \leq s_{w+1},\ w \in [q-1]$, holds, where the sets $[u],\ u \in \mathbb{N}$, and $[u_0],\ u \in \mathbb{N}$, contain the elements $\{1, 2, \dots, u-1, u\}$ and $\{0, 1, 2, \dots, u-1, u\}$ respectively. For simplicity of exposition we set $e_w = s_{w+1},\ w \in [q-1]$, but note that all our models and algorithms can also be applied to the case $e_w < s_{w+1},\ w \in [q-1]$, with gaps between the structured time windows.

A *tour* $\mathscr{A} = \{a_1, a_2, \dots, a_n\}$ contains n customers, where the indices of the customers display the sequence in which the customers are visited. To improve clarity of exposition we sometimes additionally use upper indices, i.e. $\mathscr{A} = \{a_1^{w[a_1]}, a_2^{w[a_2]}, \dots, a_n^{w[a_n]}\}$, which indicate the time windows assigned to the customers. Furthermore each tour $\mathscr{A}$ has assigned *start* and *end times* that we denote as $start_{\mathscr{A}}$ and $end_{\mathscr{A}}$ respectively. Hence the van executing tour $\mathscr{A}$ can leave no earlier than $start_{\mathscr{A}}$ from the depot and must return to the depot no later than $end_{\mathscr{A}}$. The start depot a_0 and end depot a_{p+1} are formally assigned to time windows w_0 respectively w_{q+1} with start time 0 and the end time set to ∞. Without loss of generality we set the order weights and service times of the depots to zero: $c(a_0) = c(a_{p+1}) = s(a_0) = s(a_{p+1}) = 0$.

In Table 1 we summarize the notation and input parameters introduced above. Additionally in Figure 2 we depict three time windows of a tour to provide further clarification for some of the notions.

3.2 A MILP for the cVRPsTW

In this subsection we propose a Mixed Integer Linear Program (MILP) to formally describe the off-line cVRPsTW. Note that this model is not meant to be used in practice, since it does not scale on large instances. We merely introduce it for the sake of a clear problem definition. To simplify the exposition we further assume that there are no time windows without assigned customers for any of the tours.

In addition to the notation from the previous subsection we use the following sets, variables and parameters in our MILP model of the cVRPsTW:

- The set $[p_i]$ of customers assigned to time window $w_i \in \mathscr{W}$.

	Definition	Description
$\mathscr{A}$	$\mathscr{A} = \{a_1, a_2, \ldots, a_n\}$	tour with n customers
c	$c : \mathscr{C} \to \mathbb{R}^+$	order weight function
C	$C_{\mathscr{A}} \in \mathbb{R}^+$	capacity of tour $\mathscr{A}$
$\mathscr{C}$		set of customers
e_w		end time of time window $w \in \mathscr{W}$
$end_{\mathscr{A}}$		end time of tour $\mathscr{A}$
m	$m = \lvert\mathscr{S}\rvert$	number of tours assigned to schedule $\mathscr{S}$
n	$n = \lvert\mathscr{A}\rvert$	number of customers assigned to tour $\mathscr{A}$
p	$p = \lvert\mathscr{C}\rvert$	number of customers
q	$q = \lvert\mathscr{W}\rvert$	number of time windows
s	$s : \mathscr{C} \to \mathbb{R}^+$	service time function
s_w		start time of time window $w \in \mathscr{W}$
$\mathscr{S}$	$\mathscr{S} = \{\mathscr{A}, \mathscr{B}, \ldots\}$	schedule consisting of tours $\mathscr{A}, \mathscr{B}$, etc.
$start_{\mathscr{A}}$		start time of tour $\mathscr{A}$
t	$t : \mathscr{C} \times \mathscr{C} \to \mathbb{R}^+$	travel time function
$\mathscr{W}$	$\mathscr{W} = \{w_1, \ldots, w_q\}$	set of time windows

Table 1: List of input parameters and constants of the cVRPsTW.

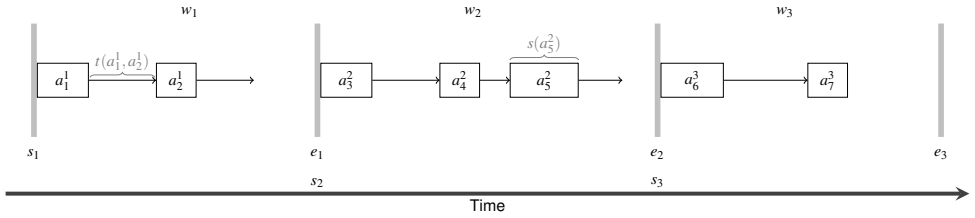

Figure 2: Illustration of three time windows with seven customers. The vertical bars show the start and end times of the respective time windows. The length of the arrows depict the travel times between customers and the vertical lengths of the boxes illustrate the service times of the customers.

- The index sets:
  - $I_0 = \{i \,:\, i \in [p_h],\ h \in [q_0]\}$,
  - $I_1 = \{j \,:\, j \in [p_\ell],\ \ell \in [q+1]\}$,
  - $I_2 = \{(i,j) \,:\, i \in [p_h],\ h \in [q_0],\ j \in [p_\ell],\ \ell \in [q+1],\ h \leq \ell \leq h+1,\ i \neq j\}$.
- The binary variables $x_{ijk},\ (i,j) \in I_2,\ k \in [m]$, with the following interpretation:

$$x_{ijk} = \begin{cases} 1, & \text{if customer } j \text{ is visited directly after customer } i \text{ on tour } k, \\ 0, & \text{otherwise.} \end{cases}$$

- The non-negative variables $d_i,\ i \in [p]$, modeling the arrival time at customer a_i.
- The non-negative variables $r_k,\ k \in [m]$, modeling the arrival time of tour k at the end depot.
- The non-negative parameters $t_{ij},\ (i,j) \in I_2$, giving the service time $s(a_i)$ at customer a_i plus the travel time $t(a_i, a_j)$ from customer a_i to customer a_j.
- Parameter $M = \max\limits_{k \in [m]} (end_k - start_k)$ stating the largest difference between start and end time of a tour.

Now we can formulate the off-line cVRPsTW as the following MILP:

$$\min \quad \sum_{k\in[m]} \sum_{(i,j)\in I_2} t_{ij}x_{ijk} \tag{1a}$$

$$\text{s.t.} \quad \sum_{k\in[m]} \sum_{\substack{i\in I_0,\ i\neq j\\ h\leq \ell\leq h+1}} x_{ijk} = 1, \quad j \in I_1, \tag{1b}$$

$$\sum_{k\in[m]} \sum_{\substack{j\in I_1,\ j\neq i\\ h\leq \ell\leq h+1}} x_{ijk} = 1, \quad i \in I_0, \tag{1c}$$

$$\sum_{k\in[m]} \sum_{i\in I_0} x_{i(n+1)k} = m, \tag{1d}$$

$$\sum_{k\in[m]} \sum_{j\in I_1} x_{0jk} = m, \tag{1e}$$

$$\sum_{\substack{i,j\in S,\\ i\neq j}} x_{ijk} \leq |S|-1, \quad S \subset [p_\ell],\ \ell \in [q],\ |S| \geq 2, \tag{1f}$$

$$\sum_{(i,j)\in I_2} c(a_j)x_{ijk} \leq C_k, \quad k \in [m], \tag{1g}$$

$$s_{w[a_i]} \leq d_i \leq e_{w[a_i]}, \quad i \in [p], \tag{1h}$$

$$d_j \geq d_i - M + x_{ijk}(t_{ij}+M), \quad i,j \in [p],\ i \neq j,\ k \in [m], \tag{1i}$$

$$r_k \geq d_j - M + (t_{i(p+1)}+M) \sum_{\substack{i\in I_0,\ i\neq j\\ h\leq \ell\leq h+1}} x_{ijk}, \quad j \in I_1,\ k \in [m], \tag{1j}$$

$$d_j \geq start_k + t_{0j}, \quad j \in [p+1],\ k \in [m], \tag{1k}$$

$$r_k \leq end_k, \quad k \in [m], \tag{1l}$$

$$x_{ijk}, \quad (i,j) \in I_2,\ k \in [m], \tag{1m}$$

$$d_i \geq 0, \quad i \in [p], \tag{1n}$$

$$r_k \geq 0, \quad k \in [m]. \tag{1o}$$

The objective function (1a) ensures minimization of the duration of the schedule. Equalities (1b) and (1c) guarantee that we visit and leave each customer exactly once. Similarly, equalities (1d) and (1e) guarantee that all vans leave the start depot and return to the end depot. Inequalities (1f) are the well-known subtour elimination constraints. The capacity constraints (1g) guarantee that on each tour the sum of the order weights of the assigned customers does not exceed the respective tour capacity. Inequalities (1h) ensure that all customers are serviced within their assigned time windows. Inequalities (1i) and (1j) connect the arrival times at the customers and at the end depot with the tours modeled through the x-variables. Finally inequalities (1k) and (1l) ensure that start and end times of all tours are respected.

Note that in contrast to our heuristics in Section 4 and specialized MILPs in Section 5, the structure of the time windows cannot be exploited in order to simplify the MILP model above. In fact, the above MILP reads similar to the standard MILP model for the cVRPTW from the literature with appropriate objective function. For instance, Kohl et al. (1999) and Bard et al. (2002) present similar mathematical models for the cVRPTW, both with differing objective functions to ours.

