On the Slot Optimization Problem in On-Line Vehicle Routing

P. Hungerländer*, A. Rendl† and C. Truden‡

13th April 2017

Abstract

The capacitated vehicle routing problem with time windows (cVRPTW) is concerned with finding optimal tours for vehicles that deliver goods to customers within a specific time slot (or time window), respecting the maximal capacity of each vehicle. The on-line variant of the cVRPTW arises for instance in online shopping services of supermarket chains: customers choose a delivery time slot for their order online, and the fleet’s tours are updated accordingly in real time, where the vehicles’ tours are incrementally filled with orders.

In this paper, we consider a challenge arising in the on-line cVRPTW that has not been considered in detail in the literature so far. When placing a new order, the customer receives a selection of available time slots that depends on the customer’s address and the current (optimized) schedule. The customer chooses a preferred time slot, and the order is scheduled. The larger the selection, the more likely the customer finds a suitable time slot, leading to higher customer satisfaction and a higher overall number of orders placed. We denote the problem of determining the maximal number of feasible time slots for a new customer order as the Slot Optimization Problem (SOP).

We formally define the SOP and propose an adaptive neighbourhood search heuristic for determining feasible slots for inserting a new customer order based on a given delivery schedule in real time. Our approach is tailored to the SOP and combines local search techniques with strategies to overcome local minima. In an experimental evaluation, we demonstrate the efficiency of our approach on a variety of benchmark sets.

Keywords: Vehicle routing problem; time windows; online optimization; time slot management; adaptive neighborhood search.

1 Introduction

In this paper, we discuss a problem that arises in the context of delivering goods or performing services to customers at their home, where the customers need to be present. The arrival time of the service at the customer is typically not known in advance. In the past, the customer would have to reserve the whole (or large parts of the) day to await the service. Nowadays, to increase customer satisfaction, companies pre-arrange a time slot (or time window) with the customer, during which the service is guaranteed to arrive.

Providing time slots bears a great advantage to the customers who can better manage their time by choosing an appropriate slot. However, to the service and delivery companies, this poses a challenge: their original problem, a capacitated Vehicle Routing Problem (cVRP), becomes a capacitated Vehicle Routing Problem with Time Windows (VRPTW). Furthermore, if customers are not offered a desirable time slot, approximately 25% of them refuse the service. Therefore, it is desirable for the service providers to schedule their fleet in such a way that maximizes slot availability for customers.

In this paper, we focus on the context of an online grocery shopping service, where customers place orders online: first, the system offers the customer a set of available time slots for delivery, then the customer chooses a
time slot, and finally, the order is inserted into the fleet’s schedule at the chosen time slot. The problem of stepwise (customer by customer) building an optimized schedule in real-time is referred to as the on-line capacitated Vehicle Routing Problem with structured Time Windows (cVRPsTW) (Hungerländer et al. 2016). An important feature of the cVRPsTW is that the time windows have a special structure: they are non-overlapping and can hold several customers.

We investigate the first step of the online cVRPsTW: maximizing the number of time slots offered to customers. Thus, we introduce a new combinatorial optimization problem, the so-called Slot Optimization Problem (SOP), that, given an existing, incomplete schedule and a customer order, is concerned with finding the maximal number of time slots into which the order can be inserted, without moving any of the existing orders into other time slots. Note that in this paper we assume that time slots are non-overlapping and can hold several customers, as with the cVRPsTW.

If we choose one slot for the new customer order, then the SOP is equivalent to the feasibility version of an appropriate cVRPTW instance. As the cVRPTW is strongly NP-hard, see e.g. Madsen and Kohl (1997), the SOP is also strongly NP-hard and consists of $q$ feasibility problems that are all strongly NP-complete.

This work stems from a collaboration with one of the world’s largest supermarket chains, and in the corresponding real-world application, time is an important factor: time slots must be offered within milliseconds to satisfy the customers’ satisfaction. This poses an additional challenge to solving this problem.

