https://pubsonline.informs.org/journal/trsc

TRANSPORTATION SCIENCE
Vol. 58, No. 5, September–October 2024, pp. 1056–1075
ISSN 0041-1655 (print), ISSN 1526-5447 (online)

Managing Concurrent Interactions in Online Time Slot Booking Systems for Attended Home Delivery

Thomas R. Visser,[a] **Niels Agatz,**[b,*] **Remy Spliet**[c]

[a] ORTEC, 2719 EA Zoetermeer, Netherlands; [b] Rotterdam School of Management, Erasmus University, 3062 PA Rotterdam, Netherlands; [c] Erasmus School of Economics, Erasmus University, 3062 PA Rotterdam, Netherlands

*Corresponding author

Contact: thomas.visser@ortec.com, https://orcid.org/0000-0003-1456-3325 (TRV); nagatz@rsm.nl, https://orcid.org/0000-0003-3514-201X (NA); spliet@ese.eur.nl, https://orcid.org/0000-0003-4821-2945 (RS)

Received: January 10, 2023
Revised: July 31, 2023; February 20, 2024; May 14, 2024
Accepted: May 15, 2024
Published Online in Articles in Advance: July 17, 2024

https://doi.org/10.1287/trsc.2022.0445

Abstract. Many goods and services require the customer to be at home to receive the delivery. In the context of attended home delivery, customers can typically choose from a menu of delivery time slots. We consider the problem of dynamically managing the offered time slots and delivery bookings given the available fleet capacity. When multiple customers interact with the online booking system at the same time, this can lead to conflicts. Although managing such concurrent interactions is an important challenge in attended home delivery systems, it has not yet been addressed in the literature. We present a concurrency control strategy and several fast route planning approaches to manage time slots in real time. To combine fast response times with high quality slotting decisions, we introduce background procedures that use the time between successive order placements to improve the performance of the time slot offer and validation procedures. Our detailed computational experiments based on realistic instances provide insights into the effectiveness of our background procedures and the complex trade-offs between waiting times, valid orders, and invalid orders. We also discuss several relevant new areas of research in concurrency control for time slot management.

Funding: This work was supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek [Grant 438-213-204]. It is co-funded by ORTEC B.V. and AH.nl.

Keywords: attended home delivery • concurrency control • time slot management

1. Introduction

Many services require the customer to be at home to facilitate their delivery. This is common for the delivery of groceries, furniture, and large appliances but also for home services such as repairs, technical support, and home care. If the customer is not at home, the service fails and the provider may have to return at a later time, unnecessarily generating additional vehicle miles and emissions. To avoid such failures, it is common for providers to allow customers to choose from a menu of appointment times or delivery windows. AmazonFresh, for example, allows customers to select a one-hour delivery window to receive their groceries. 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, that is, a set of time slot options, (iii) the customer selects one of the options or leaves, and (iv) the system processes the selection, if any. Customers can place orders up to a certain cutoff time after which a route plan is made in which all deliveries are scheduled. To ensure a smooth booking experience for the customer, it is important to limit the *response times* of the system, that is, the time between the arrival of the customer and offering the time slots.

There is a growing body of literature on managing incoming orders in attended home delivery through dynamic time slot management (DTSM) for a fixed vehicle fleet capacity (Fleckenstein, Klein, and Steinhardt 2023; Waßmuth et al. 2023). A fundamental question is how to construct a valid time slot offer for a given customer, consisting of all time slots in which the customer can be served. Constructing a valid time slot offer is computationally challenging because it requires checking whether there exists a feasible delivery schedule for the arriving customer and all previously accepted customers. Conceptually, this is equivalent to finding a feasible solution to a vehicle routing problem with time windows (VRPTW). This makes it difficult to ensure fast response times, especially when the number of customers, and the associated VRPTW instances, is large. Therefore, most of the literature has primarily focused on designing fast and high-quality heuristics for the VRPTW to construct a time slot offer.

However, in addition to the challenge of quickly determining a time slot offer, there is another challenge that is often overlooked in the literature. The concurrent interaction of customers with the system introduces additional complexity. We identify the following complications arising from the simultaneous interaction of multiple customers with the system. First, a valid time slot offer may be invalidated by another customer's selection. This can happen if the *interarrival time* between two customers is less than the sum of the response time and the *selection time*, that is, the time it takes a customer to select a time slot. For example, two customers that arrive around the same time may be offered the same time slot, even if only one more customer can be served in that time slot. A second issue is that concurrent decisions and system updates might lead to either data inconsistencies and an infeasible plan after the cutoff or additional waiting time experienced by the customer if a customer needs to wait for the system to respond to a previous customer. This means that evaluating the run times of time slot methods for each single customer independently (as is typically done in the current literature) only provides a lower bound on the actual waiting times. These issues relate to the concept of multiuser concurrency in computer science (Bernstein, Hadzilacos, and Goodman 1987; Graefe 2019) and give rise to several tradeoffs in the design and control of the system.

With a growing number of customers, there are more customers interacting with the system simultaneously. For example, at Dutch e-grocer AH.nl, it is not uncommon to have more than 50 customers from the same region interacting with the online booking system at the same time. Peak traffic typically occurs in the evening, when many customers are considering the same time slots for the next day's delivery. We first encountered the issues related to concurrent customer interaction when implementing a DTSM system at e-grocer AH.nl together with ORTEC, an international supplier of routing and scheduling optimization and advanced analytics software. To increase the utilization of their fulfilment capacity, AH.nl wanted an approach based on real-time routing to replace their previous approach which used fixed estimates of the number of orders that can be fulfilled per time slot per delivery vehicle. By reducing the slack in their route schedules, the consequences of ignoring concurrent customer interaction become more apparent. ORTEC experienced similar challenges related to concurrent customer interaction in various other DTSM implementation projects around the globe (G. Kant, Global Director Logistics Industry at ORTEC, personal communication, June 24, 2020). We have since learned that many companies are struggling with these challenges.

In this paper, we present a new model for DTSM that includes the time dimension in the different ordering phases. In particular, it incorporates the interarrival, response and selection times. Interestingly, although the interarrival and selection times can be modeled exogenously, for example, as random variables, the response time is dependent on the algorithm, implementation, and computer used to construct a time slot offer. These solution times and their impact on the system dynamics are often ignored in the literature. Our results based on realistic, detailed, real-time simulations show that state-of-art methodology from the DTSM literature cannot be applied in this more realistic environment. This suggests the need for using control methods for concurrent interaction.