Nonetheless the optimal routes of the cVRPsTW are in general more efficient, because the search space is larger if many customers are assigned to the same time window, i.e. customers can more likely be interchanged on the same tour and between different tours. Hence the cVRPsTW can be viewed as consisting of q connected cVRPs and therefore as an in-between of cVRP and cVRPTW.

3.3 The on-line cVRPsTW

In this subsection we describe the underlying dynamic process of the on-line cVRPsTW. The process for each dynamic step is summarized in Algorithm 1. We are given a *working schedule* $\mathscr{S}$ that corresponds to a set of working tours for each van. Note that some (or all) tours may be incomplete, i.e. in some (or all) tours there may be space to insert an additional customer. In each dynamic step, a new customer request $\tilde{a}$ arrives that triggers a call of Algorithm 1.

Algorithm 1 ON-LINE CVRPSTW $(\tilde{a}, \mathscr{S})$

1: **Input:** New customer request $\tilde{a}$ and working schedule $\mathscr{S}$
2: Calculate the set of available time windows $\mathscr{T}$ during which $\tilde{a}$ can be scheduled in $\mathscr{S}$
3: **if** $\mathscr{T} \neq \emptyset$ **then**
4: offer $\mathscr{T}$ to customer
5: **if** customer accepts a time window $w \in \mathscr{T}$ **then**
6: insert $\tilde{a}$ into $\mathscr{S}$ at time window w, resulting in an updated schedule $\mathscr{S}'$
7: improve updated schedule $\mathscr{S}'$
8: return $\mathscr{S}'$
9: **else** return schedule $\mathscr{S}$ ▷ customer rejected $\mathscr{T}$
10: **end if**
11: **else** reject request $\tilde{a}$ and return schedule $\mathscr{S}$ ▷ there is no space in $\mathscr{S}$ for inserting $\tilde{a}$
12: **end if**

For every new request $\tilde{a}$ we first determine the set of time windows $\mathscr{T}$ during which $\tilde{a}$ can be served in at least one tour of the current schedule (line 2). If $\mathscr{T}$ is empty (line 3), then request $\tilde{a}$ cannot be scheduled in $\mathscr{S}$ and hence is rejected (line 11). Otherwise $\mathscr{T}$ is forwarded to the customer (line 4) who picks a time window (line 5) or cancels the request (line 9). In case the customer picks a time window, the new request $\tilde{a}$ is inserted into the working schedule (line 6), the working schedule is improved (line 7), and returned (line 8). After returning the updated working schedule the system waits for another new request to arrive.

The core process consists of two alternating steps: the *insertion* step and the *improvement* step.

Insertion Step: The insertion step consists of three parts:

1. Based on the working schedule determine a set of time windows into which the new customer order can be feasibly inserted.
2. The new customer selects one time window from the provided set.
3. Update the working schedule by inserting the new customer order within the selected time window.

Improvement Step: The improvement step is concerned with solving an off-line cVRPsTW, where the working schedule can be used for warm starting. Although the warm starting information available in our application does not help to reduce the computational complexity of the Traveling Salesperson Problem with Time Windows and its extensions (Böckenhauer et al. 2009), it can usually be very well exploited by heuristics to improve their practical performance. Also note that the improvement step might be (partly) skipped during times with high customer request frequency.

Hence in summary the on-line cVRPsTW can be viewed as an iterated solution of the off-line cVRPsTW, typically under special conditions (available warm start information, strict time limits), plus in between insertion steps that increase the number of customers by one at a time.

4 Heuristic Approaches

In this section we present heuristic approaches for the on-line cVRPsTW. First, we introduce the notions of earliest and latest arrival time that are used to concisely describe the feasibility of tours and insertions

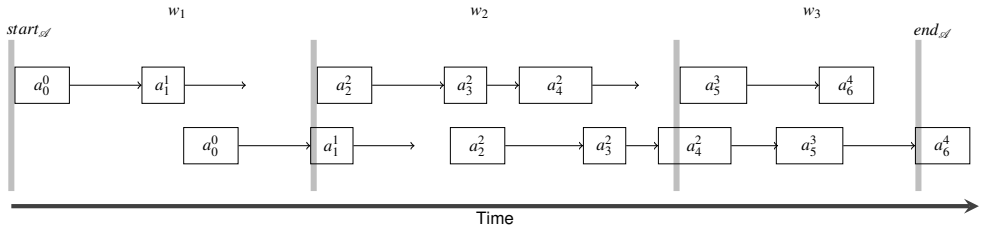

Figure 3: Illustration of earliest (above) and latest (below) arrival times at the customers and the depots for a toy example with one tour, three time windows and five customers. Note that the first time window starts when the start depot opens and the last time window ends when the end depot closes.

in Subsection 4.2. Building on these notions, in Subsection 4.3 we then present our heuristic approaches: a simple insertion procedure for inserting new customers and a local improvement heuristic for reducing the duration of the schedule.

4.1 Earliest and latest arrival times

We introduce the notion of earliest (latest) arrival time at customer a. It corresponds to the earliest (latest) time at which the delivery van may arrive at a, while not violating time window and travel time constraints on the preceding (subsequent) tour. This concept allows us to express the feasibility of tours and schedules in a concise way. Formally, the earliest (latest) arrival time at customer a, who is the i^{th} customer on the tour, is denoted as $\alpha_{a,i}$ ($\beta_{a,i}$).

Figure 3 illustrates the earliest and latest arrival times at the customers and the depots for a toy example with one tour, three time windows and six customers. In the upper tour each customer is scheduled at their *earliest* arrival time, and in the lower tour each customer is scheduled at their *latest* arrival time. It is easy to see that the earliest (latest) arrival times are obtained by shifting all customers as far left (right) on the time line as possible, while still respecting all travel times and ensuring that all customers are serviced within their assigned time windows.

Next we give a formal, recursive definition of the earliest and latest arrival time. Let us consider a fixed tour $\mathscr{A} = \{a_0, a_1, \ldots, a_n, a_{n+1}\}$, where a_0 is the start depot, a_{n+1} is the end depot and $\{a_1, \ldots, a_n\}$ is the set of customers assigned to tour $\mathscr{A}$. Note that our heuristics do not move the depots, and customers can only be inserted after the start depot and before the end depot.

Let us start with considering the base cases of our recursive definition. In (2a) we define the earliest arrival time at the start depot $\alpha_{a_0,0}$ as the earliest time at which a van is allowed to leave the start depot. Similarly, in (3a) we define the latest arrival time at the end depot $\beta_{a_{n+1},n+1}$ as the latest time the van is allowed to arrive at the end depot. Now we can recursively determine the earliest (latest) arrival time $\alpha_{a_j,j}$ ($\beta_{a_j,j}$) at each customer $a_j,\ j \in [n]$, starting with the first (last) customer of the tour:

$$\alpha_{a_0,0} = start_{\mathscr{A}}, \tag{2a}$$

$$\alpha_{a_{j+1},j+1} = \begin{cases} s_{w[a_{j+1}]}, & \text{if } g_{j+1} \leq s_{w[a_{j+1}]}, \\ g_{j+1}, & \text{if } g_{j+1} > s_{w[a_{j+1}]} \ \wedge\ g_{j+1} \leq e_{w[a_{j+1}]}, \\ \text{infeasible}, & \text{if } g_{j+1} > e_{w[a_{j+1}]}, \end{cases} \tag{2b}$$

$$\text{with } g_{j+1} = \alpha_{a_j,j} + s(a_j) + t(a_j, a_{j+1}),\ j \in [n]. \tag{2c}$$

$$\beta_{a_{n+1},n+1} = end_{\mathscr{A}}, \tag{3a}$$

$$\beta_{a_{j-1},j-1} = \begin{cases} e_{w[a_{j-1}]}, & \text{if } h_{j-1} \geq e_{w[a_{j-1}]}, \\ h_{j-1}, & \text{if } h_{j-1} \geq s_{w[a_{j-1}]} \wedge h_{j-1} < e_{w[a_{j-1}]}, \\ \text{infeasible}, & \text{if } h_{j-1} < s_{w[a_{j-1}]}, \end{cases} \tag{3b}$$

$$\text{with } h_{j-1} = \beta_{a_j,j} - s(a_{j-1}) - t(a_{j-1},a_j),\ j \in [n]. \tag{3c}$$

Note that g_{j+1} (h_{j-1}), as defined in (2c) ((3c)), corresponds to the earliest (latest) arrival time at customer a_{j+1} (a_{j-1}) without taking into account the time window constraints of customer a_{j+1} (a_{j-1}). In (2b) ((3b)) we distinguish three cases:

1. In the first case, the van arrives at customer a_{j+1} (a_{j-1}) before (after) the time window of customer a_{j+1} (a_{j-1}) has started (ended). Since a_{j+1} (a_{j-1}) cannot be serviced before (after) its time window has started (ended), the earliest (latest) arrival time $\alpha_{a_{j+1},j+1}$ ($\beta_{a_{j-1},j-1}$) is set to the start time $s_{w[a_{j+1}]}$ (end time $e_{w[a_{j-1}]}$) of a_{j+1}'s (a_{j-1}'s) time window.
2. In the second case, g_{j+1} (h_{j-1}) lies between the start and end time of a_{j+1}'s (a_{j-1}'s) time window. Hence in this case $\alpha_{a_{j+1},j+1}$ ($\beta_{a_{j-1},j-1}$) is set to g_{j+1} (h_{j-1}).
3. In the third case, the van arrives after (before) the time window of customer a_{j+1} (a_{j-1}) has ended (started). Hence a_{j+1} (a_{j-1}) cannot be serviced within its time window and $\alpha_{a_{j+1},j+1}$ ($\beta_{a_{j-1},j-1}$) is infeasible.