We discuss several heuristics for determining feasible slots for inserting a new customer based on a given delivery schedule in real time. These approaches combine local search techniques with strategies to overcome local minima and integer linear programs for selected sub-problems. In an experimental evaluation, we demonstrate the efficiency of our approaches on a variety of benchmark sets.

In summary the main contributions of this paper are:

1. We introduce a new NP-hard combinatorial optimization problem that is motivated by a real-world application.
2. We propose an adaptive neighbourhood search (ANS) heuristic that is tailored to the SOP and combines local search techniques with strategies to overcome local minima.
3. We compare various heuristic methods for solving the SOP. Additionally to our newly proposed ANS, we consider approaches suggested in our recent paper (Hungerländer et al. 2016) that can also be applied to the SOP. In particular all 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.
4. We provide an extensive computational study, illustrating that our approaches perform very well in practice.

2 Related Work


The home delivery problem (HDP), introduced by Campbell and Savelsbergh (2005), considers a very closely-related problem, where delivery requests arrive dynamically, and the system has to decide two things: first, if a new request is accepted or not, and second, in which time slot 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 slots, and the customers decide upon acceptance, while in the HDP, the system makes these decisions.

Campbell and Savelsbergh (2005) propose a two-step insertion heuristic for the HDP. The first step is concerned with re-constructing (several versions of) the schedule from scratch using a greedy randomized adaptive search procedure (GRASP). In the second step, a heuristic evaluates if the new order can be inserted into one of the newly constructed schedules, by approximating the expected profit of accepting an incoming request on four insertion criteria. If there exist several schedules where the new request can be inserted, the one with the highest profit (and its corresponding time slot) is selected. These heuristic together with the four criteria are evaluated in the experimental section, where 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 (up to 2000 customers).
introduce the time slot management problem (TSMP) in the HDP which deals with selecting time slots to minimize expected delivery costs while meeting the service requirements. More specifically, given service requirements and average weekly demands for each zip code in the delivery region, determine the set of time slots to offer in 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. The difference to our work is that [Agatz et al. (2011)] are concerned with the strategic question of which time slots to offer in general, while our paper deals with the operational question of which time slots to offer to a specific customer for a specific schedule.

[Agatz et al. (2008)] present issues and solving approaches for the attended HDP, where customers select the time slot 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 they consider 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. In summary, Agatz et al. (2008) give an exhaustive review around closing slots and corresponding pricing strategies in order to ensure dense customer clusters. But they do not consider the SOP and algorithmic approaches to it.

In [Hungerländer et al. (2016)] we present an insertion heuristic and a mixed-integer linear program for solving the cVRPsTW of which the SOP is a subproblem. In this paper, we suggest an adaptive neighbourhood search (ANS) tailored to the SOP, and show that it compares favourably with the approaches from our previous work.

### 3 Problem Description

The Slot Optimization Problem (SOP) arises when delivering services at customers who choose the service time slot. Given an existing schedule for a fleet of vehicles, and a new customer order, the aim of the SOP is to find time slots during which the given order can be serviced by at least one vehicle of the fleet, while assuring that all other scheduled orders stay within their assigned time slot.

In this paper we assume the following two characteristics concerning the structure of time slots. First, several customers fit into a single time slot. Second, time slots are non-overlapping. We exploit these two characteristics that hold true in our considered application in our solving approaches for the SOP. To simplify the exposition we additionally assume that the time slots are equidistant and consecutive.

As input parameters we are given:

1. A set of time slots \( \mathcal{W} = \{w_1, \ldots, w_q\} \), where each time slot \( w \in \mathcal{W} \) is defined through its start time \( s_w \) and its end time \( e_w \).
2. A schedule \( \mathcal{S} = \{s_1, s_2, \ldots\} \), consisting of \( |\mathcal{S}| = m \) tours with assigned capacities \( C_i, i \in \mathcal{I} \), that correspond to the capacity of the van that operates tour \( i \).
3. A set of customers \( \mathcal{C} \), with corresponding order weight function \( c : \mathcal{C} \rightarrow \mathbb{R}^+ \), service time function \( s : \mathcal{C} \rightarrow \mathbb{R}^+ \), and travel time function \( t : \mathcal{C} \times \mathcal{C} \rightarrow \mathbb{R}^+ \), where we set the travel time from a customer \( a \) to itself to 0, i.e. \( t(a, a) = 0, a \in \mathcal{C} \).
4. A new customer order \( \tilde{a} \) for which we want to calculate feasible time slots for insertion in schedule \( \mathcal{S} \).

