Simultaneous customer interaction in online booking systems for attended home delivery

Thomas R. Visser[1], Niels Agatz[2], and Remy Spliet[1]

[1] *Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands*
[2] *Department of Technology and Operations Management, Rotterdam School of Management, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands*

October 30, 2019

Abstract

In many delivery and service settings, the customer must be home when the provider arrives. If the customer is not at home, the delivery or service fails and the provider may have to return at a later point, needlessly creating additional vehicle miles and emission. To prevent such missed deliveries, it is increasingly common for providers to let customers choose from a menu of narrow time slots in which the delivery or service will take place. An important decision problem in this setting is which time slots to offer to maximize the expected number of placed orders while guaranteeing a feasible delivery schedule. In this paper, we particularly identify the occurrence of multiple customers simultaneously interact with a booking system. Indeed, customers might arrive while the system is still processing a previous customer, and customers might arrive while another customer is still deciding on his or her preferred time slot. Such simultaneous interactions lead to additional waiting time and invalid service offers, as we demonstrate in this paper. We argue that state-of-the-art procedures are not yet equipped to deal with this and present new approaches for this purpose. We illustrate their performance using real-time experiments with thousands of customers arriving over a small period of time. Our findings provide insights that may help to improve the design of current online booking systems, and opens up new areas of research.
Keywords: Attended home delivery, Concurrency control, Time Slot Management

1 Introduction

For many services, the customer must be home when the service provider arrives. This is common for the delivery of groceries, furniture and white goods but also for home services such as repairs, tech-support and home-care. If the customer is not at home, the service fails and the provider may have to return at a later point in time, needlessly creating additional vehicle miles and emission. To prevent such failures, it is increasingly common for providers to let customers choose from a menu of appointment times or narrow time slots. AmazonFresh, for example, lets customers select a one hour delivery time slot to receive their groceries. For ease of exposition, we use the term attended home delivery to refer to both delivery and home services in this paper.

To book their attended home delivery, customers typically interact with an online booking system going through the following phases: i) the customer provides a delivery location, ii) the

system shows the customer an offer, which in our context consists of a set of time slot options, iii) the customer selects one of the options or leaves, iv) the system processes the selection, if any. Companies usually want to provide an offer in *real-time* that shows which time slots are available to the customer. That is, they only want to show those delivery options that can be provided given the available capacity and the already placed orders. We refer to this as a *valid* offer. Customers prefer valid offers as they do not like to find out after the fact that their purchase attempt could not be accommodated (Breugelmans et al., 2006; Pizzi and Scarpi, 2013). At the same time, companies want to limit customer waiting times. A 2017 study suggests that every 100-millisecond delay in website load time can hurt sales conversion rates by 7% (Akamai, 2017).

However, in this paper, we show that invalid offers or waiting times are unavoidable. This is due to the simultaneous interaction of customers with the system. A simultaneous interaction occurs when a new customer arrives during the response and selection time of a previous customer. The *response time* refers to the time that the system requires to respond to a customer arrival, or the customer's selection, while the *selection time* refers to the time it takes the customer to select a time slot from the offer.

While online booking systems may crash or become irresponsive when traffic is higher than server capacity allows (Tadakamalla and Menascé, 2018), there is a more fundamental issue at play here. Even with abundant hardware capacity, simultaneous customer interaction may lead to the following complications: i) a valid offer might become invalid by a selection of another customer, and ii) additional waiting time might be experienced if a customer needs to wait for the system to respond to a previous customer. This issue relates to the concept of multi-user concurrency in computer science (Graefe, 2019; Bernstein et al., 1987) and gives rise to several trade-offs in the design and control of the system.

An offer can become invalid when the interarrival time between two customers is smaller than the sum of the response time and the selection time. Figure 1 illustrates the situation in which a customer has entered the system at the point in time that any single customer can still be served, but not more than one. If this customer places an order, no more customers can be served. If a new customer arrives during the selection time of this customer, it receives the same offer. This means that while the new customer receives a valid offer, it becomes invalid as soon as the first customer places an order. The colored area to the left of the graph in Figure 1 contains the combinations of response and selection times for which no offers are made that become invalid, and on the right offers do become invalid. Also note that the higher the response and selection time are, or equivalently, the lower the interarrival time is, the more customers will have their offers become invalid.

In practice, it is not uncommon for online booking systems to experience very short interarrival times. An extreme example is observed at the Chinese e-tailer Alibaba, who receives more than 325,000 orders per second during peak selling periods (Russell and Liao, 2018). In our collaboration with the Dutch e-grocer AH.nl, we observe that at peak times more than 50 customers simultaneously interact with its online booking system to select a time slot for home delivery.

In general, the performance of a system is determined by the interplay of the selection, response and interarrival times. While the selection and interarrival times are to a large extend

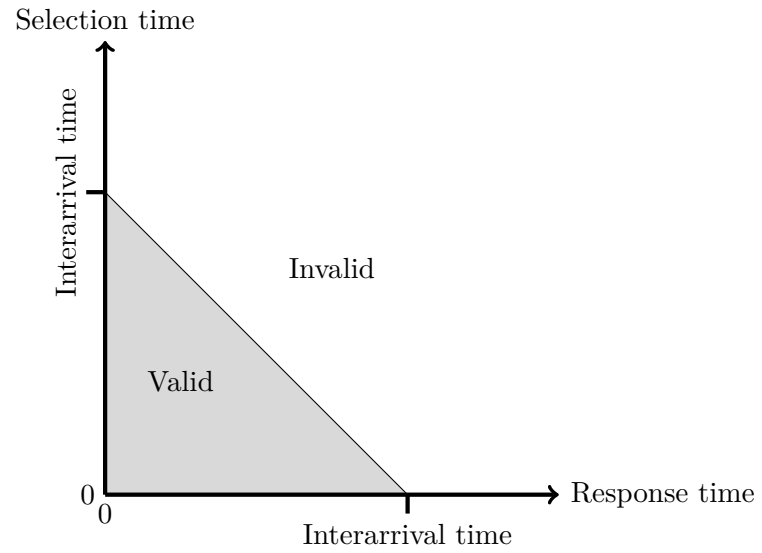

Figure 1: Timing for which offers remain valid or become invalid.

exogenous factors, the company typically has more control on the response time. In particular, the response time includes the time it takes to construct a valid offer. For the setting considered in this paper, we must construct a valid offer for each individual customer. That is because the validity of a time slot offer depends on the delivery location of the customer and the already placed orders. Moreover, constructing a valid offer is computationally challenging as it requires verifying whether there exists a delivery schedule in which this customer is visited in time. Conceptually, this is equivalent to finding a feasible solution to a vehicle routing problem with time windows (VRPTW), see Campbell and Savelsbergh (2005), which is a computationally-intensive optimization problem, see Desaulniers et al. (2014). This means that it is very challenging to achieve low response times in this context.

Building on the early work from Campbell and Savelsbergh (2005), constructing a valid time slot offer in attended home delivery has recently received an increasing amount of attention in the scientific literature, e.g., Ehmke and Campbell (2014) and Köhler et al. (2019). It is typically referred to as dynamic time slot management or DTSM (Agatz et al., 2008, 2013). This stream of literature primarily focuses on maximizing the number of valid time slots offered for each arriving customer, by finding a solution to a vehicle routing problem with time windows (VRPTW). As solving a VRPTW to optimality takes too much time in most real-world settings, the dominant approach is to heuristically check if a feasible schedule exists for a given customer and time slot. This means that the heuristic may fail to identify a feasible schedule even when one does exist. There is generally a trade-off between the quality of the heuristic and the required run time. Thus far, this literature has ignored simultaneous customer interactions and implicitly assumes that customers interact sequentially with the system. To fill this gap, we introduce a model that incorporates these interactions.

In particular, the main contributions of our paper can be summarized as follows. To the best of our knowledge, we are the first to study the simultaneous interactions between multiple customers when booking delivery services for attended home delivery. Our model combines dif-

ferent computer science and operations management components that have so far been studied independently, i.e., concurrency control and DTSM. Moreover, we include detailed operational aspects, e.g., time dependent travel times and route duration constraints. This allows for an accurate assessment of the impact of simultaneous interactions. To tackle the issues related to simultaneous customer interactions, we propose procedures for making a time slot offer and for assessing the validity of a selection. To improve the system performance without increasing customer waiting times, we propose additional procedures that can be run in the background. To assess the impact of simultaneous interactions in different circumstances, we present detailed simulation experiments including up to 8,000 customers. This allows us to evaluate the performance of the system in terms of valid and invalid orders as well as the incurred waiting times.