We present a concurrency control strategy in this paper to manage the concurrent interactions. To ensure consistency, we introduce an additional validation check that alerts the customer when the selected slot is no longer available. This means that we not only perform an initial procedure to create a time slot offer but also a validation procedure. Each procedure needs to be performed almost instantaneously to ensure a smooth booking process. Coordinating the different steps for different customers creates an extra layer of complexity in time slot management. Moreover, as these processes are fundamentally interrelated, it is not possible to solve the associated problems by adding hardware capacity or performing computations in parallel. This gives rise to a general tradeoff between the speed and quality of the procedures. Because the state-of-the-art methods as described in the literature would create prohibitively long response times if naively implemented within our framework, we propose the use of *background procedures* to allow constructing high quality time slot offers with low response times. The key idea of the background procedures is to use the periods without much traffic on the website to proactively run procedures to better prepare for the next arrivals.

The main contributions of this paper can be summarized as follows. (i) We are the first to identify a new problem in dynamic time slot management related to the concurrent interaction between multiple customers booking delivery services for attended home delivery. This is a key practical challenge that has thus far been ignored in the literature. (ii) To tackle the problem, we propose a concurrency control strategy for making time slot offers and for assessing the validity of customer selections. Within this framework, we adopt the state-of-the-art heuristic methods from the literature. Moreover, we introduce background procedures that use the time between subsequent order placements to improve the performance of the different procedures. (iii) In our extensive realistic numerical experiments with instances of up to 8,000 customers, we show the impact of different system parameters on the concurrency-related issues and show that complementing the state-of-the-art methods with our background procedures allows for a higher number of valid orders without additional waiting times. (iv) Finally, we discuss several relevant new areas of research related to concurrency control in time slot management.

The remainder of this paper is structured as follows. In Section 2, we discuss the related literature. In Section 3, we provide a problem description, and in Section 4, we describe a concurrency control strategy that can be applied in our attended home delivery setting. In Section 5, we describe the specific algorithms used within the various procedures for concurrency control, and in Section 6, we describe the background procedures. In Section 7, we describe the instances that we generate for our numerical experiments, which are based on real-world data. We provide real-time simulation experiments in Section 8 to measure the performance of the system given different parameter settings. Finally, we provide concluding remarks and a classification of new research directions in Section 9.

2. Literature Review

Building on the early work of 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 (Ehmke and Campbell 2014; Köhler, Ehmke, and Campbell 2020). It is typically referred to as dynamic or operational time slot management or DTSM (Agatz et al. 2013; Fleckenstein, Klein, and Steinhardt 2023).

In contrast to more static and tactical approaches (Agatz et al. 2011), the dynamic setting focuses on the real-time management of time slot offers based on arriving and already accepted orders. 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) for every possible time slot. As determining whether a feasible solution to a VRPTW exists is computationally prohibitive in most real-world settings, the dominant approach is to use heuristics. One disadvantage of this is that we may fail to identify a feasible schedule even when one does exist, resulting in less time slots being offered and thereby potentially losing sales. There is generally a tradeoff between the quality of the heuristic and the required run time.

Campbell and Savelsbergh (2005) are the first to study a DTSM problem. They present an insertion-based heuristic in which they first construct a new route plan for all accepted delivery requests and then try to insert the current delivery request under consideration. In Campbell and Savelsbergh (2006) the authors extend this approach by maintaining a set of multiple feasible schedules to increase the number of insertion options. The core idea of checking feasibility for a certain time slot by inserting the new customer in one or more existing route plan for the already accepted customers is the most common method presented in time slot management (Ehmke and Campbell 2014; Köhler, Ehmke, and Campbell 2020; Agatz, Fan, and Stam 2021).

Checking feasibility is a required first step in more sophisticated time slot management systems. Given a set of feasible slots, several papers have proposed dynamic pricing policies to determine incentives to steer customers to the most profitable time slots (Yang et al. 2016, Koch and Klein 2020). The work in this area typically focuses on estimating the (opportunity) costs to serve a specific customer in different time slots given the already accepted customers and forecasts of future demand (Yang and Strauss 2017, Abdollahi et al. 2023).

Although the concurrent interaction of customers with the system has thus far been ignored in the time slot management literature, there is a large body of literature on concurrency control in multiuser information systems (Graefe 2019). Most closely related to our setting is the work on concurrency issues in transaction systems for e-commerce (Chen 2009, Khaing and Myint 2017). In traditional booking systems, concurrency control helps to prevent selling the same product or capacity unit, for example, a seat on a flight, to multiple customers (Lewandowski et al. 2007). Although the capacity in terms of the fleet and labor force is fixed in our setting, the number of orders that can be served depends on their sizes and locations. This gives rise to two additional complexities. (i) In addition to performing concurrent read and write operations in a database to keep track of placed orders, we need to solve a time-consuming optimization problem to find out which time slots we can still offer, and (ii) the capacity consumption for different time slots are interrelated. This means that even finding out whether a conflict occurred is a time-consuming step.

We are only aware of one recent paper in the operations management literature that considers concurrency-related issues, albeit without explicitly using the terminology. Ausseil, Pazour, and Ulmer (2022) focus on determining supplier menus in matching requests and service suppliers in a peer-to-peer transportation platform. This gives rise to a tradeoff between offering requests to different service suppliers sequentially (pessimistic control) or simultaneously (optimistic control). However, most of the operations management literature has ignored concurrent system interactions. Furthermore, the information systems literature has only scarcely addressed topics related to attended home delivery platforms. To bridge this gap, we introduce an attended home delivery model that incorporates concurrent interactions.

3. Problem Description

In this section, we provide a problem description. 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 corresponding optimization problem.

The ordering process takes place during the time period $[0, T]$, where T is the cutoff time. That is, during $[0, T]$ customers arrive to make a delivery request, whereas 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 $q_i \geq 0$, for example, the number of items to be delivered, and required service duration $u_i \geq 0$, that is, the time required at the customer for delivery.

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 algorithm used 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 $t_i + r_i^{\text{tso}} + s_i$ or may leave without selecting a time slot. Both the selection time and the preferred time slot 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, which might include an additional delay. 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 assume that customers who place an order will be available to receive their order in their selected time slot. This is in line with current practice at our industry partner AH.nl. 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 $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ with arcs $\mathcal{A}$ and where the vertices $\mathcal{V} = \mathcal{C}' \cup \mathcal{D}$ 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 for instance be used to model congestion. As is common (Ichoua, Gendreau, and Potvin 2003; Gendreau, Ghiani, and Guerriero 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 customer 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, that is, arrival time minus departure time from the depot, may not exceed the limit S on the duration of a shift for a driver. 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 consists of the following steps. (1) Construct a time slot offer at each customer arrival, (2) perform a validation each time a time slot is selected by a customer, and (3) construct a delivery schedule after the cutoff time. The objective is to maximize the expected number of valid orders. This objective is common in practice as many online retailers focus on gaining market share by serving as many customer orders as possible, that is, maximizing the number of valid orders.