The customer-related definitions from above can be summarized by considering a weighted graph \( G = (V = \mathcal{C}, E = \mathcal{C} \times \mathcal{C}) \) 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 \( u, v, w \in \mathcal{C} \). However note that all presented solving approaches work with arbitrary asymmetric travel time functions.

The function \( w : \mathcal{C} \rightarrow \mathcal{W} \) assigns a time slot to each customer during which the delivery van has to arrive at the customer. We speak of structured time slots, if the number of customers \( |\mathcal{C}| = p \) is much larger than the number of time slots \( |\mathcal{W}| = q \), i.e. \( p \gg q \), and therefore typically several customers are assigned to the same time slot. Further we assume all structured time slots to be non-overlapping, i.e. \( e_u \leq s_{u+1}, w \in [q-1] \), holds, where the set \( [u], u \in \mathbb{N}, \) contains the elements \( \{1, 2, \ldots, u-1, u\} \). For simplicity of exposition we set \( e_u = s_{u+1}, w \in [q-1] \), but note that all our models and algorithms considered can also be applied to the case \( e_u < s_{u+1}, w \in [q-1] \), with gaps between the structured time slots.

A tour \( s_\mathcal{A} = \{a_1, a_2, \ldots, a_n\} \) contains \( n \) customers, where the indices of the customers display the sequence in which the customers are visited. Furthermore each tour \( s_\mathcal{A} \) has assigned start and end times that we denote as...
start, and end respectively. Hence the van executing tour \( \alpha \) can leave no earlier than start, from the depot and must return to the depot no later than end. The start depot \( a_0 \) and end depot \( a_{n+1} \) are formally assigned to time slots \( w_0 \) respectively \( w_{n+1} \) with start time 0 and the end time set to \( \infty \). Without loss of generality we set the order weights and service times of the depots to zero: \( c(a_0) = c(a_{n+1}) = s(a_0) = s(a_{n+1}) = 0 \).

In Figure 1 we depict three time slots of a tour to provide further clarification for some of the notions.

![Diagram](image)

Figure 1: Illustration of three time slots with seven customers. The vertical bars show the start and end times of the respective time slots. The length of the arrows depict the travel times between customers and the vertical lengths of the boxes illustrate the service times at the customers.

4 Adaptive Neighborhood Search

In this section we first review the basic methods for solving the SOP that were suggested in Hungerländer et al. (2016). Then we introduce several concepts that allow us to formulate an adaptive neighborhood search (ANS) tailored to the SOP. Our ANS considers two different neighborhoods:

1. operations with customer orders assigned to the current insertion slot (Neighborhood I), and
2. operations with customer orders assigned to any other time slot (Neighborhood II).

Our method switches among these two neighborhoods based on the current structure of the schedule and the search history.

Basic Methods for Solving the SOP In Hungerländer et al. (2016) we presented an insertion heuristic as well as a mixed-integer linear program (MILP) for tackling the SOP. Our simple insertion heuristic takes a new customer \( \tilde{a} \), a tour \( \alpha \) and a time slot \( w \in W \) as inputs, and tries to insert \( \tilde{a} \) into \( \alpha \) at time slot \( w \). 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 in the given schedule. Therefore it runs in linear time and is extremely fast in practice.

We iteratively apply the simple insertion heuristic to all time slots \( w \in W \) and all tours \( \alpha \) to calculate the set of time slots for feasibly inserting the new customer \( \tilde{a} \). As soon as we find a tour for which the feasible insertion of \( \tilde{a} \) is possible, we add \( w \) to our time slot set and do not consider insertions of \( \tilde{a} \) at \( w \) for the remainder of the tours.