4.2 Feasibility of Tours and Insertions

A schedule is *feasible*, if all its tours are feasible and a tour $\mathscr{A}$ is *feasible* if and only if it satisfies both of the following conditions:

$$s_{w[a_i]} \leq \alpha_{a_i,i} \leq e_{w[a_i]}, \quad i \in [n], \qquad \text{Time Feasibilty } \texttt{(TFEAS)},$$

$$\sum_{i\in[n]} c(a_i) \leq C_{\mathscr{A}}, \qquad \text{Capacity Feasibility } \texttt{(CFEAS)}.$$

Now we can use the previously introduced notions of earliest and latest arrival time to facilitate and algorithmically speed up feasibility checks of tours after inserting an additional customer. A new customer $\tilde{a}^w$ can be feasibly inserted *with respect to time* between customers a_i and $a_{i+1},\ i \in [n_0]$, into a feasible tour $\mathscr{A}$ if the following condition holds:

$$\alpha_{\tilde{a}^w,i+1} \leq \beta_{\tilde{a}^w,i+1}, \qquad \text{TFEAS}(\tilde{a}^w, i+1, \mathscr{A}). \tag{4}$$

This condition ensures that we arrive at customer $\tilde{a}^w$ early enough such that we can leave from $\tilde{a}^w$ early enough to handle all subsequent customers of $\mathscr{A}$ within their assigned time windows. We refer to Figure 4 for an illustration of the above condition.

Note that the generalized insertion heuristic proposed by Gendreau et al. (1998) uses concepts analog to our earliest and latest arrival times. Further they check the feasibility of possible insertions using two conditions that resemble (4).

Additionally we have to check if the sum of the weights of the customer orders assigned to tour $\mathscr{A}$ does not exceed the capacity $C_{\mathscr{A}}$. The insertion of $\tilde{a}^w$ into tour $\mathscr{A}$ is feasible *with respect to capacity* if the following condition holds:

$$\sum_{i\in[n]} c(a_i) + c(\tilde{a}^w) \leq C_{\mathscr{A}}, \qquad \text{CFEAS}(\tilde{a}, \mathscr{A}). \tag{5}$$

If we conduct an insertion that is feasible with respect to time and capacity, then we obtain a new feasible tour $\tilde{\mathscr{A}} = \{a_0, a_1, \ldots, a_i, \tilde{a}^w, a_{i+1}, \ldots, a_n, a_{n+1}\}$. Conditions (4) and (5) also form the basic building blocks of our insertion and our improvement heuristic that we describe in the following subsection.

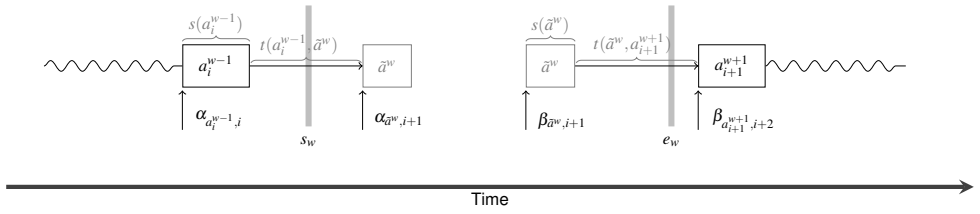

Figure 4: Illustration of a feasible insertion with respect to time of $\tilde{a}^w$ between a_i^{w-1} and a_{i+1}^{w+1}.

4.3 Insertion and Improvement Heuristics

In this subsection we present an insertion and an improvement heuristic for the on-line cVRPsTW. On the one hand we use our insertion heuristic for inserting a new customer into an existing schedule. On the other hand we also apply the insertion heuristic for determining time windows that allow for a feasible insertion of the new customer. Our local improvement heuristic is concerned with reducing the duration of the schedule and is called after inserting one (or several) new customers.

Simple Insertion Heuristic: Our simple insertion heuristic takes a new customer $\tilde{a}$, a tour $\mathscr{A}$ and a time window $w \in \mathscr{W}$ as inputs, and tries to insert $\tilde{a}$ into $\mathscr{A}$ at time window w. If successful, it returns the position at which customer $\tilde{a}$ can be inserted. Note that the simple insertion heuristic does not alter the order of customers on the given tour and therefore runs in linear time and is extremely fast in practice.

We summarize the workings of the simple insertion heuristic in Algorithm 2. First, the earliest and latest arrival times are calculated (line 3). Note that this step can be skipped if these values have already been previously determined. If the insertion is feasible with respect to capacity (line 4), then the algorithm iteratively attempts to insert the new customer between two adjacent customers a_i and a_{i+1}, $i \in [n_0]$, beginning with start depot and the first customer of the tour. As soon as a feasible insertion with respect to time is found (line 6), the position is stored (line 7), and the algorithm stops.

Algorithm 2 SIMPLE INSERTION HEURISTIC $(\tilde{a}^w, \mathscr{A})$

1: **Input:** New customer $\tilde{a}^w$ with assigned time window w and a tour $\mathscr{A}$.
2: **Set:** $P = -1$.
3: **Calculate:** $\alpha_{a_i,i}$ and $\beta_{a_i,i}$ for $i \in [(n+1)_0]$. ▷ Skip this step if these values are already known.
4: **if** CFEAS$(\tilde{a}, \mathscr{A})$ **then**
5: **for** $i \in [n_0]$ **do** ▷ Iterate over all potential insertion positions for $\tilde{a}^w$.
6: **if** TFEAS$(\tilde{a}^w, i+1, \mathscr{A})$ **then**
7: **Set:** $P = i+1$.
8: **Break loop**
9: **end if**
10: **end for**
11: **end if**
12: **Return:** P. ▷ P indicates the position at which $\tilde{a}^w$ can be feasibly inserted into $\mathscr{A}$.
13: If the insertion is not possible, P has value -1.

We iteratively apply the simple insertion heuristic to all time windows $w \in \mathscr{W}$ and all tours $[m]$ to calculate the set of time windows for feasibly inserting the new customer $\tilde{a}$. As soon as we find a tour for which the feasible insertion of $\tilde{a}$ into w is possible, we add w to our time window set and do not consider insertions of $\tilde{a}$ into w for the remainder of the tours.

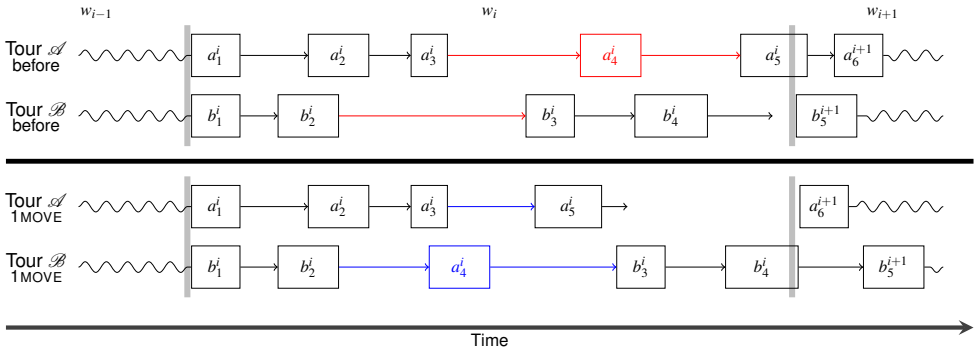

Figure 5: Reduction of the duration of the schedule induced by an improving $1\text{MOVE}(a_4^i, \mathscr{A}, \mathscr{B})$ operation.

From an algorithmic perspective we want to pre-compute as much information as possible to decide if the insertion of a new customer $\tilde{a}$ at a given position is feasible. In our set-up all pre-computed information is contained in the earliest and latest arrival times. Once we have computed the earliest and latest arrival times for all current customers, $\alpha_{\tilde{a}^w,i+1}$ and $\beta_{\tilde{a}^w,i+1}$ are determined by (2b) respectively (3b). Hence we only need a small number of simple calculations and a small number of entries from the travel matrix (which contains all travel times between pairs of customers) to decide with the help of conditions (4) and (5) whether $\tilde{a}^w$ can be feasibly inserted into tour $\mathscr{A}$ at position $i+1$.

This way, only very little information must be updated after inserting a new customer into the schedule. The benefit of a large amount of pre-calculations is that they can be done any time before a new customer request arrives. This is essential for our on-line problem that must be solved in real-time. The lower the effort for additional calculations, the faster we can respond to a new customer request.

Local Improvement Heuristic: In our local search heuristics we apply two neighborhoods for exchanging customer orders between two tours:

1. The *1-move* neighborhood moves a customer from one tour to another tour.
2. The *1-swap* neighborhood swaps two customers between two different tours.

As a $1\text{MOVE}(\tilde{a}^w, \mathscr{A}, \mathscr{B})$ operation we define the procedure where we remove customer $\tilde{a}$ from tour $\mathscr{A}$ and try to feasibly insert it into a different tour $\mathscr{B}$ during w. If at least one feasible insertion position for $\tilde{a}^w$ in $\mathscr{B}$ is found that decreases the duration of the schedule, we denote the *1-move* as *improving*. In Figure 5 we provide an illustration of an improving $1\text{MOVE}(a_4^i, \mathscr{A}, \mathscr{B})$ operation.