4. 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, the literature distinguishes between two general strategies to prevent concurrency-related conflicts, that is, (1) *pessimistic* and (2) *optimistic* strategies (Bernstein, Hadzilacos, and Goodman 1987). In the terminology of our application, pessimistic strategies prevent conflicts in the delivery schedule a priori, that is, before a customer selects their preferred option. Optimistic strategies do not necessarily prevent all conflicts, and if a conflict occurs, this has to be repaired a posteriori.

A pessimistic concurrency control strategy for DTSM is obtained by enforcing that we do not concurrently interact with more customers than can be accommodated by the available capacity. A simple and natural example of a pessimistic strategy would let customers interact with the system one by one, where a new customer must wait until the previous customer has made a selection. In the context of e-grocery delivery, where the selection time is typically much longer than the interarrival time, this would create large waiting times and would severely limit the throughput of the system, that is, the total number of customers that the system can process. For example, with customers entering the system every second and selection times of 30 seconds, in eight hours only 960 customers can be processed, and the waiting time rises to nearly eight hours. Therefore, this option is not viable in many applications, including ours. For an optimistic control strategy, conflicts may arise a posteriori, that is, after the customer selects their preferred option. This means we must introduce an additional step to evaluate whether the selection is still valid. The disadvantage of an optimistic strategy is that we only find out afterward that a selected time slot was no longer valid.

In this paper, we develop an optimistic concurrency control strategy for the time slotting context. We elaborate on possible pessimistic strategies in our future research section. In particular, 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 that are performed on the schedule in memory: *read* operations are used to access the schedule in memory, whereas *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 and *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.

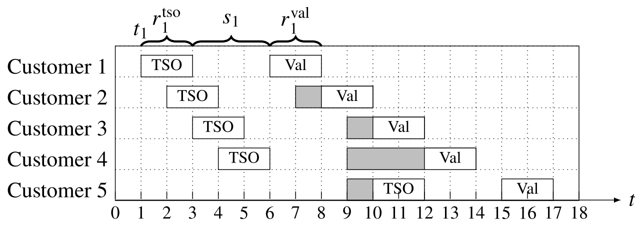
Because time slot offer procedures do not perform write operations, we run them parallel to any other procedure that is still running, and we do not let them block any other procedure. Moreover, a new time slot offer procedure is started immediately on a customer arrival if there are no validation procedures currently running. If a validation procedure is running, deciding whether to start a time slot offer procedure gives rise to a tradeoff 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 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 1 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

Figure 1. Concurrency Control Example



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, whereas 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.

5. Time Slot Offer and Validation Procedures

In this section, we describe the algorithms for the time slot offer and validation procedures that 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 used by these procedures could be characterized as myopically offering as many 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, Van De Ven, and Zhang (2019). However, such considerations require the anticipation of future customers, which we consider beyond the scope of this paper.

5.1. Time Slot Offer Procedures

A valid time slot offer to a customer, which is constructed by the online booking system at a particular time, is a menu of time slots such that for each time slot, should it be selected, a delivery schedule exists for the corresponding order and all valid orders that have been placed prior. The algorithm that creates it, is referred to as a time slot offer 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, whereas the other builds a new schedule from scratch. We use (i) a first-insert search that is similar to the cheapest insert search of Campbell and Savelsbergh (2006), Yang et al. (2016), and Köhler, Ehmke, and Campbell (2020) 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) and 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.

5.1.1. 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. Initially, the delivery schedule is empty. 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 nonempty 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 (2020) is used, which is the fastest known algorithm when both time-dependent travel times and route duration constraints are present. The time complexity is $\mathcal{O}(|\mathcal{T}|n^2p)$, 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. For each time slot in $\mathcal{T}$, the algorithm needs at most $\mathcal{O}(np)$ time to check a single insertion position. In the worst case, no feasible insert position can be found, and all n insert positions must be considered.

5.1.2. 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 that yields the smallest cost increase is found by considering all possible insert positions. The cheapest insert among all customers is performed, that is, 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 customers. 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 higher than that of the first-insert search, as evidenced by the larger time complexity $\mathcal{O}(n^4p + |\mathcal{T}|n^2p)$. Indeed, iteratively finding the cheapest insert among all $\mathcal{O}(n)$ customers will in the worst case result in $\mathcal{O}(n^3)$ overall insertion positions to be checked, each requiring a worst case of $\mathcal{O}(np)$ time to check feasibility, resulting in $\mathcal{O}(n^4p)$. After this, the first-insert search on the resulting schedule requires worst-case $\mathcal{O}(|\mathcal{T}|n^2p)$ time. Greedy construction has larger time complexity than the first-insert search. 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.

5.2. 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, whereas the other does not. They are (i) a cheapest insert search like Campbell and Savelsbergh (2006), Yang et al. (2016), and Köhler, Ehmke, and Campbell (2020) and (ii) a greedy construction heuristic as is used for the 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 time complexity of this procedure is $\mathcal{O}(n^2p)$, as in the worst-case $\mathcal{O}(n)$ insert positions must be checked of which each cost $\mathcal{O}(np)$ time. 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.

Second, we can also apply the greedy construction heuristic provided in Section 5.1 as a validation procedure. Now, only the selected time slot is considered instead of all potential time slots in $\mathcal{T}$, and the resulting time complexity is $\mathcal{O}(n^4p + n^2p) = \mathcal{O}(n^4p)$. 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.

6. 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 the quality is defined by the sum of the average travel times per arc as before. 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 procedures, a greedy randomized adaptive search and a neighborhood search.

6.1. Greedy Randomized Adaptive Search

The greedy randomized adaptive search (GRASP) works almost identically to the greedy construction heuristic described in Section 5.1. 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 time complexity $\mathcal{O}(n^4 p + n^3 \log l)$, with the first term representing the time complexity of the greedy construction heuristic and the second term representing the time needed to keep the list of l best insert positions. 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. The GRASP of Yang et al. (2016) selects customers at random rather than selecting from a list of best insertions.

6.2. Neighborhood Search

Second, 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 consists 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 (2020), which has time complexity $\mathcal{O}(n^3 k^2 p)$. In short, this complexity arises from worst-case checking $\mathcal{O}(n^2 k^2)$ possible exchange moves each costing $\mathcal{O}(np)$ time to check.