In Section 2, we present our model and in Section 3 we describe a concurrency control strategy that can be applied in our attended home delivery setting. In Section 4, we describe the specific algorithms used within the various procedures for concurrency control. We provide real-time simulation experiments in Section 6 to measure the performance of the system given different parameter settings. Finally, we provide concluding remarks in Section 7.

2 Model

In this section, we present our model. We describe the ordering process, including the four phases of a delivery request: customer arrival, time slot offer, selection, and validation. Furthermore, we provide a description of a delivery schedule, which is required to serve the customers that have placed an order, and determines the validity of a time slot offer or selection. Finally, we summarize the optimization problem corresponding to this model.

The ordering process takes place during the time period $[0, T]$, where T is the cut-off time. That is, during $[0, T]$ customers arrive to make a delivery request, while after T a delivery schedule is made and executed.

Let $\mathcal{C}$ be a collection of customers. At any moment in time it is unknown which customers will request a delivery in the future. For any customer $i \in \mathcal{C}$ that makes a delivery request, let $t_i \in [0, T]$ be the time at which customer i arrives to make a delivery request of size $q_i \geq 0$ and required service duration $u_i \geq 0$.

Next, a time slot offer is made. Let $\mathcal{T}$ be a set of time slots, where each time slot is an interval of time later than T. The time it takes to make a time slot offer $\mathcal{T}_i \subseteq \mathcal{T}$ is denoted by r_i^{tso} and is referred to as a response time. This response time is at least as large as the decision time d_i^{tso}, which is the time it takes to construct the time slot offer, while the response time might also include an additional delay. Which time slots are included in the time slot offer is a decision which is part of the optimization problem. Therefore, we do not consider d_i^{tso} and r_i^{tso} as inputs to the problem, but rather a result of the used algorithm to make this decision. At time $t_i + r_i^{\text{tso}}$ the time slot offer is presented to the customer.

After a selection time of s_i, the customer selects a time slot from $\mathcal{T}_i$ at time $t_i + r_i^{\text{tso}} + s_i$. We model the preferences of customers for time slots by using a general non-parametric rank-based choice model, see e.g., van Ryzin and Vulcano (2014). This means that we assume a customer has an ordered set of preferred time slots $\mathcal{T}_i^{\text{p}} \subseteq \mathcal{T}$, and it selects the first time slot in $\mathcal{T}_i^{\text{p}}$ that is

also in $\mathcal{T}_i$. If $\mathcal{T}_i^{\text{p}} \bigcap \mathcal{T}_i = \varnothing$, the customer leaves without selecting a time slot. Both the selection time and the set of preferred time slots are unknown, at least until the selection is made.

If the customer does not leave, it has to be decided after time $t_i + r_i^{\text{tso}} + s_i$ whether the selected time slot is valid. Let d_i^{val} be the decision time of this validation and r_i^{val} the resulting response time. At time $t_i + r_i^{\text{tso}} + s_i + r_i^{\text{val}}$ the result of a customer having gone through the ordering process is either i) a valid order, corresponding to a valid time slot selection, ii) an invalid order, corresponding to an invalid time slot selection, or iii) no order, when the customer leaves without selecting a time slot.

A new order is declared valid if a delivery schedule can be constructed that includes this order as well as all previously placed valid orders. We do not schedule deliveries for invalid orders. Next, we define a delivery schedule.

Let $\mathcal{C}'$ be a set of customers for which a delivery schedule is made. Consider the complete directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, where the nodes $\mathcal{V} = \mathcal{C}' \bigcup \mathcal{D}$ correspond to the customers $\mathcal{C}'$ and a set of depots $\mathcal{D}$. The set $\mathcal{D}$ could consist of multiple depots and each customer $i \in \mathcal{C}'$ can receive its delivery from any of the depots in $\mathcal{D}$. We denote by $\tau_{ij}(t)$ the travel time function, which provides the time to traverse arc $(i, j) \in \mathcal{A}$ when departing at time t. That is, the travel time is time dependent, which can be used to model for instance congestion. As is common, see for example Ichoua et al. (2003) and Gendreau et al. (2015), we assume that the time dependent travel time function is piecewise linear, continuous, and that the arrival time function $\alpha_{ij}(t) = t + \tau_{ij}(t)$ is strictly increasing (*first-in-first-out* property). At each depot $d \in \mathcal{D}$, there are K_d vehicles available for making deliveries, each with a capacity Q. A delivery is made when a vehicle visits a customer along a route. We define a route as a pair (ρ, t^ρ), where ρ is a simple cycle in $\mathcal{G}$ that starts and ends at the same depot, and t^ρ is a vector containing the arrival time at each customers on ρ. The total demands q_i, for all customers i visited on the route, may not exceed the vehicle capacity Q, and the duration of each route ρ, i.e., arrival time minus departure time from the depot, may not exceed D. Each depot $d \in \mathcal{D}$ has a time window $[a_d, b_d]$ between which routes from that depot must start and finish. Furthermore, the time that service starts at a customer is required to be in the selected time slot. Vehicles arriving early must wait. The vehicle can only depart from customer i after having spent the full service duration of u_i since the start of service. A delivery schedule is a set of at most $K = \sum_{d \in \mathcal{D}} K_d$ routes that visit all customers while satisfying the capacity, time window and route duration constraints.

The problem is to 1) construct a time slot offer at each customer arrival during the ordering process, 2) perform a validation after each customer selects a time slot, and 3) construct a delivery schedule after the cut-off time. The objective is to maximize the expected number of valid orders.

3 Concurrency control

In this section, we describe our concurrency control strategy for the ordering process. If multiple customers select time slots, there may not exist a delivery schedule accommodating all customers. Conceptually, there are two general strategies to prevent this. First, we can prevent such delivery schedule conflicts a-priori, i.e., before a customer selects his or her preferred option. We can do this by making sure that we do not simultanously offer more time slots than can be

accomodated by the available capacity. These strategies are related to the class of *pessimistic* concurrency control strategies in computer science (Bernstein et al., 1987). The disadvantage of these strategies is that they reduce the number of service options which may have a detrimental effect on demand, the number of valid orders and the corresponding capacity utilization.

Second, we can prevent delivery schedule conflicts a-posteriori, i.e., after the customer selects his or her preferred option. This means we must introduce an additional step to evaluate whether the selection is still valid. These strategies are related to the class of *optimistic* concurrency control strategies. The disadvantage of these strategies is that we only find out after the fact that a selected time slot was no longer valid.

As the pessimistic strategies appear too conservative in our context, this paper proposes a strategy that is relatively optimistic in nature. In particularly, we make use of two procedures: one for making the time slot offer and one for validation. We store a list of valid orders in memory. Additionally, a delivery schedule for the current valid orders can be stored in memory to aid the two procedures. For ease of writing, we refer to both the list and a delivery schedule as a schedule in memory. It is important to distinguish between two types of operations which are performed on the schedule in memory: *read* operations are used to access the schedule in memory, while *write* operations are used to replace or update the schedule in memory. In particular, the time slot offer procedure performs a read operation and does not perform a write operation on the schedule in memory. The validation procedure also performs a read operation, required to check whether the selected time slot can still be offered. Furthermore, if the selection is declared valid, a write operation is performed to include the new valid order in the schedule in memory.

Although read operations do not affect other procedures, write operations may create conflicts. Consider an example in which a validation procedure for order i is still running, when already starting in parallel a second validation procedure for order j. The procedures check whether a delivery schedule exists in which all current valid orders are satisfied, as well as i or j respectively. If both orders are individually declared valid, it might still happen that no delivery schedule exists that satisfies both i and j.

Therefore, we perform the validation procedures sequentially. That is, we block new validation procedures and put them in a queue, if another validation procedure is still running. After a validation procedure terminates, we start the validation procedure from the queue, if any remain, which has the earliest enqueue time.