As a $1\text{SWAP}(\tilde{a}^w, \mathscr{A}, \mathscr{B})$ operation we define the procedure where we try to exchange customer $\tilde{a}^w$ with any customer with assigned time window w from a different tour $\mathscr{B}$. If at least one such exchange results in a decrease of the duration of the schedule, we denote the *1-swap* as *improving*. We always select the exchange of an improving 1-swap that results in the largest decrease of the duration of the schedule. In Figure 6 we provide an illustration of an improving $1\text{SWAP}(a_4^i, \mathscr{A}, \mathscr{B})$ operation.

Our local improvement heuristic combines *1-move* and *1-swap* operations, where we focus on the *1-move* operation when possible, as in general it is computationally cheaper and more effective than the *1-swap* operation. We stop our local improvement heuristic, if we reach a local minimum of our objective function with respect to our neighborhoods, i.e. if there is no *1-move* and *1-swap* anymore that reduces the duration of the schedule.

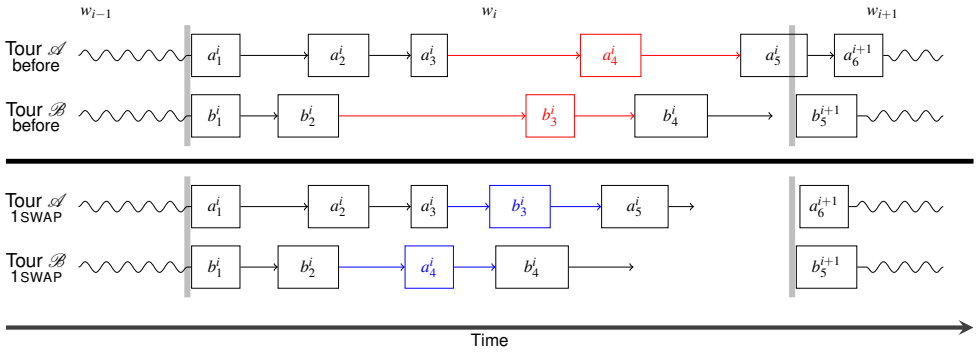

Figure 6: Reduction of the duration of the schedule induced by an improving $1\text{SWAP}(a_4^i, \mathscr{A}, \mathscr{B})$ operation.

5 MILPs for Single Tours and Single Time Windows

Large-scale instances of the on-line cVRPsTW are practically impossible to solve to optimality with exact methods in real-time. This is why we suggest heuristic approaches that employ Mixed Integer Linear Programs (MILPs) for sub-problems of the cVRPsTW. In the following subsection we discuss the sub-problem concerned with minimizing the duration (i.e. sum of travel times) of a single tour, while Subsection 5.2 deals with the sub-problem of minimizing the sum of travel times of all tours during a selected time window. In the following section we propose how to combine these two MILPs with our heuristic approaches described in the previous section for conducting both standard and fast insertion and improvement steps.

5.1 Traveling Salesperson Problem with Structured Time Windows

The Traveling Salesperson Problem with structured Time Windows (TSPsTW) is concerned with minimizing the duration of a single tour $\mathscr{A}$ of the cVRPsTW. All other tours in the schedule $\mathscr{S}$, except for $\mathscr{A}$, are fixed. We refer to Figure 7 for an illustration of the TSPsTW.

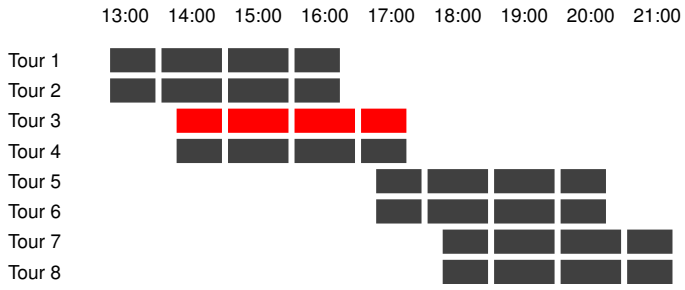

Figure 7: Illustration of the TSPsTW, where we optimize Tour 3 and fix the other 7 tours.

As the TSPsTW only models a single tour, it is not necessary to include a capacity constraint into our MILP below. The total capacity of a tour is independent of the sequence of the customers on the tour and hence can be checked before considering the MILP. As in Section 4, we consider a tour $\mathscr{A}$, where a_0 and a_{n+1} are the start and end depot of $\mathscr{A}$. If there are time windows that have no customers assigned, we exclude them from the set $[q]$. Additionally we use the following sub-problem specific variables:

- The non-negative variables $w_i,\ i \in [(q-1)_0]$, give the wait time during time window i, where the wait time of $\mathscr{A}$ is the time between start and end time of the tour that is neither travel time nor service time. Wait time arises when a time window is not fully 'filled' with customers so that the van has to wait until the start of the next time window to serve the next customer. Furthermore we set the wait time of the last customer time window w_q to zero because we should immediately drive to the end depot after servicing the last customer.
- The binary variables $x_{ij} \in \{0,1\},\ (i,j) \in I_2$, with the interpretation:

$$x_{ij} = \begin{cases} 1, & \text{if customer } a_j \text{ is visited directly after customer } a_i, \\ 0, & \text{otherwise.} \end{cases}$$

Now we can formulate the TSPsTW as the following MILP:

$$\min \quad \sum_{(i,j)\in I_2} t_{ij} x_{ij} \tag{6a}$$

$$\text{s.t.} \quad \sum_{\substack{i\in I_o,\ i\neq j \\ h\leq \ell \leq h+1}} x_{ij} = 1, \quad j \in I_1, \tag{6b}$$

$$\sum_{\substack{j\in I_1,\ j\neq i \\ h\leq \ell \leq h+1}} x_{ij} = 1, \quad i \in I_0, \tag{6c}$$

$$\sum_{\substack{i,j\in S, \\ i\neq j}} x_{ij} \leq |S|-1, \quad S \subset [p_k],\ k \in [q],\ |S| \geq 2, \tag{6d}$$

$$\sum_{\substack{(i,j)\in I_2 \\ h\leq k-1}} t_{ij} x_{ij} + \sum_{i\in[(k-1)_0]} w_i + start_{\mathscr{A}} \geq s_k, \quad k \in [q], \tag{6e}$$

$$\sum_{\substack{(i,j)\in I_2 \\ h,\ell\leq k}} t_{ij} x_{ij} + \sum_{i\in[k_0]} w_i + start_{\mathscr{A}} \leq e_k, \quad k \in [q], \tag{6f}$$

$$\sum_{(i,j)\in I_2} t_{ij} x_{ij} + \sum_{i\in[q_0]} w_i + start_{\mathscr{A}} \leq end_{\mathscr{A}}, \tag{6g}$$

$$x_{ij} \in \{0,1\}, \quad (i,j) \in I_2, \tag{6h}$$

$$w_i \geq 0, \quad i \in [(q-1)_0]. \tag{6i}$$

The objective function (6a) minimizes the duration of the tour. Equalities (6b) guarantee that we visit all customers and the end depot exactly once and equalities (6c) ensure that we leave the start depot and all customers exactly once. Inequalities (6d) model the subtour elimination constraints. Inequalities (6e) and (6f) guarantee that the arrival times at all customers are neither before the start nor after the end of their assigned time window. Finally inequalities (6g) ensure that we arrive at the end depot before the end time of tour $\mathscr{A}$.

Note that our MILP for the TSPsTW can be used for solving much larger instances to optimality than corresponding approaches for the closely-related Traveling Salesperson Problem with Time Windows. For a detailed discussion we refer to our forthcoming paper (Hungerländer et al. 2017).

5.2 Single Time Window Problem

The Single Time Window Problem (STWP) is concerned with minimizing the duration of (all) tours during a single time window w of the schedule, while keeping the tours during all other time windows (before and after w) fixed. More formally let $\mathscr{A} = \left\{a_0^0, a_1^1, \ldots a_i^{w-1}, a_{i+1}^w, \ldots, a_{j-1}^w, a_j^{w+1}, \ldots, a_n^q, a_{n+1}^{q+1}\right\}$ be a given tour, then we distinguish between the following three subtours:

1. The *pre time window subtour* $\left\{a_0^0, a_1^1, \dots a_i^{w-1}\right\}$ finishes at customer $a_i,\ i \in D_s$, where D_s is the set of start depots of the STWP.
2. The *post time window subtour* $\left\{a_j^{w+1}, \dots, a_n^q, a_{n+1}^{q+1}\right\}$ starts at customer $a_j,\ j \in D_f$, where D_f is the set of end depots of the STWP.
3. The *current time window subtour* $\left\{a_{i+1}^w, \dots, a_{j-1}^w, \right\}$ contains all customers assigned to time window w.

Now in the STWP we keep all pre and post time window subtours fixed, while minimizing the duration of the current time window subtours. For an illustration of the STWP we refer to Figure 8.

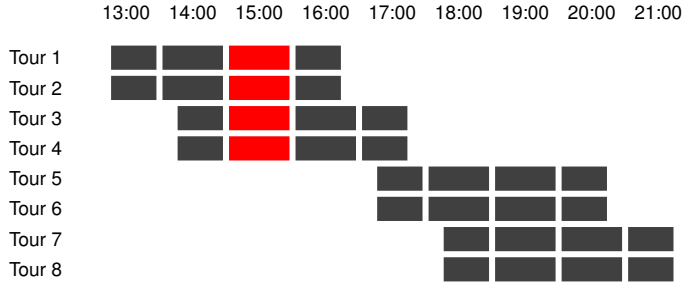

Figure 8: Illustration of the STWP, where we optimize all tours during the third time window and fix the tours during all other time windows.