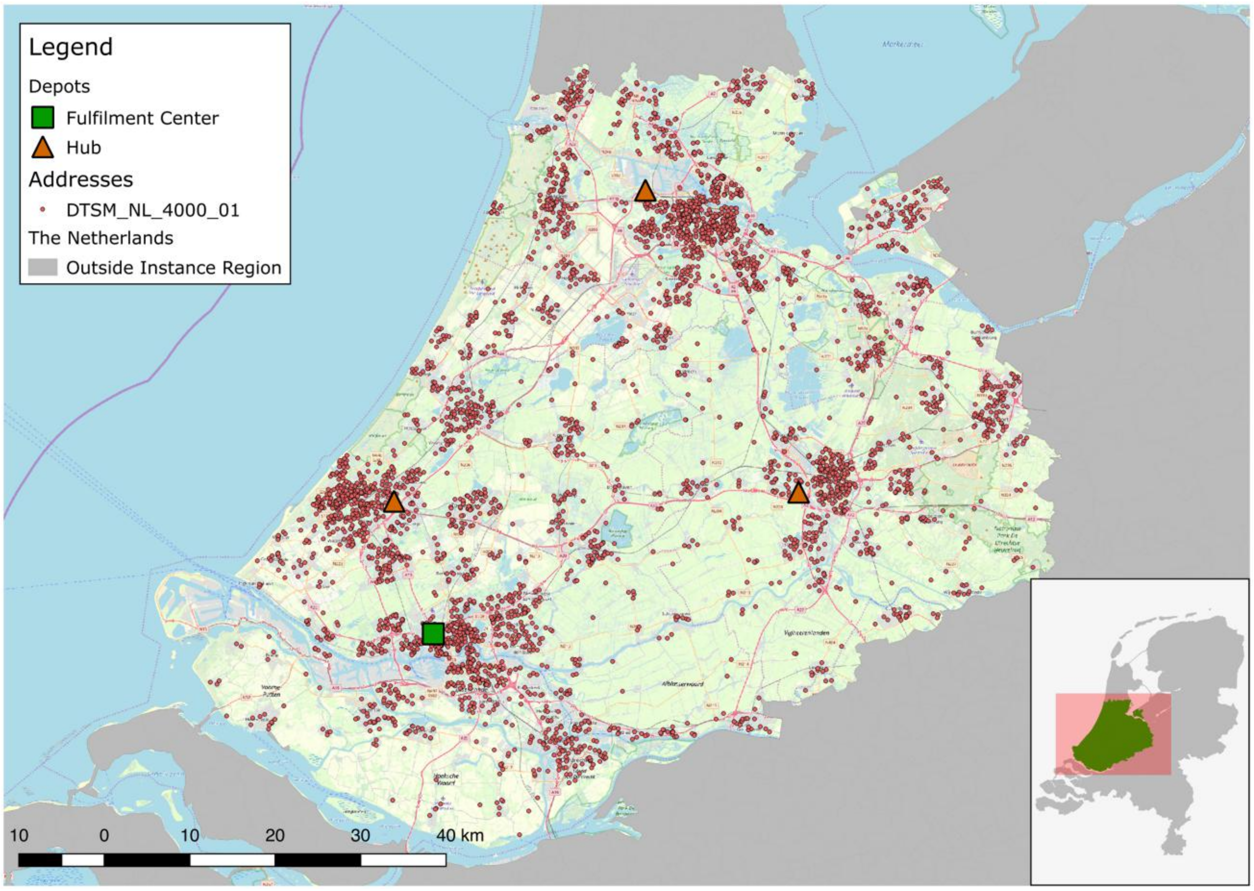
7. Data and Instances

For our numerical experiments, we build instances based on real-world online retail data provided by Ortec BV. To test our solution and approaches in different possible environments, we present a number of computer simulations in Section 8 for different instances. In particular, our instances consist of a sequence of customer arrivals. We randomly draw different numbers of customers from a service region which includes the four largest cities in the Netherlands (Amsterdam, Rotterdam, The Hague, and Utrecht), various surrounding urban satellite cities and towns, and rural areas. There are four depots, each located with easy highway access near a major city, and each with a fleet of identical vehicles. One depot is a so-called *fulfilment center*, which is the main warehouse facility where orders are picked and where vehicles are stationed. The three other depots are *hub* locations, which are not warehouses but transfer locations. There are also vehicles stationed at each hub. Figure 2 illustrates the locations of the depots, as well as 4,000 customers for an example instance.

We consider a morning shift with a depot time window of [0600, 1500]. The set of time slots is T = {[0700, 0800], [0800, 1400], [0800, 1000], [0900, 1100], [1000, 1200], [1100, 1300], [1200, 1400]}. These time slots are overlapping and include three different lengths. We assume that all customers arrive before the cutoff time. As the online grocery retailer in our specific case uses different time slot prices to balance demand over time, we assume that all time slots are equally popular. In particular, we model customer preferences as follows. Each customer has an ordered set of time slot preferences $\mathcal{T}_i^p$ containing $|\mathcal{T}_i^p| = 2$ time slots, which are drawn uniformly from the set of time slots $\mathcal{T}$. Because of the overlap and different widths among the time slots, this popularity is not evenly spread over time across the planning horizon [0600, 1500]. We only model the initial time slot choice and do not explicitly model what happens if the customer learns after validation that the selected time slot is invalid. One interpretation of this is that we make the conservative assumption that a customer leaves if his or her preferred time slot choice turns out to be invalid. The service duration per customer is randomly generated according to an empirical distribution.