Because time slot offer procedures do not perform write operations, we run them parallel to any other procedure that is still running, and we do not let them block any other procedure. Moreover, a new time slot offer procedure is started immediately upon a customer arrival if there are no validation procedures currently running. If a validation procedure is running, deciding whether or not to start a time slot offer procedure gives rise to a trade-off between the number of invalid orders and waiting time. When starting a time slot offer procedure while a validation procedure is running, it could be that the ensuing time slot offer becomes invalid when the validation terminates. For that reason, one might wait for the current validation procedure, or even all validation procedures in the queue, to terminate. This gives a lower likelihood of the time slot offer becoming invalid at the cost of increased response time, experienced as waiting time by the customer. In this paper, we let the time slot offer procedure wait for a single

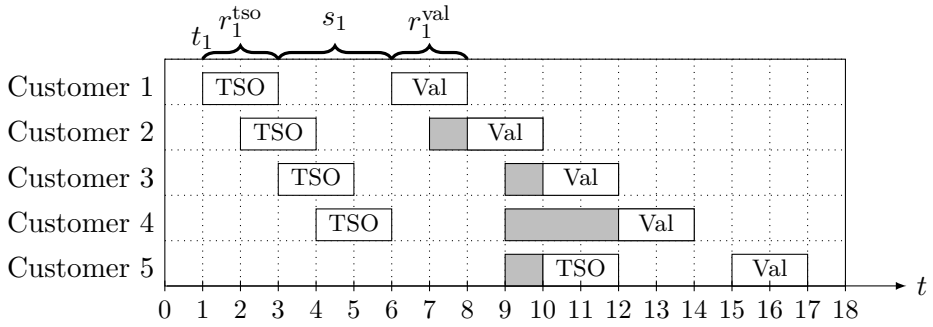

Figure 2: Concurrency control example.

current validation procedure to terminate, but we do not let it wait for the rest of the queue as preliminary tests show that this could lead to unreasonably long waiting times.

To illustrate the interaction between the different procedures, Figure 2 provides a small example with five customers. The five customers arrive at times $(t_1, t_2, t_3, t_4, t_5) = (1, 2, 3, 4, 9)$, and each customer takes three time units for their selection.

Customer 1 arrives at $t_1 = 1$ and a time slot offer is constructed (TSO). The customer receives the time slot offer after a response time of $r_1^{\text{tso}} = 2$. After $s_1 = 3$ units of selection time, customer 1 picks a time slot, at which point the validation procedure (Val) is run. After $r_1^{\text{val}} = 2$ units of time, the selected time slot of customer 1 is declared valid. This triggers a write operation.

Customer 2 arrives at time 2, which is during the time slot offer procedure for customer 1. We immediately start a time slot offer procedure in parallel, since there are no validation procedures currently running. After the selection time of customer 2, the validation procedure must be run. However, the validation procedure for customer 1 is still running, so execution is blocked until the previous validation procedure terminates. The response time is the actual decision time of the procedure plus the waiting time incurred by being blocked. In case of customer 2, the response time is three time units, while the decision time is only two time units. As can be seen by the response times of customers 3 and 4, the waiting can increase when many validation procedures are blocked, even if the decision times are all the same: the response time for customer 4 is five time units. Customer 5 arrives during the validation procedure of customer 2, and the time slot offer procedure is therefore blocked until that single validation procedure has terminated.

4 Time slot offer and validation procedures

In this section, we describe the algorithms for the time slot offer and validation procedures which we use in our numerical experiments. The algorithms used for the individual procedures are based on the state-of-the-art in the DTSM literature. The key feature of the algorithms is that they are very fast.

The strategy employed by these procedures could be characterized as myopically offering as much time slots as possible to every new customer. This increases the likelihood of an individual customer to place an order. This strategy serves as a heuristic for our optimization problem. To maximize the expected number of customers, it may sometimes be better to not offer all

possible time slots to a customer, see Liu et al. (2019). However, such considerations require the anticipation of future customers, which we consider beyond the scope of this paper.

4.1 Time slot offer procedures

A time slot offer procedure constructs a valid time slot offer. That is, for each time slot in the offer, should it be selected, a delivery schedule exists for the corresponding order and all valid orders that have been placed at the start time of the procedure. We emphasize that a valid time slot offer does not guarantee that each time slot selection results in a valid order. During the response and selection time other valid orders might be placed that makes the time slot offer invalid.

We consider two different algorithms for constructing valid time slot offers. The main difference between these two algorithms is that one builds on an existing delivery schedule in memory, while the other builds a new schedule from scratch. We use i) a first insert search which is similar to the cheapest insert search of Campbell and Savelsbergh (2006), Yang et al. (2016) and Köhler et al. (2019), and ii) a greedy construction heuristic that resembles the approach used by Campbell and Savelsbergh (2005) without their use of randomness. Our greedy construction heuristic differs slightly from the approach used by Ehmke and Campbell (2014) and Cleophas and Ehmke (2014), which constructs a schedule including the newly arriving customer for each possible time slot. It will be obvious from our numerical experiments that running multiple greedy construction heuristics in this fashion does not seem a realistic option in terms of response time, at least not when run on a single thread.

4.1.1 First insert search

For first insert search, a delivery schedule for the current valid orders needs to be available in memory. This delivery schedule does not yet contain a visit to the current customer under consideration. For each time slot, we iteratively consider all routes and all positions on these routes in which a visit might be inserted to the new customer during the considered time slot. Here, we first go over the empty routes, and then the non-empty routes. When a feasible insert position is found, the search immediately terminates and the selected time slot is included in the time slot offer. For checking feasibility, the forward/backward algorithm of Visser and Spliet (2019), which is the fastest known algorithm when both time-dependent travel times and route duration constraints are present. The complexity is $\mathcal{O}(|\mathcal{T}|\, n^2 p)$, where $|\mathcal{T}|$ is the number of time slots, n is the number of valid orders and p is the highest number of breakpoints among the time-dependent travel time functions.

4.1.2 Greedy construction heuristic

Next, we explain the greedy construction heuristic. In this case, instead of using a delivery schedule from memory, a new delivery schedule for the current valid orders is constructed from scratch at the start of the time slot offer procedure. Then, the first-insert search as described above is applied to this schedule instead of to a schedule in memory. We initialize with an empty route for every vehicle. Next, iteratively, for every customer that is not yet inserted on a route,

a feasible insert position in the schedule which yields the smallest cost increase is found by considering all possible insert positions. The cheapest insert among all customers is performed, i.e., the customer is inserted at that position. This procedure is repeated until all customers are scheduled, or no feasible insert position can be found.

The construction heuristic makes use of a cost criterion to compare different delivery schedules. We aim to find a delivery schedule that maximizes the expected number of accepted costumers. Note that we cannot directly compare schedules with respect to this objective, as any feasible insert position would have the same value. Therefore, we use a costs criterion for which we expect a schedule with lower costs to allow for the inclusion of more additional customers than a schedule with higher costs. In particular, we use the average travel time on an arc as its cost, and define the cost of a delivery schedule as the total costs of the used arcs.

The greedy construction heuristic creates a completely new delivery schedule each time. As a result, the decision time is expected to be greater than that of the first-insert search, as evidenced by the larger complexity $\mathcal{O}(n^4p + |\mathcal{T}|\, n^2p)$. However, if the quality of the schedule in memory used by first insert search is low, the greedy construction heuristic might be more successful in finding a feasible delivery schedule.

4.2 Validation procedures

A validation procedure takes as input a time slot selection from a customer and checks whether a corresponding feasible delivery schedule can be found. If so, the corresponding order is declared valid and the schedule in memory is updated, otherwise the order is declared invalid and the customer is not considered any further. Also for this procedure, we consider two different approaches in our experiments, of which one requires a delivery schedule in memory while the other does not. They are i) a cheapest insert search like Campbell and Savelsbergh (2006), Yang et al. (2016) and Köhler et al. (2019), and ii) a greedy construction heuristic like is also used as time slot offer procedure.