In order to formulate a MILP modeling the STWP we first have to introduce the following sub-problem specific sets, variables and parameters:

- The sets $D_s = \{1, \dots, m\}$ and $D_f = \{n-m+1, \dots, n\}$ contain the m start and end depots respectively. The start (end) depot of tour $k \in [m]$ corresponds to the last (first) stop of tour k before (after) time window w. If w is the first (last) time window, then the start (end) depot of all tours is the actual start (end) depot.
- The sets $\overline{D}_s^i,\ i \in D_f$, contain the start depots that must not be on the same tour as the end depot i.
- The set $[p_w]$ contains all customers assigned to the current time window w plus the corresponding start and end depots $D_s \cup D_f$.
- The sets $[p_{\text{out}}] = [p_w] \setminus D_f$ and $[p_{\text{in}}] = [p_w] \setminus D_s$ contain all customers and depots with outgoing and ingoing edges respectively.
- The binary variables $x_{ij} \in \{0,1\},\ i \in [p_{\text{out}}],\ j \in [p_{\text{in}}],\ i \neq j$, with the interpretation:

$$x_{ij} = \begin{cases} 1, & \text{if customer } a_j \text{ is visited directly after customer } a_i, \\ 0, & \text{otherwise.} \end{cases}$$

- The continuous variables $y_i,\ i \in [p_w]$, state the sum of the order weights of the tour from the original start depot up to customer or depot a_i.
- The continuous variables $d_i,\ i \in [p_w]$, give the arrival time at customer or depot a_i.
- The variables $q_i \in D_s,\ i \in [p_w]$, indicate the start depot of the tour containing customer or depot a_i.
- The parameters $c_i,\ i \in [p_w] \setminus \{D_s \cup D_f\}$, give the order weight of customer a_i. Furthermore the parameters $c_i,\ i \in D_s$, state the sum of the order weights of the customers of the pre time window subtour finishing at start depot a_i. Analogously the parameters $c_i,\ i \in D_f$, give the sum of the order weights of the customers of the post time window subtour starting from end depot a_i.

- The parameters $m_1 = \min_{i \in D_s} c_i$, and $m_2 = \min_{i \in D_s} \alpha_{a_i,i}$ are set to (as tight as possible) lower bounds of our continuous variables y_i and $d_i,\ i \in [p_w]$, respectively.
- The parameters $m_3 = \max_{k \in [m]} C_k$ and $m_4 = \max_{i \in D_f} \beta_{a_i,i}$ are set to (as tight as possible) upper bounds of our continuous variables y_i and $d_i,\ i \in [p_w]$, respectively.
- The parameters

$$M_1 = \left(m_3 - \min_{i \in D_s} c_i\right) + \max_{i \in [p_w] \setminus \{D_s \cup D_f\}} c_i, \qquad M_2 = (e_w - s_w) + \max_{i \in [p_{\text{out}}], j \in [p_{\text{in}}], i \neq j} t_{ij}, \qquad M_3 = \max_{i \in D_s} i,$$

are used in the capacity, arrival time and tour assignment constraints respectively.

Now we can formulate the STWP as the following MILP:

$$\min \quad \sum_{\substack{i \in [p_{\text{out}}],\ j \in [p_{\text{in}}],\\ i \neq j}} t_{ij} x_{ij} \tag{7a}$$

$$\text{s.t.} \quad \sum_{\substack{i \in [p_{\text{out}}],\\ i \neq j}} x_{ij} = 1, \quad j \in [p_{\text{in}}], \tag{7b}$$

$$\sum_{\substack{j \in [p_{\text{in}}],\\ j \neq i}} x_{ij} = 1, \quad i \in [p_{\text{out}}], \tag{7c}$$

$$\sum_{\substack{i,j \in S,\\ i \neq j}} x_{ij} \leq |S| - 1, \quad S \subset [p_w] \setminus \left\{D_s \cup D_f\right\},\ |S| \geq 2, \tag{7d}$$

$$y_i = c_i + m_3 - C_i, \quad i \in D_s, \tag{7e}$$

$$y_j - y_i - M_1 x_{ij} \geq c_j - M_1, \quad i \in [p_{\text{out}}],\ j \in [p_{\text{in}}],\ i \neq j, \tag{7f}$$

$$y_i = m_3, \quad i \in D_f, \tag{7g}$$

$$d_i = \alpha_{a_i,i}, \quad i \in D_s, \tag{7h}$$

$$d_j - d_i - M_2 x_{ij} \geq t_{ij} - M_2, \quad i \in [p_{\text{out}}],\ j \in [p_{\text{in}}],\ i \neq j, \tag{7i}$$

$$s_w \leq d_i \leq e_w, \quad i \in [p_w] \setminus \left\{D_s \cup D_f\right\}, \tag{7j}$$

$$d_i = \beta_{a_i,i}, \quad i \in D_f, \tag{7k}$$

$$q_i = i, \quad i \in D_s, \tag{7l}$$

$$q_i \neq j, \quad i \in D_f,\ j \in \overline{D}_s^i, \tag{7m}$$

$$q_j - q_i - M_3 x_{ij} \geq -M_3, \quad i \in [p_{\text{out}}],\ j \in [p_{\text{in}}],\ i \neq j, \tag{7n}$$

$$q_j - q_i + M_3 x_{ij} \leq +M_3, \quad i \in [p_{\text{out}}],\ j \in [p_{\text{in}}],\ i \neq j, \tag{7o}$$

$$x_{ij} \in \{0,1\}, \quad i \in [p_{\text{out}}],\ j \in [p_{\text{in}}],\ i \neq j, \tag{7p}$$

$$m_1 \leq y_i \leq m_3,\ m_2 \leq d_i \leq m_4,\ q_i \in D_s, \quad i \in [p_{\text{in}}]. \tag{7q}$$

The objective function (7a) minimizes the duration of the schedule during time window w. Equalities (7b) guarantee that we visit all customers and the end depots exactly once, and equalities (7c) ensure that we leave the start depots and all customers exactly once. Inequalities (7d) are the well-known subtour elimination constraints. Note that in our computational experiments it proved to be most efficient to use only constraints with $|S| \leq 3$.

For modeling tours with different capacities, in (7e) we set $y_i,\ i \in D_s$, to the sum of the weights of the orders of the pre time window subtour finishing at the start depot a_i plus the difference $m_3 - C_i$. Inequalities (7f) guarantee that each order weight is considered on its assigned tour and equalities (7g) ensure that each tour respects its capacity constraint.

Equalities (7h) set the arrival time d_i of our start depots $i \in D_s$ to their earliest arrival times $\alpha_{a_i,i}$. Inequalities (7i) and (7j) guarantee that the arrival times at the customers consider all travel times correctly and are within the customers' assigned time windows. Equalities (7k) ensure that the arrival times at the end depots D_f are early enough to service the customers on the remainder of the tours within their assigned time windows. Note that we could also model constraints (7g) and (7k) as inequalities but using equalities leads to a significant speed-up when solving the MILP.

Constraints (7l) – (7o) allow to directly identify each of the tours through the values of the variables $q_i,\ i \in [p_w]$, and consequently to define feasible combinations of pre and post time window subtours. Equalities (7l) assign the indices $i \in D_s$ of the start depots to the variables q_i. Constraints (7m) exclude forbidden combinations of start and end depots. Note that such exclusions are e.g. necessary when considering overlapping tours with different shift patterns; for details see the experimental set-up of Scenario II in Section 7. Finally inequalities (7n) and (7o) guarantee that the variables $q_i,\ i \in [p_w]$, have the same value for customers and depots a_i assigned to the same tour.

6 Algorithmic Strategies

In this section we describe how to combine our approaches presented in the previous two sections for conducting both standard and fast insertion and improvement steps. Afterwards in Section 7, we suggest how to appropriately combine the different step types dependent on the customer request frequency that arises in different application scenarios.

6.1 The Insertion Step

In the insertion step we aim to quickly identify all time windows during which a new customer $\tilde{a}$ can be inserted into (at least one of) the working tours. We suggest to use the following procedure, where our three approaches for inserting a new customer are called successively:

1. Simple-**insertion:** Apply the simple insertion heuristic that is our computationally cheapest approach and runs within 1 millisecond (ms) for all benchmark instances considered in the following section.
2. TSPsTW-**insertion:** Use the TSPsTW MILP for all time windows $w \in \overline{\mathscr{T}}$, where $\overline{\mathscr{T}} = \mathscr{W} \setminus \mathscr{T}$ is the set containing all time windows that have not yet been identified as feasible. This MILP runs within a few hundred ms for all scenarios considered in our computational study and is applied to all tours covering w as follows:
   (a) Select a tour $\mathscr{A} \in \mathscr{S}$ that covers w.
   (b) Add $\tilde{a}$ to the set of customers of tour $\mathscr{A}$ that are assigned to w.
   (c) Try to solve the TSPsTW MILP for tour $\mathscr{A}$ without objective function.
   (d) Stop as soon as a feasible solution is found or a defined time limit is reached and update $\mathscr{T}$.
3. STWP-**insertion:** Use the STWP MILP for all time windows $w \in \overline{\mathscr{T}}$. This MILP takes between a few ms and several minutes dependent on the respective benchmark instance. We apply it as follows:
   (a) Add $\tilde{a}$ to the set of customers assigned to w.
   (b) Try to solve the STWP MILP for w without objective function.
   (c) Stop as soon as a feasible solution is found or a defined time limit is reached, and update $\mathscr{T}$.