We also suggested a MILP for determining feasible time slots within a single tour. We denote this MILP as the Traveling Salesperson Problem with structured Time Windows (TSPsTW). For tackling the SOP we consider the feasibility version of the TSPsTW MILP. Due to the space restrictions of this conference paper we refer to Hungerländer et al. (2016) for further details on both our simple insertion heuristic and our TSPsTW MILP.

Arrival Times Next let us give a formal, recursive definition of the earliest and latest arrival time that are needed for the definition of access, exit and loss time below. We consider a fixed tour \( \alpha = \{a_0, a_1, \ldots, a_n, a_{n+1}\} \), 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 \( \alpha \). Note that our approaches do not move the depots, and customers can only be inserted after the start depot and before the end depot.

The earliest (latest) arrival time \( \alpha_{a,j} (\beta_{a,j}) \) gives the earliest (latest) time at which the van may arrive at \( a \), who is the \( j \)th customer on the tour, while not violating slot and travel time constraints on the preceding (subsequent) tour:

\[
\alpha_{a,0} = \text{start}, \quad \alpha_{a,j+1, 1} = \max \left\{ s_{a_{j+1}}, \alpha_{a, j} + s(a_j) + t(a_j, a_{j+1}) \right\}, \quad j \in [n],
\]

\[
\beta_{a, n+1, 1} = \text{end}, \quad \beta_{a,j-1, 1} = \min \left\{ e_{a_{j-1}}, \beta_{a,j} - s(a_{j-1}) - t(a_{j-1}, a_j) \right\}, \quad j \in [n].
\]
Now we can concisely define the feasibility of a tour and a schedule with the help of the earliest arrival time. A schedule \( \mathcal{S} \) is feasible, if all its tours are feasible and a tour \( \mathcal{A} \) is feasible if it satisfies both of the following conditions:

\[
s_{w[a_i]} \leq \alpha_{w[a_i]}, \quad i \in [n], \quad \sum_{i \in [n]} c(a_i) \leq C_{w[a_i]},
\]

Time Feasibility (TFEAS),

Capacity Feasibility (CFEAS).

**Insertion and Neighbourhoods** For a given tour \( \mathcal{A} = \{a_0, a_1, \ldots, a_i, \ldots, a_n, a_{n+1}\} \) and a new customer \( \tilde{a} \) we define the insertion operator \( +_i \) as

\[
\mathcal{A} +_i \tilde{a} := \{a_1, \ldots, a_i, \tilde{a}, a_{i+1}, \ldots, a_n\}, \quad i \in [n].
\]

Furthermore for a given tour \( \mathcal{A} \) we define the first resp. the last index associated with a given time slot \( w \in \mathcal{W} \) as

\[
f(w) := \max_{i \in [n]} \{i : w > a_i\} + 1, \quad l(w) := \max_{i \in [n]} \{i : w < a_i\} - 1.
\]

As we assume that each tour starts and ends at a depot both indices are always properly defined.

We consider 1-move and 1-swap operations for local improvement of a given schedule. The 1Move(\( \tilde{a}^w, \mathcal{A}, \mathcal{B} \)) operation removes customer \( \tilde{a} \) from tour \( \mathcal{A} \) and tries to feasibly insert it into a different tour \( \mathcal{B} \) during time slot \( w \). The 1Swap(\( \tilde{a}^w, \mathcal{A}, \mathcal{B} \)) operation exchanges customer \( \tilde{a}^w \) with another customer at time slot \( w \) from a different tour \( \mathcal{B} \). In our ANS we apply 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 if we reach a local minimum of our current objective function with respect to our current neighborhood.

**Access, Exit, Loss and Free Time** The following concepts aim for a quantification of the interdependencies between time slots. Based on these concepts, the ANS decides whether to perform local improvement operations inside or outside of the time slot assigned to the new customer who has to be inserted.