We use a standard procedure, like in Ichoua, Gendreau, and Potvin (2003), to model time-dependent travel times. We obtain a nominal travel time on each arc using *OSRM*, an open-source routing service for *openstreetmap*, which represents the travel time at a

Figure 2. (Color online) Locations of an Instance with 4,000 Customers



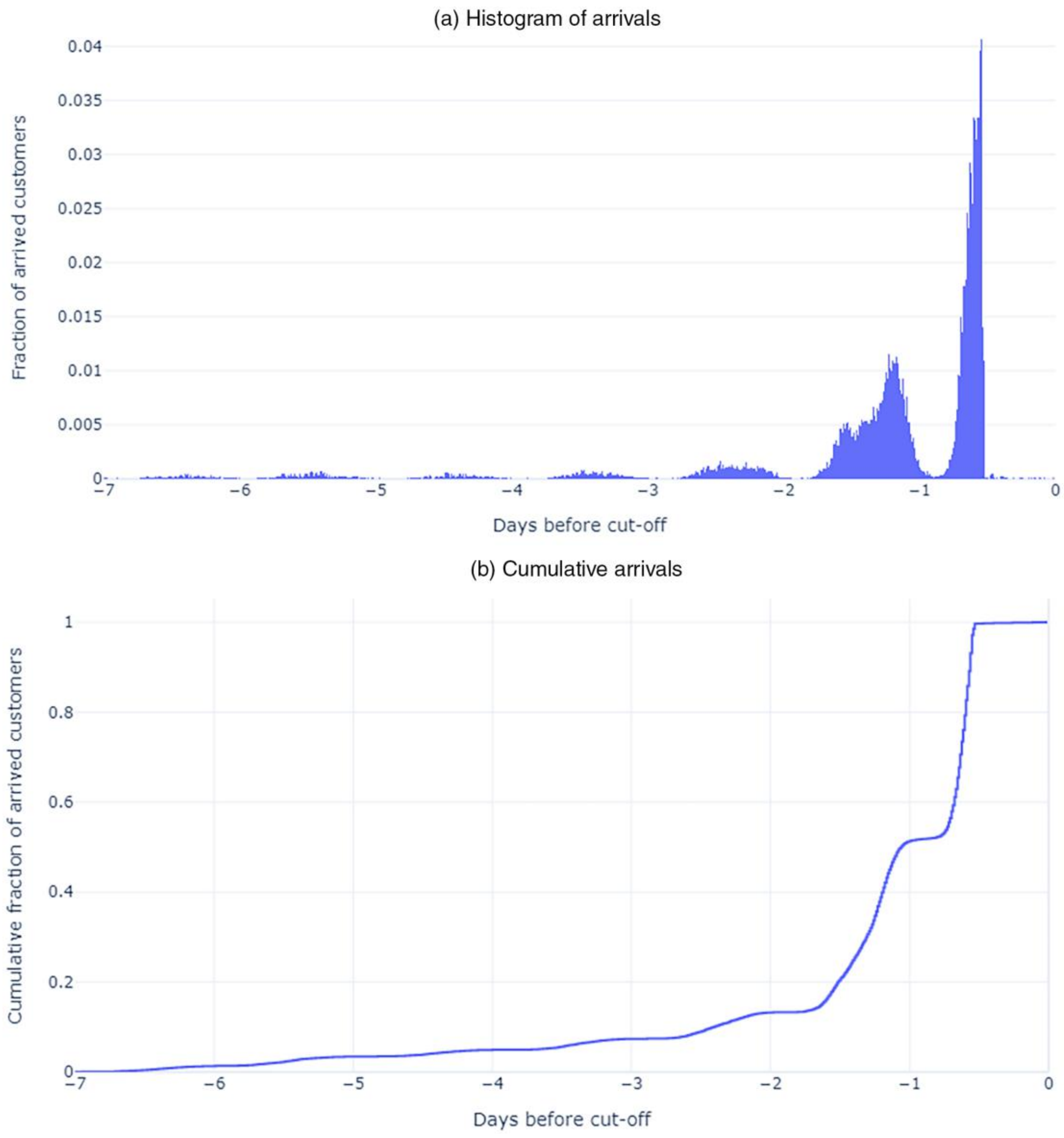
nominal speed and which we round to minutes. Furthermore, we use the following speed profiles: nominal speed at times in [0600, 0700], half speed in [0700, 1000], and nominal speed again in [1000, 1500]. This speed profile models a congestion period in between free flow. The same speed profile is used for all arcs in our network.

We model the preferences of customers for time slots by using a general nonparametric rank-based choice model (van Ryzin and Vulcano 2014). This means that we assume a customer has an ordered set of preferred time slots $\mathcal{T}_i^{\mathrm{p}} \subseteq \mathcal{T}$, and it selects the first time slot in $\mathcal{T}_i^{\mathrm{p}}$ that is also in $\mathcal{T}_i$. If $\mathcal{T}_i^{\mathrm{p}} \cap \mathcal{T}_i = \emptyset$, the customer leaves without selecting a time slot.

Figure 3 shows the empirical arrival time distribution. In Figure 3(a), a histogram is shown that provides the fraction of the total number of customers that placed an order over time. In Figure 3(a), we show the cumulative percentage of arrivals over time. Customers that did not place an order are not included in this figure. Figure 3(a) demonstrates that roughly half of all the orders are placed in a very short amount of time. From the figure, it can also be inferred that the interarrival time decreases toward the cutoff time. This can be explained because it is convenient for most customers to place their order as late as possible. Typically, the number of accepted orders also increases over time and is highest around the cutoff time. This means that the decision times are also longer because the underlying routing problems we need to solve to check the capacity are larger. As a result, we observe a combination of unfavorable circumstances near the cutoff time: customers arrive with short interarrival times, whereas capacity is scarce, and the decision times are high. This suggests that most waiting time and invalid orders are likely to occur in this period. This means that in testing our methods it is mostly relevant to look at what happens in these (final) peak periods.

To disentangle the effects of the different individual drivers, we first present a set of controlled experiments in which we control the interarrival and selection times. For a given experiment, we fix the parameter to a particular constant value. By varying the different parameter values one at a time while reducing random stochastic noise, we can get more insights into the impact of each parameter. For the same reason we also

Figure 3. (Color online) Histogram of Arrivals (a) and Cumulative Fraction of Arrived Customers (b)



fix the demand per customer to a constant. One interpretation of this setup is that we specifically focus on the interarrival times around the peak times at the end of the booking period, which can be considered fairly constant. In Section 8.6, we also present experiments in which we use varying interarrival times, demand and selection times per customer, which are randomly generated according to realistic distributions. We want to emphasize that using the empirical distributions, we can see very low interarrival times for these realistic instances. For example, in realistic instance 1, the minimum interarrival time is 3.5 ms, and there is a substantial number of arrivals with interarrival times in the range of 10–100 ms. As explained previously, these low interarrival times occur at a crucial period of time with limited remaining capacity and high decision times.