Once the customer has selected a time window $\tilde{w}$ from the set $\mathscr{T}$, we insert $\tilde{a}$ into $\tilde{w}$ at the found insertion point. Note that we apply our insertion approaches in ascending order with respect to their computational effort such that we complete the average insertion step as fast as possible. During times with high customer

request frequency we suggest to apply a fast insertion step (i.e. only using simple-**insertion** and TSPsTW-**insertion**) or even a very fast insertion step (i.e only using simple-**insertion**) that can handle up to 1000 customer requests per second on all the benchmark instances considered in our computational study.

6.2 The Improvement Step

In the improvement step we aim to reduce the duration of the schedule. We solve all MILPs to optimality and use our working schedule for warm starting. We suggest to use one of the following four procedures, where our three approaches for improving the duration of the schedule are called in various combinations:

- Local-**improvement:** Our computationally cheap, yet quite effective local improvement heuristic builds the foundation of all our procedures.
- Local+TSPsTW-**improvement**: After the local improvement heuristic we additionally use our TSPsTW MILP for all tours that have changed since the last improvement step.
- Local+STWP+TSPsTW-**improvement**: After the local improvement heuristic we first apply our STWP MILP to the time window w into which the new customer $\tilde{a}^w$ has been inserted. Then we again use the TSPsTW MILP for all tours that have changed since the last improvement step.
- Local+3STWP+TSPsTW-**improvement**: After the local improvement heuristic we apply our STWP MILP to the time windows w, $w-1$ and $w+1$, where w is the time window into which the new customer $\tilde{a}^w$ has been inserted. Finally we again use the TSPsTW MILP for all tours that have changed since the last improvement step.

The above improvement procedures are arranged in ascending order with respect to their computational effort, and we choose them in our computational study dependent on the customer request frequency.

7 Computational Experiments

In this section we present computational results on a variety of benchmark sets based on different scenarios that are motivated by our application. First, we describe our benchmark instances and their related application scenarios in Subsection 7.1. In Subsection 7.2, we describe our experimental set-up and in Subsection 7.3 we present the results of our experiments and interpret them. Finally, in Subsection 7.4, we summarize the main findings of our computational evaluation.

7.1 Benchmark Instances and Scenarios

We design benchmark instances that reflect real-world problems as they arise in online shopping, regarding travel times, length of time windows, duration of service times, customer order weights and their proportions to van capacities. All instances can be downloaded from `http://tinyurl.com/vrpstw`.

We evaluate our algorithms on four scenarios that represent different instance setups for showcasing different features of our application and our solving approaches. Before discussing each scenario in detail, we give general information on our instances that holds for all scenarios:

- **Travel times:** The customers are randomly placed on a square-grid. Their coordinates are sampled from a two-dimensional uniform distribution and the travel times are calculated as the Euclidean distance between customers rounded to integers.
- **Customer weights:** The order weights of customers are sampled from a truncated normal distribution with mean of 7 and standard deviation of 2, where the lower bound is 1 and the upper bound is 15.

- **Time window preferences:** The customer preferences for particular time windows are simulated by randomly selecting a time window for each customer, following a uniform distribution, except for Scenario II, where we use a multinomial distribution.
- **Shift patterns:** All tours have same start and end times, except for Scenario II. Therefore, in Scenario I, III and IV, there are no illegal combinations of pre and post time window subtours and we can omit the tour assignment constraints (7l) – (7o) in the STWP MILP.

In Table 2 we highlight the differences between our four application scenarios.

Instances	Customers	Tours	Time Windows	Time Window Preferences	Tour Start and End Times	Improvment Step
Scenario I	100 200 300	5 6 7	5 10 15	uniform	same	after each insertion
Scenario II	250	5-7	13-15	multinomial	differs between tours	after each insertion
Scenario III	1000 1000	45-55 18-23	5 15	uniform	same	after each insertion
Scenario IV	2000	120	5	uniform	same	each *i*th insertion

Table 2: Overview of the characteristics of and differences between our four application scenarios.

Scenario I: Uniform Time Window Preferences. The first scenario corresponds to a standard setup, where the customers' preferences for time windows are uniformly distributed. Note that this is in contrast to real-world applications, where some time windows are more prominent among customers than others (which we explore in Scenario IV). However, we chose a uniform distribution to obtain even results over all time windows that allow for an easier identification and clearer interpretation of the key findings.

For this scenario we compare all our algorithmic strategies discussed in the previous section, both for the insertion and for the improvement step. We use instances with 100/200/300 customers, 5/10/15 time windows and 5/6/7 vans that have a capacity of 150/300/450 respectively.

Scenario II: Varying Tour Start and End Times. This scenario is used to illustrate the use of the tour assignment constraints (7l) – (7o). In contrast to the instances from Scenario I, the tours have different start and end times, where Tour i starts at the beginning of time window i. Tours with shifted start and end times provide two benefits:

1. *More efficient use of the loading infrastructure at the depot:* When using varying tour start and end times, the depot continuously services some vans throughout the whole day. This is in contrast to having peak times at the depot during which all vans must be loaded respectively unloaded at once, which happens if all tours have the same start and end times.
2. *Increase of delivery capacity:* Tour start and end times can be set to increase delivery capacity during time windows that are requested the most by the customers.

We set the time window preferences of the customers by sampling from a multinomial distribution, where each time window's probability is relative to the number of tours covering it. In this scenario, only pre and post time window subtours of the same original tour can be combined, i.e. $\overline{D}_s^i = D_s \setminus \{i\}$ for $j \in D_f$ and a_i and a_j are assigned to the same original tour. Hence in this scenario the tour assignment constraints (7l) – (7o) are needed in the STWP MILP. We again compare all our algorithmic strategies for both the insertion and the improvement step.

Scenario III: Large Instances. We also generate larger benchmark instances that have the same characteristics as in Scenario I, but contain 1000 customers. We consider two setups:

1. Many (45–55) short tours with a capacity of 150 and 5 time windows each.
2. A moderate number (18–23) of long tours with a capacity of 450 and 15 time windows each.

We compare all algorithmic strategies except for insertion and improvement strategies employing the STWP MILP that does not scale on these large-scale instances.

Scenario IV: Improvement step only every *i*th iteration. Finally, we consider a scenario with very high customer request frequency followed by longer periods of low customer activity that can be used to improve the schedule. Note that this frequency pattern is common in real-world applications. We generate benchmark instances that have the same characteristics as Scenarios I and III, but contain 2000 customers. To accommodate the high amount of customer requests, the improvement step is no longer run after each successful insertion step, but instead after every 10/25/50 successful insertions. Due to the instance sizes we again do not consider insertion and improvement strategies that employ the STWP MILP. Note that we do not run benchmarks on instances with more than 2000 customer orders, as for such instances loading the travel matrix becomes the computational bottleneck of the overall procedure in the distributed system of our online-application.

7.2 Experimental Setup

All experiments were performed on a Ubuntu 14.04 machine equipped with an Intel Xeon E5-2630V3 @ 2.4 GHz 8 core processor and 132 GB RAM. We use Gurobi 6.5.1 as an IP-solver in single thread mode.

In all our experiments we iteratively insert new customers into the schedule, simulating customers placing orders online, where the preferred time window of each customer is set beforehand in the benchmark instance. If the preferred time window is not offered to the customer, we assume that the customer does not place the order and hence the customer is not inserted into the schedule.

For each of the solving approaches that we apply in our experimental evaluation, we determine the following metrics:

- Insertion step:
  - Number of feasible time windows determined for each customer: corresponds to the number of time windows in which the order can be inserted.
  - Number of MILPs solved for both TSPsTW and STWP in each insertion step. Recall that the MILPs are solved whenever the simple heuristic cannot find a time window Hence this number reflects the difficulty of the current insertion step.
  - Run time of each insertion step.
- Improvement step:
  - Reduction of the duration of the schedule: corresponds to the reduction of traveling time over all vans.
  - Number of MILPs solved for both TSPsTW and STWP.
  - Run time of each improvement step.

Note that during the insertion step all MILPs have a time limit of 15 seconds, while in the improvement step, the time limit is set to 60 seconds.

7.3 Results

We now present the results of our computational evaluation for each scenario. The key findings are additionally summarized in the following subsection. Each result table, such as for example Table 3, is

structured in the same way: For each instance size (given in the columns) we evaluate the different features per solving algorithm (given in the rows).

Scenario I: Uniform Time Window Preferences. Scenario I is the standard problem setup where customers have uniformly distributed preferences over the time windows. We summarize the results for Scenario I in Tables 3 and 4 and make the following observations, starting with the insertion heuristics.

Insertion, Scenario I	100 customers 150 capacity 5 time windows			200 customers 300 capacity 10 time windows			300 customers 450 capacity 15 time windows		
Tours:	5	6	7	5	6	7	5	6	7
Average runtime (sec:ms)									
Simple-insertion	1	1	1	1	1	1	1	1	1
TSPsTW-insertion	31	5	1	163	27	1	674	81	2
TSPsTW+STWP-insertion	7:179	904	1	14:929	2:066	1	24:993	8:421	63
Average number of time windows offered									
Simple-insertion	4.45	4.91	5.00	8.69	9.80	10.00	13.10	14.67	14.99
TSPsTW-insertion	4.50	4.92	5.00	8.77	9.83	10.00	13.21	14.71	15.00
TSPsTW+STWP-insertion	4.55	4.93	5.00	9.05	9.88	10.00	13.51	14.79	15.00
Average improvement over simple-insertion (%)									
TSPsTW-insertion	0.89	0.37	0.00	0.93	0.30	0.00	0.86	0.29	0.01
TSPsTW+STWP-insertion	2.06	0.53	0.00	4.09	0.88	0.00	3.17	0.80	0.02
Average number of MILPs solved									
TSPsTW	2.65	0.51	0.00	6.41	1.16	0.00	9.27	1.88	0.04
STWP	0.50	0.08	0.00	1.23	0.18	0.00	1.79	0.29	0.01
Customers inserted									
Total	93.2	100	100	182.8	199.4	200	273	298	300

Table 3: Results for the insertion steps of Scenario I.