The cheapest insert search uses a delivery schedule stored in memory. We go over the delivery schedule in memory until the cheapest feasible insert position is found, where the costs are defined like before by the average travel times per arc. If no feasible insert position is found, the selection is declared invalid. Otherwise, the cheapest insert is performed and the delivery schedule in memory is updated. The complexity of this procedure is $\mathcal{O}(n^2p)$. It is not always necessary to wait until the cheapest insert search is completed before confirming that the selection is valid. To limit the response time, this could already be confirmed when the first feasible insert position is found. In this way, the response time may be lower than the decision time of the procedure. Therefore, in this paper when presenting response times we only include the first feasible insert time, although the decision time of course represents the computation time of the entire cheapest insert search.

Secondly, we can also apply the greedy construction heuristic provided in Section 4.1 as a validation procedure. Now, only the selected time slot is considered instead of all potential time slots in $\mathcal{T}$, and the resulting complexity is $\mathcal{O}(n^4p)$. If the construction heuristic is unsuccessful, the selection is declared invalid. Otherwise, the corresponding order is declared valid and the delivery schedule in memory (if any) is replaced by the newly constructed delivery schedule. In

this case, a confirmation of validity can only be given after the heuristic has terminated.

5 Background procedures

The algorithms that we use as time slot offer and validation procedures, are used because of their low computation times. However, the delivery schedules produced by these algorithms might not be of high quality, resulting in a low number of valid orders. Observe that some of the algorithms which we have presented, rely on a delivery schedule that is stored in memory. Therefore, improving the delivery schedule in memory might result in more valid orders. To achieve this, we run an additional procedure in the background.

A background procedure performs a read operation to access the delivery schedule in memory. It then attempts to find a better delivery schedule, where as before the quality is defined by the sum of the average travel times per arc. Only if upon termination a better delivery schedule is found, a write operation is performed to replace the delivery schedule in memory.

We maintain concurrency control by avoiding conflicting write operations as follows. A write operation is postponed until the entire queue of validation procedures is empty. Moreover, the write operation is not performed at all if since the start of the background procedure a new valid order has been placed. In that case, the improved delivery schedule is simply wasted. Note that it might sometimes be possible to repair the result of a background procedure, to update the resulting schedule with the new valid orders. However, we consider such repair schemes beyond the scope of this paper. In our numerical experiments, we only run one background procedure at any time, although multiple could be run in parallel. A background procedure is first started when a valid order is placed. When a background procedure terminates, a new background procedure is started at the earliest time that the validation procedure queue is empty.

Next, we present two algorithms that can be used as background procedure, a greedy randomized adaptive search and a neighborhood search.

5.1 Greedy randomized adaptive search

The greedy randomized adaptive search (GRASP) works almost identical to the greedy construction heuristic described in Section 4.1. However, instead of identifying the single cheapest feasible insert position in each iteration, the l cheapest feasible insert positions are found. Next, one of those l options is randomly selected and performed, each with equal probability. This algorithm has complexity $\mathcal{O}(n^4p + n^3 \log l)$. Observe that for $l = 1$, GRASP is equivalent to the greedy construction heuristic.

The GRASP is designed to provide different delivery schedules at every run. Therefore, by continually running GRASP as a background procedure, a diverse range of delivery schedules is explored. This approach strongly resembles the GRASP approaches used in Campbell and Savelsbergh (2005) and Campbell and Savelsbergh (2006), where GRASP is run a fixed number of times between each customer placement. Clearly, a fixed number of runs is not a natural stopping criterion in our model. Note that the GRASP of Yang et al. (2016) selects customers at random rather than selecting from a list of best insertions.

5.2 Neighborhood Search

Secondly, we propose a neighborhood search approach which uses a lexicographic k-exchange (Kindervater and Savelsbergh, 1997). The background procedure performs one neighborhood search iteration on the delivery schedule in memory. This consists of evaluating all schedules that can be obtained by performing a move on the delivery schedule. The best feasible move is executed. In a k-exchange neighborhood, a move consist of the exchange of a segment of customers of length up to k in one route with a segment of customers of length up to k in another route. We use the lexicographic k-exchange with ready-time function tree and forward/backward data structures (TREE+F/B) as described in Visser and Spliet (2019), which has complexity $\mathcal{O}(n^3k^2p)$.

6 Computational experiments

In this section, we present the results of our numerical experiments on several randomly generated instances with 2,000, 4,000 and 8,000 arriving customers. In Section 6.1, we describe how these random instances are generated, and in Section 6.2 we provide the computational set-up of our experiments. Next, in Section 6.3 we report the decision and response times of our methods to the instances with 2,000 customers. Similarly, we report the number of valid and invalid customers for these instances in Section 6.4. In these sections we demonstrate the effects that interarrival, response and selection times have on the number of valid and invalid orders. The placement of invalid orders does not occur throughout the entire ordering process, but rather at times where the capacity limits are almost reached. We illustrate this with an example in Section 6.5, which has implications for the number of invalid orders that can be expected for larger instances. Finally, in Section 6.6 we report on the number of valid and invalid orders, and response times for the larger instances with 4,000 and 8,000 customers.

6.1 Instances

Our aim is to study the effects of selection time and interarrival times on the performance of our methods. We generated 10 instance sets with 2000, 4000, and 8000 arriving customers, which gives a total of 30 instance sets. Each instance set consist of multiple instances in which only the selection time and the interarrival time between the customers are varied. That is, the customer locations, their time slot preference, and order of arrival are fixed for all instances in one instance set.

The characteristics of the instances are inspired by our collaboration with AH.nl. We choose the delivery area to coincide with the major urban area of The Netherlands. Our region contains four major Dutch cities (Amsterdam, Rotterdam, The Hague and Utrecht), various surrounding urban satellite cities and towns, and rural areas. Data from 2017 is used, in which our selected region contains roughly 3.3 million registered households, which is about 43% of the total number of registered households in the Netherlands (van Leeuwen et al., 2017).

We consider four depots, each located with easy highway access near a major city, and each with a fleet of identical vehicles. We assume identical demand sizes for each customer that places an order and vehicles have a capacity limit of 33 customers. Furthermore, each route has

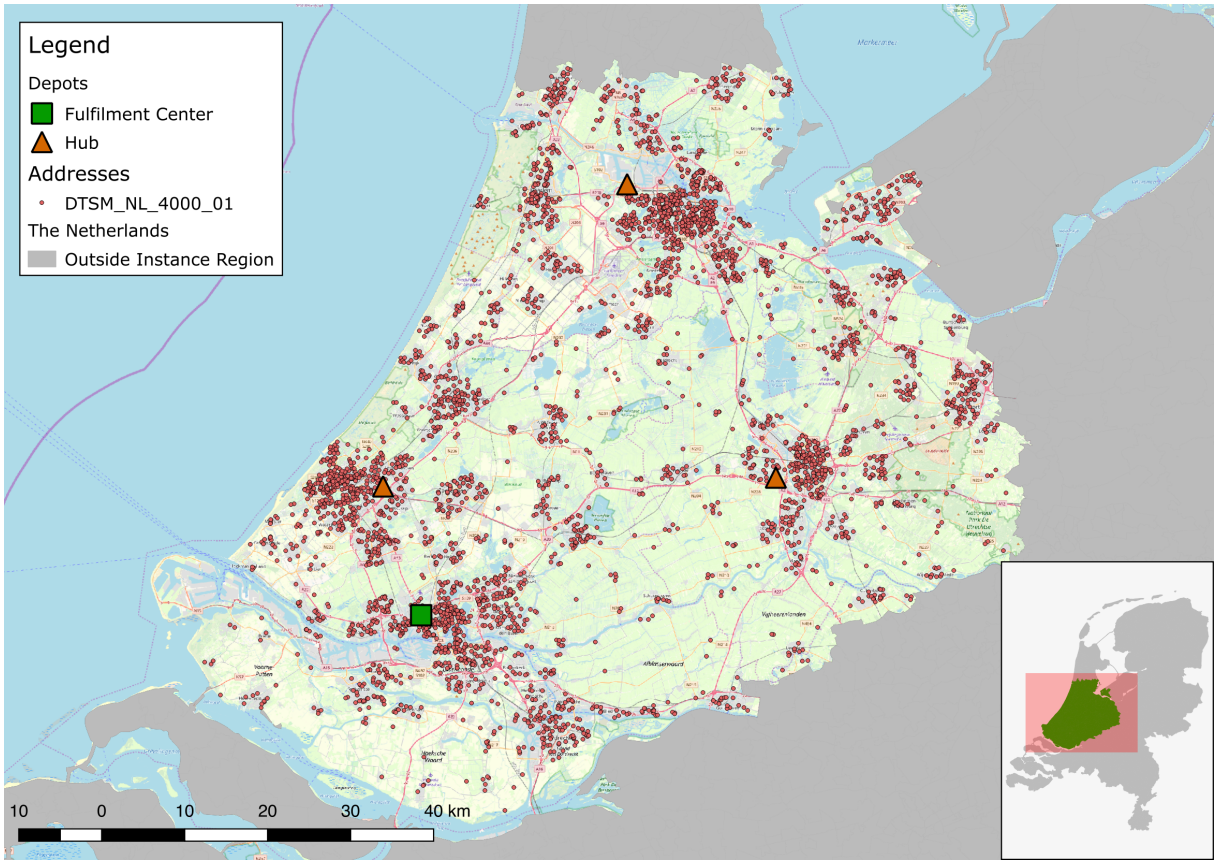

Figure 3: The locations of an instance with 4,000 customers.