We generated 10 instance sets with 2,000, 4,000, and 8,000 arriving customers, which gives a total of 30 instance sets. The demands of all customers are equal, and each vehicle has a capacity that allows it to carry the demand of 33 customers. In Table 1, we summarize the number of vehicles stationed at the fulfillment centers and hubs for the different instance sizes. Within each instance set, we consider the following constant interarrival times Δ: 1 μs, 1 ms, 10 ms, 100 ms, 1 second, and 10 seconds. As stated previously, for the empirical arrival time distributions, interarrival times in the order of 1 ms and upward are observed. Furthermore, preliminary experiments show that our decision times, and therefore also the response times, can be very low, that is, less than 1 ms. We highlight here that in absolute terms, our response times are perhaps

Table 1. Number of Vehicles Available and Total Capacity in Number of Customers

	No. of vehicles			
n	Total	Fulfillment	Hub	Total capacity (no. of customers)
2,000	50	20	10	1,650
4,000	100	40	20	3,300
8,000	200	80	40	6,600

underestimates of real-life response times, because these also include, for example, time for communication over the Internet. Hence, relatively speaking, 1 μs represents the case that interarrival times are lower than response times, which is not unrealistic. Finally, 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. Furthermore, 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 of 0 seconds, which means the customer immediately selects a time slot or leaves after the time slot offer is presented.

8. Computational Experiments

In this section, we present the results of our computational experiments. In Section 8.1, we report on the decision and response times of our algorithms. Similarly, we report the number of valid and invalid orders for these instances in Section 8.2. 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 8.3, which has implications for the number of invalid orders that can be expected for larger instances. In Section 8.4, we report on the number of valid and invalid orders, and response times for larger instances. Finally, we provide the results of our experiments on instances in which also the interarrival times, selection times and demands vary stochastically in Section 8.6.

We report the decision and response times of the various algorithms presented in this paper, and the number of valid orders achieved by these algorithms. 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, whereas for the background procedure, there is a third option of not using any algorithm. One might also combine options. In particular, in our experiments we consider a validation procedure that first performs a cheapest insertion and afterward performs a greedy construction. Table 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, 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 because 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. The DTSM configurations are essentially multithreaded, but our custom discrete event simulator allows us to simulate a multithreaded configuration using only one thread. This way, we avoid the CPU time measurements to include possible overhead that is specific to a multithreaded/parallel implementation and the used CPU architecture. Only this one thread was run at any time on the CPU.

8.1. 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, for the following reason. 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,

Table 2. Configurations Investigated in This Paper

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

and increases as more valid orders are placed, according to the time 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 procedures and validation procedures, averaged over 10 instances. The column Interarrival 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.

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 longer 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 longer. For shorter interarrival times, the average maximum response times of all time slot offer procedures are equal to their decision times. Moreover, these times are short, 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. Therefore, 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 nine minutes, for INSCON it is almost three minutes, whereas 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.

8.2. 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, that is, 0 or 30 seconds. Again, the column Interarrival provides the interarrival time Δ between consecutive customers. These results are based on a detailed simulation of the interaction between decision times, interarrival times, and selection times. Preliminary experiments showed the relevance of explicitly taking the decision times into account, even when the selection times are relatively long. Comparing the results

Table 3. Decision and Response Times (min:s:ms) for Instances with 2,000 Customers and a Selection Time of 30 Seconds

	Time slot offer						Validation					
	Max. decision			Max. response			Max. decision			Max. response		
Interarrival time (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 4. Number of Valid and Invalid Orders for Instances with 2,000 Customers

		No. of valid orders					No. of invalid orders				
Sel. (s)	Interarrival time (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

with and without decision times, we found that the realistic decision times have a significant impact on the results in terms of invalid orders.

We first comment on the number of valid orders. For the configurations without the background procedures, that is, 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, that is, 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 to outperform the other methods in terms of response time and number of valid orders when the interarrival time is sufficiently large. We observe that the number of valid orders is not affected by selection time.

The number of invalid orders, due to concurrency conflicts, is substantially affected by interarrival and selection times for all configurations. We observe that the number of invalid orders decreases with an increasing interarrival time. 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. When the selection time is 30 seconds, 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 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 sequence of arrival, so neither is their sequence 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 as the selection time is increased from 0 to 30 seconds, whereas in the other 14 cases, a substantial increase is observed.

8.3. 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 for 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 second, and 10 seconds, 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.

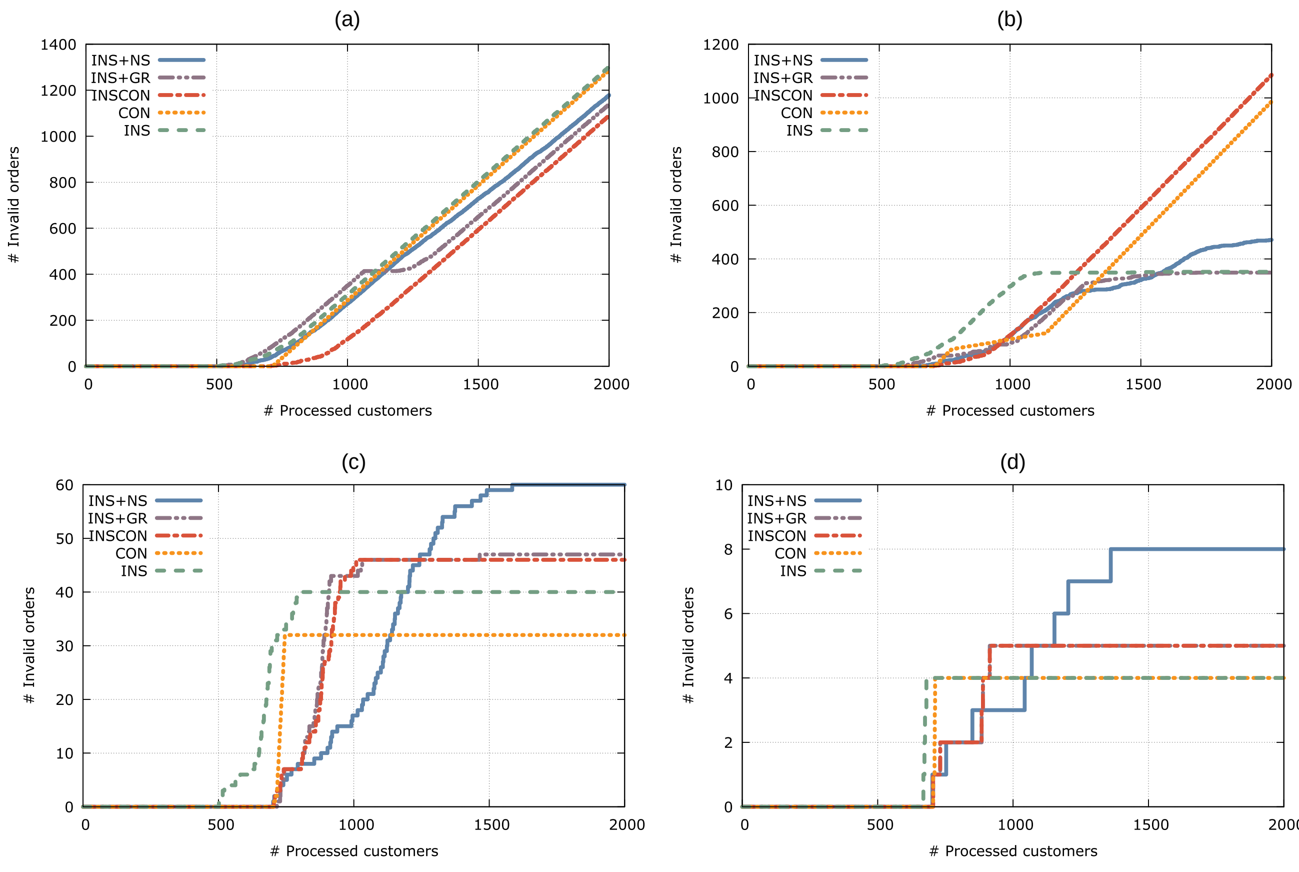
In case of an interarrival time of 10 ms in Figure 4(a), we observe a steady increase in the number of invalid orders until all customers are processed. For the other interarrival times, and almost all configurations, the number of invalid orders stops increasing at some point. This happens when it is 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.