First, in Table 3 we see that simple-insertion has the lowest runtimes, in fact it takes less than one millisecond for all instances considered in our computational study. Furthermore it returns a quite remarkable average number of time windows per step (between 4.45 and 14.99 time windows, depending on the instance size) which is very close to the respective overall number of time windows (5, 10 and 15).

Second, we observe that the hybrid insertion heuristics, TSPsTW-insertion and TSPsTW+STWP-insertion, on average increase the number of time windows offered per step, except for the instances with 7 tours that cannot be further improved because simple-insertion already always identifies all time windows as feasible for inserting the new customer. Further note that the runtimes of our hybrid heuristics and the corresponding average number of solved MILPs decrease with increasing number of tours, since the problem becomes easier when more tours (vans) are available.

The results for the improvement steps of Scenario I are summarized in Table 4. We compare the simple local-improvement heuristic (Local) with three hybrid improvement heuristics: Local+TSPsTW, Local+STWP+TSPsTW and Local+3STWP+TSPsTW that each execute the simple local-improvement heuristic before applying the various MILPs.

Local achieves average improvements between 0.23% and 0.95% per step and needs very little runtime (from 11ms to 259ms for the largest instance). Local+TSPsTW provides even better improvements than Local (between 0.27% and 1.13% per step) and takes about 50% additional runtime than Local alone (from

Improvement, Scenario I	100 customers 150 capacity 5 time windows			200 customers 300 capacity 10 time windows			300 customers 450 capacity 15 time windows		
Tours:	5	6	7	5	6	7	5	6	7
Average runtime (sec:ms)									
Local	11	12	12	36	44	43	259	90	90
Local+TSPsTW	25	22	20	62	64	59	321	134	125
Local+STWP+TSPsTW	1:475	1:406	1:072	932	1:206	322	1:652	1:353	561
Local+3STWP+TSPsTW	3:364	2:309	1:381	1:448	2:027	558	4:842	7:733	837
Average improvement over insertion step (%)									
Local	0.95	0.83	0.81	0.44	0.38	0.32	0.31	0.27	0.23
Local+TSPsTW	1.13	0.93	0.94	0.56	0.48	0.41	0.37	0.32	0.27
Local+STWP+TSPsTW	1.38	1.15	1.18	0.76	0.65	0.55	0.53	0.42	0.36
Local+3STWP+TSPsTW	1.54	1.38	1.29	0.88	0.76	0.66	0.63	0.52	0.45
Average number of TSPsTW MILPs solved									
Local+TSPsTW	1.49	1.43	1.44	1.50	1.39	1.36	1.48	1.39	1.37
Local+STWP+TSPsTW	5.00	6.00	7.00	5.00	6.00	7.00	5.00	6.00	7.00
Local+3STWP+TSPsTW	5.00	6.00	7.00	5.00	6.00	7.00	5.00	6.00	7.00
Average number of STWP MILPs solved									
Local+STWP+TSPsTW	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Local+3STWP+TSPsTW	2.58	2.58	2.57	2.84	2.81	2.80	2.86	2.86	2.86

Table 4: Results for the improvement steps of Scenario I.

20ms to 321ms). Local+STWP+TSPsTW is able to further improve the schedule (between 0.36% and 1.38% per step), but also requires clearly more runtime than Local+TSPsTW (from 322ms to 1sec650ms). Finally, Local+3STWP+TSPsTW provides the strongest improvements (between 0.45% and 1.54% per step), but also requires the most runtime (from 837ms to 7sec733ms). Note that the average improvement per step of the schedule's duration is quite remarkable as the insertion of a new customer in average increases the schedule's duration by less than 1% and schedule without the new customer has also been optimized.

In summary, the results for Scenario I are very satisfactory. First, all our approaches comply to the tight runtime restrictions that are required in our real-world on-line shopping application. Second, all heuristics provide excellent results: for the insertion step, the heuristics provide almost the whole set of time windows, and for the improvement step, all heuristics are able to substantially reduce the schedule's duration, where the hybrid heuristics achieve a larger improvement for a reasonable investment of runtime.

Scenario II: Varying Tour Start and End Times. In Scenario II we consider instances where tours start and end at different times in order to test the tour assignment constraints (7l) – (7o). We summarize the results for the insertion steps in Table 5 and for the improvement steps in Table 6.

When comparing results for Scenario I and Scenario II, we notice that the tour assignment constraints do not substantially impact the performance of the heuristics. More specifically, for the insertion step, we see that the ratio between the average runtime and the number of solved MILPs is very similar to the ratio in Scenario I with 300 customers (Table 3). The increase of the average number of solved MILPs indicates that the instances of Scenario II are harder than the instances of Scenario I since there are more time windows to consider. For the improvement step, when comparing runtimes of Scenario I with 300

customers (Table 4) to Scenario II, we notice that they are almost the same.

Insertion, Scenario II	250 customers 375 capacity		
Time windows:	13	14	15
Tours:	5	6	7
Average runtime (min:sec:ms)			
Simple-insertion	1	1	1
TSPsTW-insertion	439	279	133
TSPsTW+STWP-insertion	2:05:424	22:190	6:379
Average number of time windows offered			
Simple-insertion	9.14	11.97	14.15
TSPsTW-insertion	9.28	12.11	14.24
TSPsTW+STWP-insertion	9.55	12.33	14.38
Average improvement over simple-insertion (%)			
TSPsTW-insertion	1.52	1.14	0.62
TSPsTW+STWP-insertion	4.50	2.95	1.64
Average number of MILPs solved			
TSPsTW	14.99	8.04	3.01
STWP	3.71	1.88	0.76
Customers inserted			
Total	185	223	242.6

Table 5: Results for the insertion steps of Scenario II.

Scenario III: Large Instances. In Scenario III we examine the performance of our approaches on large instances with 1000 customers. We consider two setups: one setup with *many short* tours and few time windows, and another setup with *fewer long* tours and many time windows. For this scenario we assess all our methods except for the STWP-based improvement heuristic, which is not fast enough on these large instances to comply with our tight runtime restrictions.

The results for Scenario III are summarized in Table 7 for the insertion step, and in Table 8 for the improvement step. The first setup with many short tours corresponds to the left column, and the second setup with few long tours corresponds to the right column.

First, we observe that the runtimes for both the insertion step and the improvement step are very low, even for these large instances, which demonstrates that our solving approaches scale very well. Furthermore, we see a similar improvement of the hybrid heuristics over the simple heuristics as for the smaller instances of Scenario I and Scenario II.

When comparing the two different setups, we notice that the average runtimes for both insertion and improvement step are higher in the second setup with fewer long tours. The reason for this lies in the larger number of customers and time windows per tour which makes these instances more difficult to solve. We also observe that the improvement achieved by the hybrid heuristics compared to the simple heuristics is similar in both setups.

In summary all methods considered perform impressively for both setups as they are able to produce high-quality schedules within our tight runtime restrictions on these large instances.

Scenario IV: Improvement step only every *i*th iteration. In Scenario IV, we evaluate our algorithms on instances that have recurring periods during which many customers send requests. To accommodate

Improvement, Scenario II	250 customers 375 capacity		
Time windows:	13	14	15
Tours:	5	6	7
Average runtime (sec:ms)			
Local	33	105	200
Local+TSPsTW	62	173	259
Local+STWP+TSPsTW	1:558	1:783	2:436
Local+3STWP+TSPsTW	3:779	3:780	5:107
Average improvement over insertion step (%)			
Local	0.55	0.43	0.36
Local+TSPsTW	0.64	0.51	0.42
Local+STWP+TSPsTW	0.75	0.58	0.49
Local+3STWP+TSPsTW	0.81	0.63	0.53
Average number of TSPsTW MILPs solved			
Local+TSPsTW	1.50	1.46	1.47
Local+STWP+TSPsTW	4.50	5.17	5.84
Local+3STWP+TSPsTW	4.78	5.55	6.32
Average number of STWP MILPs solved			
Local+STWP+TSPsTW	1.00	1.00	1.00
Local+3STWP+TSPsTW	2.93	2.94	2.96

Table 6: Results for the improvement steps of Scenario II.

these periods of high request frequency, we run the improvement step only after each ith successful insertion step, instead of after each insertion step. Furthermore, the instances in Scenario IV are the largest we consider, with 2000 customers in total.

Tables 9 and 10 summarize the results for Scenario IV, for the insertion step and the improvement step, respectively. Each column shows results for different is. First, we notice that the runtime of both the insertion and the improvement step increases with i: the more often we skip an improvement step, the longer it takes to insert a new customer or to improve the schedule's duration.

For the insertion step, we notice a small decline in the average number of offered time windows with increasing i. Furthermore, we also see that we can schedule fewer customers with increasing i. This is an expected result, since a decrease in improvement steps is expected to reduce the quality of the schedule. However, the average number of inserted customers is still very high (between 1971.6 and 1954.6 out of 2000), which demonstrates that we pay little for omitting the (expensive) improvement step during periods with high customer request frequency.