	# Vehicles			Total cap.
n	Total	Fulfill.	Hub	(# cust.)
2,000	50	20	10	1,650
4,000	100	40	20	3,300
8,000	200	80	40	6,600

Table 1: Number of vehicles available and total capacity in number of customers.

a maximum duration of 6 hours due to labour regulations. We identify one depot as a *fulfilment center*, the main warehouse facility where orders are picked, while the three other depots are *hub* locations, where orders are only transferred to the delivery vehicles. This difference is reflected in our choice of the available vehicles, as shown in Table 1. The fulfilment center has twice the number of vehicles available compared to a single hub location. The total number of vehicles increases proportionally to the number of arriving customers, and therefore also the number of customers that can be accommodated based on capacity alone. However, in our instances the resulting vehicle routes appear to be mostly constrained by time, i.e., time slots and route duration, rather than by vehicle capacity.

For each of the 30 instance sets, we separately generate customer locations by selecting at random from a list of real address locations. Because our data set does not include which of the real address locations are households, we in fact first select a zip code (PC4 level) weighted on the number of households (van Leeuwen et al., 2017), and then uniformly select a real address

location at random within that zip code. This way we limit over-representation of industrial addresses in our selection. Figure 3 provides the locations of the depots and customers for an instance with 4,000 customers. The used region for all instances is shown in the inset of Figure 3. The service time of each customer is fixed and equal to five minutes.

We model a morning shift with a depot time window of [06:00, 15:00]. The set of time slots is $\mathcal{T} = \{[07{:}00, 08{:}00], [08{:}00, 14{:}00], [08{:}00, 10{:}00], [09{:}00, 11{:}00], [10{:}00, 12{:}00], [11{:}00, 13{:}00], [12{:}00, 14{:}00]\}$. Notice that these time slots are overlapping, and mostly two hours except for one shorter one hour and one longer six hour time slot.

Each customer has an ordered set of time slot preferences $\mathcal{T}_i^{\mathrm{p}}$ containing $|\mathcal{T}_i^{\mathrm{p}}| = 2$ time slots, which are drawn uniformly from the set of time slots $\mathcal{T}$. This means that each time slot is equally popular, but, because of the overlap and different widths among the time slots, this popularity is not evenly spread over time across the planning horizon [06:00, 15:00].

To generate a time-dependent travel time function, we follow a standard procedure as described, among others, by Ichoua et al. (2003). We define a nominal travel time on each arc, which represents the travel time at a nominal speed. Furthermore, we define a speed profile, which represents a time dependent speed. As is common, we use a piece-wise constant speed profile, resulting in a piece-wise linear time-dependent travel time function. The nominal travel times between all locations are calculated using *OSRM*, an open-source routing service for *openstreetmap*, and rounded to minutes. As speed profile we use nomimal speed at times in [06:00, 7:00], half speed in [07:00, 10:00], and nominal speed again in [10:00, 15:00]. This speed profile models a congestion period in between free flow. We use the same speed profile for all arcs in our network.

In our experiments, one of the aims is to demonstrate the effect that the selection time has on performance. For this reason, we consider two cases of customer selection times: i) each customer has a selection time of 30 seconds, and ii) each customer has a selection time 0 seconds, which means the customer immediately selects a time slot or leaves after the time slot offer is presented.

Furthermore, within each instance set, we consider seven different cases of the interarrival times. All cases are based on a constant interarrival time between the arriving customers, i.e., the arrival time t_i of the ith customer is $t_i = t_{i-1} + \Delta$, with $i \in \{2, \ldots, n\}$, $t_1 = 0$ and interarrival time Δ. We consider the following cases of constant interarrival times: $\Delta = 1\,\mu\text{s}$, $1\,\text{ms}$, $10\,\text{ms}$, $100\,\text{ms}$, $1\,\text{s}$, and $10\,\text{s}$. Note that the decision times of our algorithms can be less than 1 ms, but are always more than 1 µs. Furthermore, we consider an arbitrarily large interarrival time, larger than the sum of the selection and response time, referred to as $\Delta = \infty$. This represents the scenario in which no invalid orders can occur.

In all experiments, we use fixed selection and interarrival times to allow for a clear interpretation of our results, as this limits random noise. For the same purpose, we assume that all customers arrive before the cut-off time which we therefore do not specify.

6.2 Computational set-up

In our experiments, we focus on five configurations of the algorithms used for concurrency control. Observe that there are two algorithms for the time slot, validation and background procedures, while for the background procedure there is a third option of not using any algorithm.

Configuration	Time Slot Offer	Validation	Background
CON	Greedy Construction	Greedy Construction	No
INS	First Insertion	Cheapest Insertion	No
INSCON	First Insertion	Cheapest Insertion and Greedy Construction	No
INS+GR	First Insertion	Cheapest Insertion	GRASP
INS+NS	First Insertion	Cheapest Insertion	Neighborhood Search

Table 2: Configurations investigated in this paper.

One might also combine options. In particular, in our experiments we consider a validation procedure that first performs a cheapest insertion, and afterwards performs a greedy construction. Table 6.2 provides a summary of the configurations that we use. The first column provides the name of each configuration, and the other columns provide the algorithm selected for each of the procedures.

We use the following parameters for the background procedures. The GRASP used within INS+GR selects from $l = 3$ possible moves. Each time a new valid order is placed, the first new GRASP run uses $l = 1$, rather than $l = 3$, and therefore resembles a (deterministic) greedy construction algorithm. The INS+NS searches a k-exchange neighborhood. We set $k = 33$, which is an upper bound on the number of customers that fit in one route. All possible segments of length 1 up to 33 customers are tried for exchange. In the case of a sufficiently large interarrival time, i.e., $\Delta = \infty$, the INS+GR and INS+NS background procedures are limited to 10 and 100 successive runs between customer arrivals, respectively. In our experiments, a maximum number of 100 successive runs for INS+NS, is not limiting since a local optimum is typically reached in less iterations.

We use a discrete event simulator to simulate the ordering process using our configurations, which is coded in C++11 and compiled using GCC version 6.3. All simulation runs are executed as a single thread on an Intel® Xeon® E5-2650 v2 with 2.6 GHz (Turbo Boost up to 3.6 GHz) and 64 GB of RAM running Debian Linux version 9. All CPU times were measured using `std::chrono::high_resolution_clock`, a high resolution wall-clock timer, and were used with microsecond precision inside the simulations. Although the DTSM configurations are essentially multi-threaded, our custom discrete event simulator allows us to simulate a configuration using only one thread. This way, we avoid the CPU time measurements to include possible overhead that is specific to a multi-threaded/parallel implementation and the used CPU architecture. Only this one thread was run at any time on the CPU.

6.3 Decision and response times

In this section, we report on the decision and response times of applying the configurations CON, INS and INSCON to the instance sets consisting of 2,000 arriving customers with a selection time of 30 seconds. These times can be interpreted as waiting times experienced by a customer, but also affects the number of time slot offers that become invalid. As running the background procedure has no direct impact on the decision and response times, we do not consider INS+GR and INS+NS in this section.

We report the maximum times, which is a more insightful statistic than the often used average

times. Observe that the computational effort required to find a delivery schedule increases with the number of valid orders, hence so does the decision time of any procedure. As a result, the decision time is low at the start of the simulation, and increases as more valid orders are placed, according to the complexity of the used algorithm. The maximum times determine the applicability of a configuration, and are typically observed at the end of the ordering process.