As the configurations that use a background procedure, that is, 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, we see new valid orders again and also new invalid orders. This effect can most clearly be seen in Figure 4(d) for INS+NS.

8.4. 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. As shown in Table 1, not only the number of customers is larger in these instance sets, the capacity is also proportionally larger.

Figure 4. (Color online) Number of Invalid Orders for an Instance with 2,000 Customers with Interarrival Times



Note. (a) 10 ms, (b) 100 ms, (c) 1 second, and (d) 10 seconds.

Table 5. Results for the Large Instance Sets with 30-Second Interarrival Time

		No. of valid orders			No. of invalid orders			TSO max. response (ms)		
$\vert\mathcal{C}\vert$	Interarr. (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 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). Column $|\mathcal{C}|$ provides the amount of arriving customers, and once more, column Interarrival provides the interarrival time Δ between consecutive customers.

First, we observe that the average maximum 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. The response times of the validation procedures are larger than those of the time slot offer procedures. Although we do not report the average maximum response times of the validation procedure in Table 5, the largest value is observed for instances with 8,000 customers and interarrival times of 0.001 ms, which is three seconds 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 8.3, 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 8.3 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, the number of invalid orders also increases.

8.5. Impact of Background Procedures on the Route Schedules

Our background procedures can significantly reduce the decision times and response times as reported in Table 3 and increase the number of valid orders as seen in Table 4. In addition to these benefits, we can also improve the quality of the route schedules in memory by using the background procedures. This relates to the travel distances per order as these are associated with fulfilment costs, fuel consumption, and emissions. The quality of the route schedule at the end of the order intake period is especially relevant when the retailer wants to start order picking and execution based on this schedule immediately after the cutoff time. Table 6 provides an overview of the average number of kilometers per order, averaged over the instances in the instance set. As before we only include INS, INS+GR, and INS+NS. For illustrative purposes we present the results for an interarrival time of one second. The results show that the background procedures create route schedules with substantially lower travel distances per order.

8.6. Realistic Instances

Next, we present the results of our experiments on the realistic instances, in which demand, interarrival times, and selection times are stochastic. We use 10 instances with $n = 4{,}000$ customers for this purpose, which are generated as described in Section 7. We assume a capacity of 60 vehicles at the fulfilment center and 20 vehicles at each hub. We use an empirical distribution for demand per customer, for the service duration per customer, and for the customer arrival process as shown in Figure 3. These data are only available for customers that actually placed a valid order. This

Table 6. Valid Orders and Average Travel Distance per Order

	No. of valid orders			Kilometers per order		
$\lvert\mathcal{C}\rvert$	INS	INS+GR	INS+NS	INS	INS+GR	INS+NS
2,000	742.4	931.8	925.9	15.9	10.8	7.6
4,000	1,506.8	2,235.1	2,705.9	14.2	8.1	6.0
8,000	3,403.8	5,177.2	5,091.3	12.1	6.1	6.9

means, for example, that in our experiment the interarrival times are underestimates of real-world interarrival times. We generate the selection time in seconds using a uniform distribution on $[0, 60]$ based on average values available to us. We apply the configuration INS+NS, which was arguably the best performing method in our previous experiments.

The average maximum response times for the time slot offer and the validation are negligibly close to zero for this configuration in these instances. This suggests that our best concurrency control methods perform well in realistic instances. In Table 7, for each of the 10 instances, we provide the number of valid and invalid orders. The number of invalid orders varies between 10 and 24, with an average of 16.5. On average, less than 1% of the customers will experience an invalid order and thus find out that their selected time slot is no longer available. All these invalid orders occur during a relatively short time period. Figure 5 illustrates this for instance 1. All invalid orders occur during a peak in the number of arrivals, whereas at the same time, the number of valid orders and hence decision times are at their highest, and capacity is most scarce.

9. Concluding Remarks and Future Research Directions

In this paper, we introduce the problem of concurrent customer interactions in online booking systems for attended home delivery. We argue that there is a fundamental tradeoff between the number of valid and invalid orders and waiting times, when allowing customers to choose their time slots. Our experiments focused primarily on maximizing the number of valid orders while limiting waiting times and invalid orders. Our results show that invalid orders and waiting times due to concurrent interactions are inevitable in this case, even with fast state-of-the-art methods. Moreover, our detailed experiments with different numbers of customers, interarrival times, and selection times shed light on the drivers of concurrent interactions and the different tradeoffs in the design of concurrency control strategies. For example, our experiments suggest that using a background procedure increases the number of customers that can be accommodated and leads to acceptable response times. However, this success comes at the cost of having more invalid orders.

As we are one of the first to explore the concept of concurrency control in time slot management, there are many avenues for future research. Here, we discuss three relevant directions. In Section 9.1, we discuss research directions for pessimistic concurrency control, in Section 9.2 we discuss research direction for optimistic concurrency control, and in Section 9.3, we discuss alternative time slot choice mechanisms. We conclude in Section 9.4.