For a tour \( \mathcal{A} \) and a new customer \( \tilde{a}^w \) who is inserted into time slot \( w \), we define the access and exit time of \( w \) as

\[
\chi^-_{w}(\mathcal{A}, \tilde{a}^w) := \max \{\alpha_{w[f(w)]}, \alpha_{w, f(w)}\} - s_w, \quad \chi^+_{w}(\mathcal{A}, \tilde{a}^w) := e_w - \min \{\beta_{a_{l(w)}}, f(w) + 1, \beta_{a_{l(w)}}, l(w) + 1\}.
\]

The access time \( \chi^-_{w} \) corresponds to the amount of time that is "lost" at the beginning of time slot \( w \), caused by service time or travel time from the last customer order of the previous time slot \( w - 1 \). Similarly, the exit time \( \chi^+_{w} \) corresponds to the loss of time at the end of the time slot \( w \), due to travel time to the first order in the next time slot \( w + 1 \). Figure 2 illustrates both the access and exit time. We further denote \( \chi_w(\mathcal{A}, \tilde{a}^w) := \chi^-_{w}(\mathcal{A}, \tilde{a}^w) + \chi^+_{w}(\mathcal{A}, \tilde{a}^w) \) as the loss time of time slot \( w \). If \( \chi_w(\mathcal{A}, \tilde{a}^w) = 0 \), then a violation of TFEAS for \( \tilde{a}^w \) can only be repaired by removing customers assigned to \( w \) from tour \( \mathcal{A} \). Hence in this case our ANS only considers local improvement operations with customers assigned to \( w \), i.e. 1-move/2-swap operations within Neighborhood \( I \).

We also want to quantify the amount of service and travel time that is needed for inserting \( \tilde{a}^w \) during slot \( w \). For a given tour \( \mathcal{A} \) we define the free time of time slot \( w \) as

\[
\lambda_w(\mathcal{A}) := \left( e_w - s_w \right) - \sum_{i = f(w)}^{l(w) - 1} \left( s(a_i) + t(a_i, a_{i+1}) \right),
\]

where (I) gives the length of \( w \) and (II) indicates the amount of service and travel time that must be handled within \( w \). For the insertion of customer \( \tilde{a}^w \) at a certain position \( i \in \{f(w) - 1, \ldots, l(w)\} \), the corresponding additional amount of service and travel time within time slot \( w \) can be calculated by \( \lambda_w(\mathcal{A} +_i \tilde{a}^w) - \lambda_w(\mathcal{A}) \). For illustrations of access, exit, loss and free time we refer to Figures 2 and 3.
Hungerländer, Rendl, Truden

On the Slot Optimization Problem in On-Line Vehicle Routing

\[ a - f(w) - 1 + a - f(w) + 1 + a - l(w) - 1, \]

\[ \chi_{w}(\mathcal{A}, \tilde{a}^w) \]

Figure 2: Illustration of the loss time \( \chi_{w}(\mathcal{A}, \tilde{a}^w) \) in case the new customer \( \tilde{a}^w \) is inserted between two other customers assigned to time slot \( w \).

\[ \lambda_w(\mathcal{A}) \]

Figure 3: Illustration of free time \( \lambda_w(\mathcal{A}) \) for a toy example with a single time slot.

**Feasibility and Infeasibility Conditions**

Now we can give a necessary infeasibility condition for the insertion of a new customer. The insertion of \( \tilde{a}^w \) into \( \mathcal{A} \) is infeasible, if the following inequality holds:

\[
\max_{i \in \{f(w) - 1, \ldots, l(w)\}} \lambda_w(\mathcal{A} + i \tilde{a}^w) < 0. \tag{1}
\]

Note that the above condition solely depends on the customers assigned to time slot \( w \). Similarly the insertion of \( \tilde{a}^w \) into \( \mathcal{A} \) is feasible for at least one insertion position, if the following inequality holds:

\[
\max_{i \in \{f(w) - 1, \ldots, l(w)\}} \lambda_w(\mathcal{A} + i \tilde{a}^w) - \chi_{w}(\mathcal{A}, \tilde{a}^w) \geq 0. \tag{2}
\]