Table 3 shows the maximum decision and response times for the different time slot offer and validation procedures, averaged over ten instances. The column 'Interarr.' provides the interarrival time Δ between consecutive customers. Recall that we denote a sufficiently large interarrival time by ∞. All times in the table are reported as minutes:seconds:milliseconds.

	Time slot offer						Validation					
Interarr.	max. Decision			max. Response			max. Decision			max. Response		
(s:ms)	CON	INS	INSCON	CON	INS	INSCON	CON	INS	INSCON	CON	INS	INSCON
0.001	0.2	0.0	0.0	0.2	0.0	0.0	612.5	0.5	1:162.1	13:19:572.6	181.7	4:43:202.8
1	0.2	0.0	0.0	0.2	0.0	0.0	622.9	0.5	1:144.4	13:19:382.8	0.3	4:37:178.1
10	0.2	0.0	0.1	0.2	0.0	0.1	617.3	0.5	1:158.5	13:02:296.1	0.2	4:22:187.3
100	586.4	0.8	0.2	1:091.1	0.8	992.7	612.9	0.5	1:147.4	8:56:632.6	0.2	2:57:381.6
1:000	618.2	0.9	0.8	716.0	0.9	258.1	569.3	0.5	1:152.8	569.3	0.2	257.8
10:000	611.6	0.7	0.8	611.6	0.7	0.8	553.3	0.4	1:170.6	553.3	0.2	0.2
∞	602.8	1.0	0.7	602.8	1.0	0.7	533.9	0.5	1:165.3	533.9	0.2	0.1

Table 3: Decision and response times (min:s:ms), for instances with 2,000 customers and a selection time of 30 s.

Overall, we observe that the time slot offer procedures take little time, indeed the average maximum decision times are less than a second. As expected, we also see that CON takes substantially more time than INS and INSCON. The average maximum response times of the time slot offer procedures are higher than the decision times, as additional waiting times are incurred due to being blocked by a validation procedure. This difference can be observed for interarrival times of 100 ms and higher. For lower interarrival times, the average maximum response times of all time slot offer procedures are equal to their decision times. Moreover, these times are low, 0.2 ms or less. The reason is that in this case, all customers have arrived before the first customer has made a selection. Hence, no valid order has been placed and the time slot offer consisting of all possible time slots is easy to compute.

Similar effects also apply to the decision times of a validation procedure. One difference is that for the short interarrival times of 10 ms or less, valid orders will have been placed, making the validation procedure for these instances computationally more demanding than the time slot offer procedure. However, validation procedures are put in a queue. So unlike the time slot offer procedure, validation procedures are blocked until all previously enqueued validation procedures are run. As a result, although the decision times are comparable to those of the time slot offer procedures, the response times increase substantially. For example, for an interarrival time of 100 ms, the average maximum response time for CON is almost 9 minutes, for INSCON it is almost 3 minutes, while the INS configurations still only requires 0.2 ms. In practice it might be crucial to confirm validity in a short amount of time, for instance when a customer is made to wait for confirmation of a placed order. It seems that in such a situation CON and INSCON are less suitable than INS.

Sel.	Interarr.	# Valid orders					# Invalid orders				
(s)	(s:ms)	CON	INS	INSCON	INS+GR	INS+NS	CON	INS	INSCON	INS+GR	INS+NS
0	0.001	739.5	684.2	925.9	693.3	686.7	1,260.5	1,315.8	1,074.1	1,306.7	1,313.3
	1	742.7	689.4	927.4	684.3	691.4	1,257.3	0.0	1,072.6	0.0	0.0
	10	740.9	689.4	926.1	902.8	825.8	1,259.1	0.0	1,073.9	0.0	0.0
	100	742.3	689.4	925.5	951.8	1,117.7	798.1	0.0	1,070.8	0.0	0.0
	1:000	742.5	689.4	930.4	926.3	1,174.5	0.4	0.0	0.0	0.0	0.0
	10:000	742.5	689.4	930.4	930.3	1,176.8	0.0	0.0	0.0	0.0	0.0
30	0.001	741.6	676.6	936.7	681.3	674.2	1,258.4	1,323.4	1,063.3	1,318.7	1,325.8
	1	742.4	689.5	933.1	689.2	696.5	1,257.6	1,310.5	1,066.9	1,310.8	1,303.5
	10	742.4	689.5	933.1	905.2	800.8	1,257.6	1,310.5	1,066.9	1,094.8	1,199.2
	100	739.6	692.5	933.8	944.1	1,085.3	1,057.0	342.9	1,065.7	371.9	486.2
	1:000	742.4	688.6	931.8	925.9	1,180.0	30.7	42.2	37.0	40.4	51.1
	10:000	742.0	688.8	931.1	932.1	1,173.3	3.5	4.8	4.4	4.2	6.0
	∞	742.5	689.4	930.4	934.4	1,176.8	0.0	0.0	0.0	0.0	0.0

Table 4: The number of valid and invalid orders for instances with 2,000 customers.

6.4 Number of valid and invalid orders

Next, we present the number of valid and invalid orders that result from applying CON, INS, INSCON, INS+GR and INS+NS to the instance sets consisting of 2,000 arriving customers. Table 4 shows the number of valid and invalid orders, averaged over the instance sets. The column 'Sel.' provides the selection time, i.e., 0 or 30 seconds. Again, the column 'Interarr.' provides the interarrival time Δ between consecutive customers.

We first comment on the number of valid orders. For the configurations without the background procedures, i.e., CON, INS and INSCON, Table 4 shows that the interarrival and selection times do not have a large effect on the number of valid orders. As expected, we see that CON leads to more valid orders than INS. The configuration INSCON leads to even more valid orders, which is due to the larger search space of the validation procedure.

Looking at the configurations that use the background procedures, i.e., INS+GR and INS+NS, we see that they yield more valid orders when the interarrival time increases. This is because longer interarrival times allow the background procedure more time to find better delivery schedules. For sufficiently large interarrival times, the number of valid orders of INS+GR is comparable to that of INSCON, which does not use the background procedure. Moreover, INS+NS has the highest number of valid orders when the interarrival times are 100 ms or more. The configuration INS+NS seems the dominant procedure in terms of response time and number of valid orders when the interarrival time is sufficiently large.

On the one hand, the number of valid orders are not much affected by selection time, and the interarrival time only has an effect when an improvement procedure is used. On the other hand, the number of invalid orders is subtantially affected by interarrival and selection times for all configurations.

We observe that when the interarrival time increases, the number of invalid orders goes down. Indeed, for an interarrival time of 0.001 ms, we see a substantial number of invalid orders, which decreases until there are no invalid orders when the interarrival time is sufficiently large. Even when the selection time is 0 seconds, we see invalid orders. In particular, we see invalid orders for all configurations when the interarrival time is 0.001 ms. For 1 ms, 10 ms and 100 ms this

also occurs for CON and INSCON due to their higher response times. For INS, INS+GR and INS+NS there are no invalid orders for interarrival times of 1 ms or more because the response time plus the selection time is lower than the interarrival time in this case. Note that when the selection time is 30 s, and the interarrival time is 10 ms or less, all customers arrive before the first customer has made a selection. This is why the number of invalid orders is the same for interarrival times of 1 ms and 10 ms, for the configurations CON, INS and INSCON,. When the interarrival time is 0.001 ms, we observe slight deviations in the number of invalid orders. This is because, due to slight variations in the response time, the sequence in which customers receive their time slot offer is not necessarily the same as their order of arrival, so neither is their order of making a selection.

Table 4 shows that an increase in selection time affects the number of invalid orders, when considering interarrival times of 100 ms and more. Among the five configurations and these three interarrival times, only in one case does the number of invalid orders go down, while in the other 14 cases a substantial increase is observed.

6.5 Invalid orders over time

Invalid orders occur when a time slot is selected that can no longer be accommodated. Roughly stated, this happens when during the response and selection time, one or more other customers use up the remaining capacity. This means that a feasible delivery schedule can be found at the time of making the time slot offer but not at the time that the customer chooses a time slot.