For the improvement step, we observe an increase in achieved improvement per step with increasing i. This is an expected, but also promising result: the less often we improve the solution, the more we can improve in each step. In other words: the improvement that we omit after an insertion step can be made up for in a future improvement step (up to a certain extent). It demonstrates again that performing the improvement step only every ith step is a viable option during periods with high customer request frequency.

In summary, the results show that skipping the improvement step allows us to deal with temporarily high customer request rates, even for very large schedules. Furthermore, we see that applying the improvement step less often leads to an increased improvement per step at the cost of longer run times.

Insertion, Scenario III	1000 customers 150 capacity 5 time windows			1000 customers 450 capacity 15 time windows		
Tours:	45	50	55	18	20	23
Average runtime (sec:ms)						
simple-insertion	1	1	1	1	1	1
TSPsTW-insertion	184	73	8	1:061	479	17
Average number of time windows offered						
Simple-insertion	4.29	4.75	4.97	12.91	14.18	14.97
TSPsTW-insertion	4.33	4.76	4.97	13.08	14.29	14.98
Average improvement over simple-insertion (%)						
TSPsTW-insertion	0.81	0.34	0.04	1.31	0.76	0.04
Average number of MILPs solved						
TSPsTW	24.59	10.08	1.13	36.28	15.39	0.61
Customers inserted						
Total	865.2	952.8	995	877.2	953.4	998.8

Table 7: Results for the insertion steps of Scenario III.

Improvement, Scenario III	1000 customers 150 capacity 5 time windows			1000 customers 450 capacity 15 time windows		
Tours:	45	50	55	18	20	23
Average runtime (sec:ms)						
Local	2:110	2:824	2:373	709	931	1:133
Local+TSPsTW	2:167	2:875	3:018	784	1:016	1:226
Average improvement over insertion step (%)						
Local	0.70	0.68	0.68	0.27	0.27	0.27
Local+TSPsTW	0.74	0.71	0.70	0.30	0.30	0.30
Average number of TSPsTW MILPs solved						
Local+TSPsTW	3.96	4.15	4.29	2.21	2.33	2.45

Table 8: Results for the improvement steps of Scenario III.

This characteristic should be considered when customizing the approach to the requirements of specific suppliers.

7.4 Summary of Key Findings

We now summarize the main findings of our computational study that hold true for all our experiments.

- All our approaches use very little runtime and provide very good results with respect to solution quality.

Insertion, Scenario IV	2000 customers 5 time windows		150 capacity 120 tours
i:	10	25	50
Average runtime (ms)			
Simple-insertion	1	1	1
TSPsTW-insertion	24	31	44
Average number of time windows offered			
Simple-insertion	4.92	4.91	4.88
TSPsTW-insertion	4.93	4.92	4.89
Average improvement over simple-insertion (%)			
TSPsTW-insertion	0.15	0.16	0.26
Average number of MILPs solved			
TSPsTW	4.03	5.04	7.38
Customers inserted			
Total	1971.6	1966.8	1954.6

Table 9: Results for the insertion steps of Scenario IV.

Improvement, Scenario IV	2000 customers 5 time windows		150 capacity 120 tours
i:	10	25	50
Average runtime (min:sec:ms)			
Local	1:43:333	3:16:111	5:07:762
Local+TSPsTW	1:43:650	3:16:614	5:08:393
Average improvement over insertion step (%)			
Local	2.93	5.62	8.97
Local+TSPsTW	3.02	5.78	9.15
Average number of TSPsTW MILPs solved			
Local+TSPsTW	28.54	39.60	51.42

Table 10: Results for the improvement steps of Scenario IV.

- The simple insertion heuristic is able to determine feasible insertion positions for most time windows in remarkably little time
- Although computationally cheap, the local improvement heuristic proves to be very effective and in average achieves significant reductions of the duration of the schedule during each call.
- Both our MILP-based heuristics yield further quality improvements for both the insertion and the improvement step. As most feasible time windows for easy instances are found by the simple insertion heuristic, the impact of our MILPs is more pronounced for instances with less tours. We emphasize in particlar that:
  - The TSPsTW can be solved to optimality extremely fast on all instances. Especially during

the improvement step on large instances, the runtime for solving the TSPsTW MILPs becomes negligible compared to the corresponding runtime of our local improvement heuristic. Furthermore, the TSPsTW MILPs are always solved to optimality before reaching the time limit.

  - The STWP MILP is very effective and fast enough for moderately sized instances with up to 300 customers.

- The average reduction of the duration of the schedule per step is seemingly small (around 1%), but nonetheless quite remarkable because:
  - We use an improvement step after each insertion step in Scenarios I - III. When calling the improvement step only every *i*th iteration (as in Scenario IV), the average reduction of the duration of the schedule per step increases dramatically.
  - In average the duration of the schedule is increased by less than 1 % when a new customer is inserted, where the impact of an insertion on the schedule's duration decreases for growing instance size. Accordingly we observe that the average improvement per step is smaller for instances with a large number of customers.
- Our suggested algorithmic strategies for both insertion and improvement step comply with the respective requirements of the different scenarios and corresponding instance sizes considered, which allows a broad field of application.

8 Conclusion

In this paper we presented the capacitated Vehicle Routing Problem with structured Time Windows (cVRPsTW) that arises in the context of delivering goods, where customers choose the delivery time window, and the delivery schedule is updated as new customers arrive. The special feature of the cVRPsTW is the *structure* of the time windows that we exploit in our suggested solving approaches. We considered an on-line variant of the problem with very strict runtime limitations that arises in online grocery shopping.

We introduced a two-step approach, consisting of an insertion and an improvement step. For the insertion step, we proposed a simple insertion heuristic, as well as two hybrid heuristics that each employ a specialized mixed-integer linear program that solves a subproblem of the cVRPsTW to optimality. Similarly, for the improvement step, we introduce a simple local improvement heuristic and three hybrid heuristics that employ mixed-integer linear programs closely related to the ones for the insertion step.

In our computational evaluation we assess our approaches on four different scenarios to reflect various setups and to evaluate different features of the solving approaches. The results demonstrate that our approaches comply with very strict time limits and produce very good results within seconds, rendering them applicable to a real-world setting.

In summary, the contributions of this paper are as follows. First, we introduce a set of novel heuristics for solving the on-line variant of the cVRPsTW that exploit the special structure of the time windows and provide very good results in a very short time, also on very large problem instances, thus complying with the strict requirements of real world applications. Second, we conduct an extensive computational study on a large set of benchmark instances with different size (ranging from 100 customers to 2000 customers) and several features, such as variations in customer request frequency and the shift patterns of the van tours.

References

N. Agatz, A. M. Campbell, M. Fleischmann, and M. W. P. Savelsbergh. Challenges and opportunities in attended home delivery. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 379–396. Springer US, 2008.

N. Agatz, A. M. Campbell, M. Fleischmann, and M. W. P. Savelsbergh. Time Slot Management in Attended Home Delivery. *Transportation Science*, 45(3):435–449, 2011.

N. Azi, M. Gendreau, and J.-Y. Potvin. An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, 202:756–763, 2010.

R. Baldacci, A. Mingozzi, and R. Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1): 1–6, 2012.

J. F. Bard, G. Kontoravdis, and G. Yu. A Branch-and-Cut Procedure for the Vehicle Routing Problem with Time Windows. *Transportation Science*, 36(2):250–269, 2002.

R. W. Bent and P. Van Hentenryck. Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers. *Operations Research*, 52(6):977–987, 2004.

H.-J. Böckenhauer, J. Kneis, and J. Kupke. Approximation hardness of deadline-TSP reoptimization. *Theoretical Computer Science*, 410(21–23):2241–2249, 2009.

A. M. Campbell and M. W. P. Savelsbergh. Decision Support for Consumer Direct Grocery Initiatives. *Transportation Science*, 39(3):313–327, 2005.

N. A. El-Sherbeny. Vehicle routing with time windows: An overview of exact heuristic and metaheuristic methods. *Journal of King Saud University*, 22:123–131, 2010.

Eurostat. E-commerce statistics, November 2015. URL `http://ec.europa.eu/eurostat/statistics-explained/index.php/E-commerce_statistics`.

M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows. *Operations Research*, 46(3):330–335, 1998.

L. Hong. An improved LNS algorithm for real time vehicle routing problem with time windows. *Computers and Operations Research*, 39:151–163, 2012.

P. Hungerländer, K. Maier, J. Pöcher, A. Rendl, and C. Truden. The Traveling Salesman Problem with structured Time Windows, 2017. Forthcoming paper.

G. Ioannou, M. Kritikos, and G. Prastacos. A Greedy Look-Ahead Heuristic for the Vehicle Routing Problem with Time Windows. *The Journal of the Operational Research Society*, 52(5):523–537, 2001.

N. Kohl, J. Desrosiers, O. B. Madsen, M. M. Solomon, and F. Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116, 1999.

R. Macedo, C. Alves, J. M. V. de Carvalho, F. Clautiaux, and S. Hanafi. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo polynomial model. *European Journal of Operational Research*, 214:536–545, 2011.

O. Madsen and N. Kohl. An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation. *Operations Research*, 45(3):395–406, 1997.

V. Pillac, M. Gendreau, C. Gueret, and A. L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225:1–11, 2013.

U. Ritzinger, J. Puchinger, and R. F. Hartl. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231, 2016.

P. Toth and D. Vigo, editors. *The Vehicle Routing Problem: Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics, 2nd edition, 2014.

X. Yang, A. K. Strauss, C. S. M. Currie, and R. Eglese. Choice-Based Demand Management and Vehicle Routing in E-Fulfillment. *Transportation Science*, 50(2):473–488, 2016.