**Description of our Adaptive Neighborhood Search (ANS)**

Finally we can now concisely describe the workings our ANS for the SOP. First we ensure that the current tour fulfills CFEAS after the insertion of the new customer order. Next in Step 2 we try to increase the free time through 1-move/2-swap operations within Neighbohood I until the infeasibility condition (1) does not hold anymore. If we reach a local minimum and inequality (1) is still satisfied, then we terminate the algorithm because the following steps will not be able to facilitate an insertion of the new customer order. Step 3 is concerned with reducing the loss time through local improvement operations within Neighbohood II until either the new customer order can be inserted into the tour or the loss time is equal to zero. Finally in Step 4 we try to further increase the free time through 1-move/2-swap operations within Neighborhood I until either the insertion of the new customer order is possible or a local minimum is reached. For further details we refer to Algorithm 1.

5 Computational Experiments

In this section we evaluate the proposed ANS approach and compare it with the heuristics from our previous work in [Hungerländer et al., 2016].

**Instances**

Our benchmark instances are based on the same benchmarks as in [Hungerländer et al., 2016] that reflect real-world problems as they arise in online shopping, regarding travel times, length of time slots, duration of service times, customer order weights and their proportions to van capacities. In order to compare the performance of the various methods for solving the SOP we construct benchmark instances which consist of

- a feasible schedule which contains \( p \) customers, and
- a new customer which has no time slot assigned yet.
Algorithm 1 Adaptive Neighborhood Search (ANS) for the SOP

1: Step 1: Check capacity feasibility
2: while $\sum_{i \in [n]} c(a_i) + c(\tilde{a}^w) > C_{\text{opt}}$ do
3:   Reduce $\sum_{i \in [n]} c(a_i)$ by applying 1-move/2-swap operations to customers $\{a_i \in \mathcal{A} \}$.
4: end for
5: Terminate and return false if capacity feasibility can not be achieved.
6: Step 2: Increase free time
7: while Condition (1) holds true do
8:   Increase $\lambda_w(\mathcal{A})$ by applying 1-move/2-swap operations to customers $\{a_i \in \mathcal{A} : w[a_i] = w\}$.
9: end for
10: Terminate and return false if Condition (1) is still satisfied.
11: Step 3: Reduce loss time
12: while insertion of $\tilde{a}^w$ is not TFEAS and $\chi_w(\mathcal{A}, \tilde{a}^w) > 0$ do
13:   Reduce $\chi_w(\mathcal{A}, \tilde{a}^w)$ by applying 1-move/2-swap operations to customers $\{a_i \in \mathcal{A} : w[a_i] \neq w\}$.
14: end for
15: Step 4: Further increase free time
16: while insertion of $\tilde{a}^w$ is not TFEAS do
17:   Increase $\lambda_w(\mathcal{A})$ by applying 1-move/2-swap operations to customers $\{a_i \in \mathcal{A} : w[a_i] = w\}$.
18: end for
19: Return true if feasible insertion can be found, otherwise return false.

The instances are generated by iteratively inserting customers into the schedule using the simple insertion heuristic from our previous paper. We distinguish two scenarios. In the first scenario we perform no optimization between the insertion steps. In the second scenario, the schedule is re-optimized after each customer insertion. Thus, the schedule in the second scenario is better than that of the first scenario, and therefore we refer to the second scenario as “optimized schedule”. All instances can be downloaded from [http://tinyurl.com/vrpstw](http://tinyurl.com/vrpstw).

Setup For each generated instance we solve the SOP using the simple insertion heuristic and the TSPsTW feasibility problem from our previous work, and the newly proposed adaptive neighborhood search (ANS). For each approach we record the number of slots that can be offered to the new customer $\tilde{a}^w$, as well as the corresponding run time required. We can use the simple insertion heuristic as a basic reference to compare against because due to their construction the two other methods determine at least the feasible time slots that are found by the simple insertion heuristic.