At the start of the ordering process there is often ample capacity available to accommodate new customers and no invalid orders will occur. However, when the limits of capacity are reached, time slot offers can become invalid. This means that there is only a limited amount of time during which the system is at risk of getting invalid orders. The number of invalid orders during this time, depends on the interarrival, response and selection times, as explained before.

To illustrate the dynamics over time, we consider a single instance from our set, with 2,000 arriving customers and a selection time of 30 seconds. For the interarrival times of 10 ms, 100 ms, 1 s and 10 s, Figure 4 shows graphs of the number of invalid orders after each customer is processed. The graphs show that there are no invalid orders among the first 500 delivery requests, because the capacity limits are not yet reached. As soon as the remaining capacity becomes limited, invalid orders start occurring. As the different configurations produce different delivery schedules and valid orders, we see that the first invalid orders occur at different points in time.

In case of an interarrival time of 10 ms in Figure 4(a), we observe a steady increase in the number of invalid orders until all customers are processed. For the other interarrival times, and almost all configurations, the number of invalid orders stops increasing at some point. This happens when it is evident that the capacity limit has been reached, and new customers are not offered any time slots. From that moment onward, new customers arrivals result in no order, instead of an invalid order. This demonstrates that there is a limited time range during which invalid orders occur.

Note that as the configurations that use a background procedure, i.e., INS+GR and INS+NS, improve the schedule in memory, it may be possible that after a period in which no time slots are offered, the time slots would again be offered. This means that after a period of no orders,

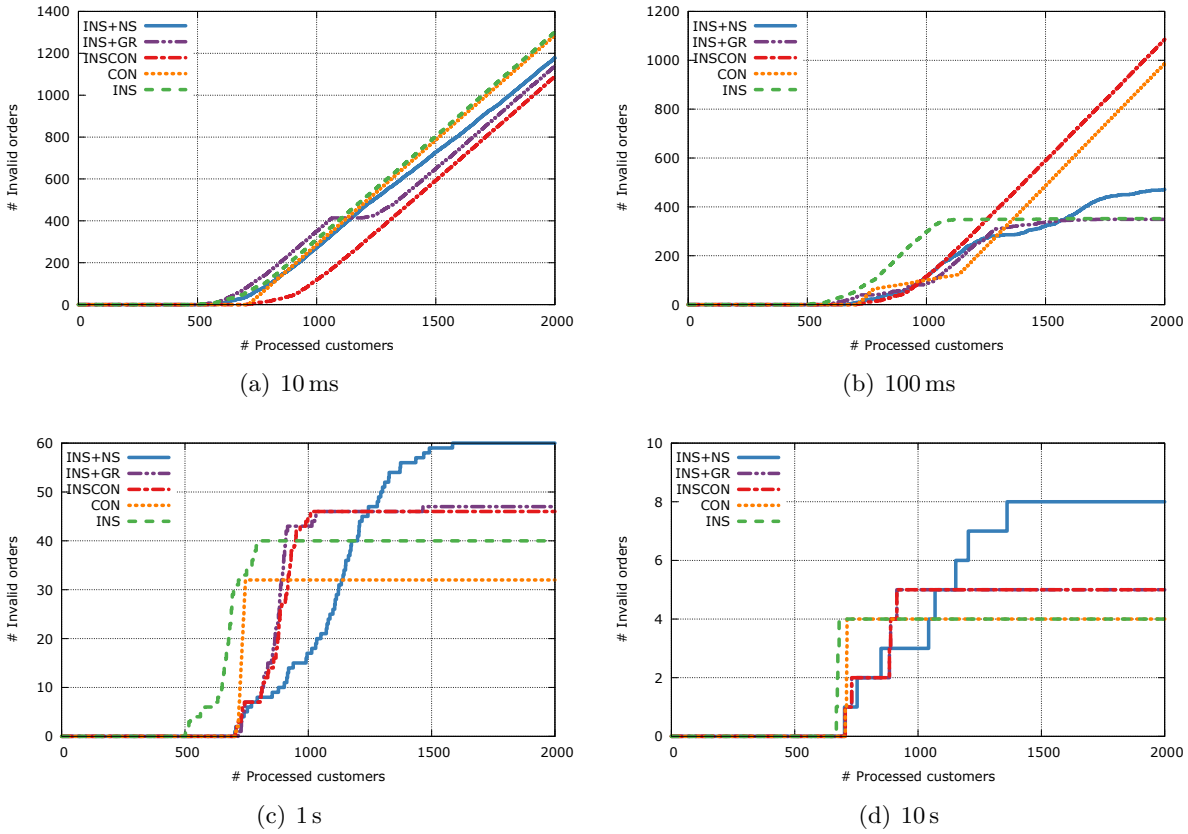

Figure 4: Number of invalid orders for an instance with 2000 customers with interarrival times: (a) 10 ms, (b) 100 ms, (c) 1 s, and (d) 10 s.

we see new valid orders again, but also new invalid orders. This effect can most clearly be seen in Figure 4(d) for INS+NS.

6.6 The impact of the number of customers

Next, we present the results for the configurations INS, INS+GR and INS+NS to the larger instance sets containing 4,000 and 8,000 arriving customers, and a selection time of 30 seconds. We do not include CON and INSCON, because the response times are prohibitively large. We report the response times and number of valid and invalid orders for these larger instances. Note that as shown in Table 1, not only the number of customers is larger in these instance sets, the capacity is also proportionally larger.

Table 5 shows the average number of valid and invalid orders, as well as the average maximum response time of the time slot offer procedure in milliseconds, 'TSO max Response (ms)'. The column '$|\mathcal{C}|'$ provides the amount of arriving customers, and once more the column 'Interarr.' provides the interarrival time Δ between consecutive customers.

First, we observe that the maximum average response times of the time slot offer procedures are less than 6.6 ms in all cases, even for instances with 8,000 arriving customers. We wish to note that the response times of the validation procedures are larger than those of the time slot offer procedures. For instances with 8,000 customers and interarrival times of 0,001 ms, the largest average maximum response time of the validation procedure is observed, which is 3 seconds

$\|\mathcal{C}\|$	Interarr.	# Valid orders			# Invalid orders			TSO max. Response (ms)		
	(s:ms)	INS	INS+GR	INS+NS	INS	INS+GR	INS+NS	INS	INS+GR	INS+NS
4,000	0,001	1,498.9	1,513.4	1,497.0	2,501.1	2,486.6	2,503.0	0.1	0.1	0.1
	1	1,503.0	1,512.4	1,532.2	2,497.0	2,487.6	2,467.8	0.1	0.1	0.1
	10	1,503.0	1,648.8	1,564.5	2,497.0	2,351.2	2,435.5	0.0	0.1	0.1
	100	1,505.6	2,177.1	2,134.7	383.4	863.1	715.1	2.3	1.7	2.1
	1:000	1,506.8	2,235.1	2,705.9	44.2	93.1	134.1	2.3	3.0	2.8
	10:000	1,505.3	2,168.6	2,736.3	4.6	5.2	6.0	2.1	3.7	1.7
8,000	0,001	3,387.8	3,375.9	3,370.5	4,612.2	4,624.1	4,629.5	0.1	0.1	0.1
	1	3,386.0	3,412.4	3,390.7	4,614.0	4,587.6	4,609.3	0.1	0.1	0.1
	10	3,390.0	3,379.1	3,445.0	3,264.5	3,260.7	3,261.6	3.8	3.1	3.0
	100	3,398.2	4,225.1	3,948.9	398.3	610.0	614.3	4.0	5.3	5.2
	1:000	3,403.8	5,177.2	5,091.3	41.3	151.5	234.2	4.1	5.4	6.6
	10:000	3,401.3	5,135.6	6,083.3	3.7	10.3	11.1	4.0	6.0	3.0

Table 5: Results for the large instance sets, with 30 s interarrival time.

for all configurations. This may still be acceptable, even if a customer is kept waiting for a confirmation.

Although the response times increase with the number of customers, this is not necessarily the case for the number of invalid orders. As illustrated in Section 6.5, invalid orders only occur for the duration of time when nearing the capacity limits. This duration is primarily dependent on the response, selection and interarrival times. For instances with an interarrival time of 100 ms or more, the simulation encompasses this full duration. As a result, it can be observed from Tables 4 and 5 that the number of invalid orders using the configuration INS is roughly the same for all instances. That is, it is independent of the instance size.