We consider medium-sized SOP instances with up to 980 customers at fill level 100% and large instances with up to 1834 customers at fill level 100%. In Table 1 we report the number of feasible time slots found by each method and corresponding run time required averaged over 100 instances for our two scenarios and two different fill levels. 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 ILP-solver. All computations are run in single thread mode.

Interpretation of results At fill level 85% the instances are still rather easy and hence already the simple insertion heuristic determines nearly all feasible time slots for inserting the new customer. All remaining time slots are identified as feasible by the ANS within only a few milliseconds. While the TSPsTW feasibility problem is able to determine more time slots than the simple insertion heuristic, it is outperformed both with respect to time slots found and run time required by the ANS.

The same overall relation between the methods holds true at fill level 95%, only that the differences between the methods are much more pronounced, especially for optimized schedules. Here the simple insertion heuristic is
only able to determine around half of the feasible insertion time slots that are found by ANS. Again the TSPsTW feasibility problem is clearly inferior to ANS both with respect to time slots found and run time required. In summary ANS is the clearly best method for the SOP concerning solution quality. Additionally even for large-scale instances, the ANS adheres to very strict run time limitations that apply for the insertion of new customers in our online grocery shopping application.

<table>
<thead>
<tr>
<th>Summarized results of SOP benchmark experiments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance size</td>
</tr>
<tr>
<td>Schedule optimized</td>
</tr>
<tr>
<td># customers at fill level 100%</td>
</tr>
<tr>
<td>Fill level</td>
</tr>
<tr>
<td>85%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Average run time (sec:ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple insertion heuristic</td>
</tr>
<tr>
<td>TSPsTW feasibility problem</td>
</tr>
<tr>
<td>ANS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Average number of feasible time windows found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple insertion heuristic</td>
</tr>
<tr>
<td>TSPsTW feasibility problem</td>
</tr>
<tr>
<td>ANS</td>
</tr>
<tr>
<td>All findings combined</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Average improvement over simple-insertion (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSPsTW feasibility problem</td>
</tr>
<tr>
<td>ANS</td>
</tr>
<tr>
<td>All findings combined</td>
</tr>
</tbody>
</table>

Table 1: Comparison of our three methods for solving the SOP. All values are averages over 100 instances.

6 Conclusion

In this paper we presented the Slot Optimisation Problem (SOP) that occurs in the context of online grocery shopping services, where customers select a time slot for their delivery online, when booking their service. More specifically, the SOP is concerned with finding the maximal number of time slots at which a new customer order can be inserted into an existing delivery schedule.

We proposed a novel adaptive neighbourhood search (ANS) heuristic that exploits the special structure of time slots motivated by our application. This way it can detect feasibility of insertion of a new customer order into a given time slot in a computationally cheap way. We assessed our ANS approach in a computational evaluation on a diverse set of benchmarks that replicate the real-world setting of a large supermarket chain. We compared the ANS with two other heuristics that we have introduced in the context of the capacitated vehicle routing problem with stuctured time windows (cVRPsTW): a simple insertion heuristic and a MILP-based heuristic. In our experiments we observe that the ANS outperforms both heuristics on every benchmark set and furthermore complies with the strict time limitations that are set in real-world applications.
References

N. Agatz, A. M. Campbell, M. Fleischmann, and M. Savels. Challenges and opportunities in attended home
N. Agatz, A. M. Campbell, M. Fleischmann, and M. W. P. Savelsbergh. Time Slot Management in Attended Home
A. M. Campbell and M. W. P. Savelsbergh. Decision support for consumer direct grocery initiatives. Transportation
P. Hungerländer, K. Maier, J. Pöcher, A. Rendl, and C. Truden. Solving an on-line capacitated vehicle routing prob-
lem with structured time windows. Technical Report TR-AAUK-M-O-16-12-30, Alpen-Adria Universität
Klagenfurt, Mathematics, Optimization Group, 2016.
O. Madsen and N. Kohl. An optimization algorithm for the vehicle routing problem with time windows based on
doi: 10.1137/1.9780898718515.