However, for the configurations INS+GR and INS+NS, which use a background procedure, the number of invalid orders is higher for the instances with 4,000 and 8,000 customers than for the instances with 2,000 customers. Because the ordering process spans a longer time, the improvement procedure potentially replaces the schedule in memory at more separate moments. Although this can have a positive effect on the number of valid orders as explained in Section 6.5, and demonstrated in Table 5, there is also a downside. Now, the system is more often in the situation of being close to the capacity limit. As a result also the number of invalid orders increases.

To conclude, we wish to point out that the situation of our industry partner most closely resembles the case of interarrival times of 1 s and selection times of 30 s. In our simulation on this set of instances, there are at least 30 customers interacting simultaneously with the system, depending on the response times. In this case, we see that we are faced with at least an average of 41.3 invalid orders for all our configurations.

6.7 Travel distance per order

After the cut-off time, the company can start with the execution of the deliveries. In some settings, there may be time to further optimize the routes before starting order picking and execution. However, as customers are increasingly expecting shorter lead-times, there is often not much time available. As such, it is interesting to look at the quality of the delivery schedule in memory at the cut-off time of the different procedures. Here, we report the average travel distance per order as this relates to emissions, fuel consumption and traffic congestion.

	# Valid orders			km per order		
$\|\mathcal{C}\|$	INS	INS+GR	INS+NS	INS	INS+GR	INS+NS
2000	742,4	931,80	925,9	15,9	10,8	7,6
4000	1506,8	2235,1	2705,9	14,2	8,1	6,0
8000	3403,8	5177,2	5091,3	12,1	6,1	6,9

Table 6: Valid orders and km per order.

Table 6 provides an overview of the average number of kilometers per order, averaged over the instances in the instance set. Like before we only include INS, INS+GR and INS+NS. For illustrative purposes we present the results for an interarrival time of 1 s. The results show that the background procedures help to create delivery schedules in memory that have a substantially lower travel distance per order.

7 Concluding remarks

This study aims to contribute to a better understanding on the performance of online booking systems. We argue that serving a large number of customers requires a booking system that allows for simultaneous customer interactions. These simultaneous interactions occur when customers arrive during the response time of the system or during the selection time of previous customers. We demonstrate that waiting time or invalid offers are unavoidable. In particular, we provide a detailed study in the context of attended home delivery. This setting is relevant, as booking systems are indispensable to avoid missed deliveries and the needless creation of additional vehicle miles. Moreover, the risk of problems related to simultaneous customer interactions is high, because it is challenging to ensure short response times as checking the capacity availability involves solving a computationally challenging optimization problem.

The detailed simulation experiments show that customers may incur high waiting times and many orders may be invalid. This is especially the case when using standard time slot offer procedures. We also introduce the use of a background procedure to improve the quality of the time slot offers. This is specifically tailored to the dynamics of a booking system with simultaneous customer interactions. Our experiments suggest that using a background procedure results in a high number of customers that can be accommodated, as well as having acceptable waiting times, at the expense of having more invalid orders.

As this is the first study focussing on simultaneous customer interactions in booking attended home delivery, it opens many possible directions for future research. It is interesting to study the impact of other concurrency control strategies on the trade-offs between valid orders, invalid orders, waiting times, and service levels. Especially the design of strategies that prevent invalid orders gives rise to several questions. For example, for every time slot that is offered to a customer, we can store a separate delivery schedule that includes this customer. For every next customer, we only offer a time slot if it can result in a valid order for all these delivery schedules. This ensures that for every possible selection by the first customer, we can accomodate the next customer. However, this would result in a combinatorial explosion of the number of stored delivery schedules, and a corresponding increase in response times. It is interesting to investigate

whether it is possible to prevent invalid orders more efficiently.

Furthermore, it is not clear how to deal with invalid orders in practice. One could, for example, simply inform the customer that the selected service option is no longer available. Moreover, it is also possible to dynamically update all offers to customers currently in the system whenever a customer commits an order. Both approaches may, however, have a negative impact on customer satisfaction and potentially reduce future purchases. To better understand how customers react to different approaches requires empirical research at the interface between marketing and operations management.

Acknowledgments

This research was funded by NWO, the Netherlands Organisation for Scientific Research, as part of the multi-annual research programme on Sustainable Logistics. It is co-funded by ORTEC B.V. and AH.nl.

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*, volume 43 of *Operations Research/Computer Science Interfaces*. Springer US, 2008.

N. Agatz, A. M. Campbell, M. Fleischmann, J. van Nunen, and M. W. P. Savelsbergh. Revenue management opportunities for internet retailers. *Journal of Revenue & Pricing Management*, 12(2):128–138, 2013.

Akamai. Akamai Online Retail Performance Report: Milliseconds Are Critical, Apr. 2017. Retrieved from https://www.akamai.com/uk/en/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp.

P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*, volume 370. Addison-wesley New York, 1987.

E. Breugelmans, K. Campo, and E. Gijsbrechts. Opportunities for active stock-out management in online stores: The impact of the stock-out policy on online stock-out reactions. *Journal of Retailing*, 82(3): 215–228, 2006.

A. M. Campbell and M. W. P. Savelsbergh. Decision support for consumer direct grocery initiatives. *Transportation Science*, 39(3):313–327, 2005.

A. M. Campbell and M. W. P. Savelsbergh. Incentive schemes for attended home delivery services. *Transportation Science*, 40(3):327–341, 2006.

C. Cleophas and J. F. Ehmke. When are deliveries profitable? *Business & Information Systems Engineering*, 6(3):153–163, 2014.

G. Desaulniers, O. B. Madsen, and S. Ropke. Chapter 5: The Vehicle Routing Problem with Time Windows. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*, volume 18 of *MOS-SIAM Series on Optimization*, chapter 5, pages 119–159. SIAM - Society for Industrial and Applied Mathematics, Philadelphia, second edition, 2014.

J. F. Ehmke and A. M. Campbell. Customer acceptance mechanisms for home deliveries in metropolitan areas. *European Journal of Operational Research*, 233(1):193 – 207, 2014.

M. Gendreau, G. Ghiani, and E. Guerriero. Time-dependent routing problems: A review. *Computers & Operations Research*, 64:189 – 197, 2015.

G. Graefe. On transactional concurrency control. *Synthesis Lectures on Data Management*, 14(5):1–404, 2019.

S. Ichoua, M. Gendreau, and J.-Y. Potvin. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, 144(2):379–396, 2003.

G. A. P. Kindervater and M. W. P. Savelsbergh. Vehicle routing: Handling edge exchanges. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 337–360. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

C. Köhler, J. F. Ehmke, and A. M. Campbell. Flexible time window management for attended home deliveries. *Omega*, 2019. doi: 10.1016/j.omega.2019.01.001. URL http://www.sciencedirect.com/science/article/pii/S030504831830803X.

N. Liu, P. M. van de Ven, and B. Zhang. Managing appointment booking under customer choices. *Management Science*, 2019.

G. Pizzi and D. Scarpi. When out-of-stock products do backfire: Managing disclosure time and justification wording. *Journal of Retailing*, 89(3):352–359, 2013.

J. Russell and R. Liao. Singles' Day: China's $25 billion shopping festival explained. *TechCrunch*, Nov. 2018. Retrieved from https://techcrunch.com/2018/11/09/alibaba-singles-day-11-festival/.

V. Tadakamalla and D. A. Menascé. An analytic model of traffic surges for multi-server queues in cloud environments. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 668–677. IEEE, 2018.

N. van Leeuwen, T. Guldemond, and F. Faqiri. Statistische gegevens per vierkant en postcode 2017. *Statistics Netherlands (CBS)*, Nov. 2017. Retrieved from https://www.cbs.nl/nl-nl/dossier/nederland-regionaal/geografische-data/gegevens-per-postcode.

G. van Ryzin and G. Vulcano. A market discovery algorithm to estimate a general class of nonparametric choice models. *Management Science*, 61(2):281–300, 2014.

T. R. Visser and R. Spliet. Efficient move evaluations for time-dependent vehicle routing problems. Technical report, Accepted for publication in Transportation Science, 2019. URL http://hdl.handle.net/1765/100852.

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.