9.1. Pessimistic Concurrency Control

Pessimistic concurrency control prevents conflicts that are caused by concurrent interactions with the system. For DTSM, pessimistic concurrency control means enforcing that we do not concurrently interact with more customers than can be accommodated by the available capacity. In Section 4, we provided a natural example of a pessimistic concurrency control strategy by letting customers interact sequentially with the system. This is clearly not viable in many applications due to the long waiting times. However, it may be possible to design more sophisticated pessimistic concurrency control methods with less waiting times.

To prevent invalid orders, we could choose to not offer all currently feasible slots but only offer those slots that are feasible in any specific scenario that can unfold given the current state of the system. The fundamental question is as follows. Is it feasible to visit the newly arrived customer during a specific time slot, given the current valid orders, for every realization of the response time, selection time, and time slot selection of every customer currently interacting with the system? If the answer is yes, there is no risk of invalid orders, and the time slot can be offered. If the answer is no, there is a second question to ask, namely whether there is any such realization in which the customer can be visited during the time slot under consideration. If the answer is again no, the time slot is not offered. If the answer is yes, we may decide to wait. These questions seem to fit well with the paradigm of robust optimization. This gives rise to questions on how to best model

Table 7. Results of Using NS+GR on 10 Realistic Instances with 4,000 Customers

Instance	No. of valid orders	No. of invalid orders
1	3,238	18
2	3,239	14
3	3,277	17
4	3,235	13
5	3,244	20
6	3,230	19
7	3,247	24
8	3,259	17
9	3,184	10
10	3,213	13
Average	3,236.6	16.5

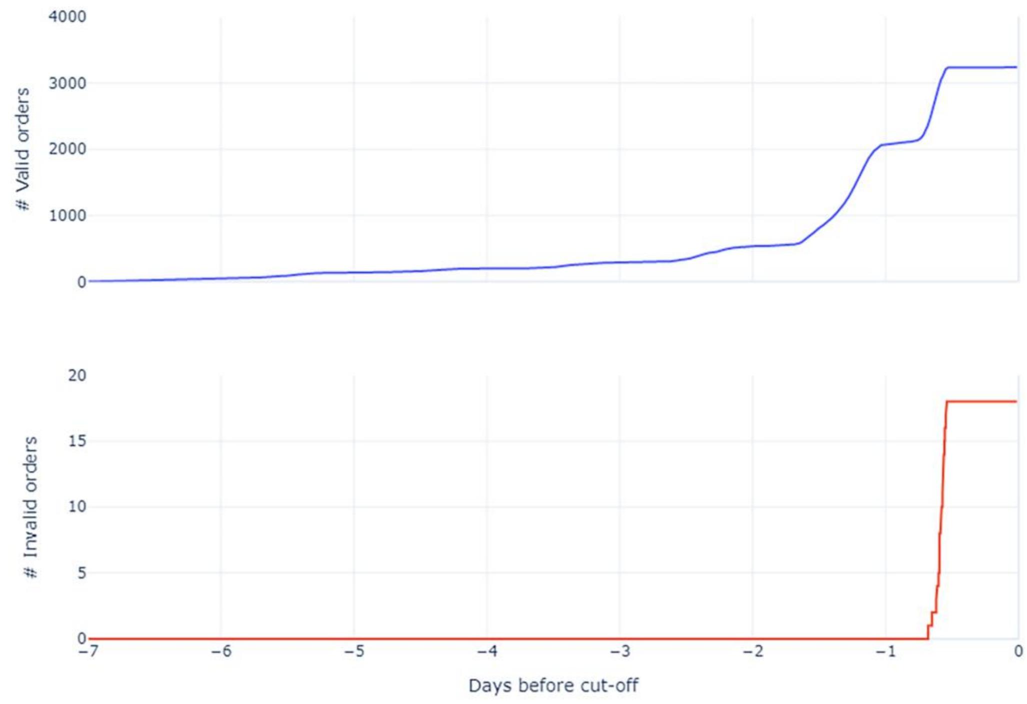


Figure 5. (Color online) Number of Valid and Invalid Orders over Time

the uncertainty sets to appropriately describe the set of possible realizations of response time, selection time, and time slot selection. Another interesting methodological avenue of research involves finding fast computational approaches to solve the underlying optimization problems. This involves exploring tradeoffs between the modeling accuracy of the uncertainty sets and computation times. Very detailed models of the state space may not be tractable given the combinatorial nature of the underlying routing problems and the complex and intricate uncertainties based on the interaction of different parts of the model. Furthermore, it is shown by van der Hagen et al. (2024) that machine learning approaches can quickly assess the feasibility of accepting a certain delivery order in a certain time slot given the already accepted orders. It is interesting to study how machine learning approaches can be used to predict and prevent concurrency conflicts.

In this context, it is also interesting to explore different ways to interact with customers. Do we need to provide the whole menu of possible time slots simultaneously or can we buy more time by incrementally revealing possible time slots? That is, for some time slots it may be quickly determined that there is no risk of an invalid order, whereas for others, we need more time. This gives rise to new modeling questions but also questions on customer preferences and behavior. It is, for example, relevant to understand how customers respond to waiting times. How long do customers wait before dropping out? How does waiting affect future sales and choice behavior? Such insights help practitioners decide whether pessimistic concurrency control is beneficial or harmful based on the balance between avoiding invalid orders and the negative impact of waiting times on business.

9.2. Optimistic Concurrency Control

Optimistic concurrency control for DTSM does not avoid invalid orders. In Section 9.2.1, we discuss research directions on how to deal with these invalid orders when they occur. In Section 9.2.2, we discuss research opportunities focused on minimizing the expected number of invalid orders.

9.2.1. Dealing with Invalid Orders. A basic mechanism for dealing with invalid orders is to immediately inform the customer corresponding to an invalid order that their selected time slot is no longer available, and then offer a new (possibly empty) set of time slot options. Unfortunately, our study suggests that it can take a significant amount of time to determine that an order is invalid, especially during busy periods due to the interaction between different order arrivals in the system. Therefore, in practice, it may be more appropriate to cancel (or reschedule) orders after the booking period has ended, prior to route planning.

In our experiments, we define an invalid order as the incoming order that cannot be accommodated given the current set of accepted (valid) orders. However, an invalid order really only means that we cannot deliver all currently placed orders within their corresponding time slots. Therefore, there is no fundamental reason to cancel specifically the last incoming order to resolve the infeasibility of the route plan. This provides more freedom. For example, given the accumulation of orders, we may want to find the minimum number of orders to cancel to ensure a feasible schedule. Alternatively, we could minimize the total (long-term) revenue loss from cancelling orders. This gives rise to a new selective or prize-collecting variant of the vehicle routing problem with time windows.

Instead of canceling an order, we may serve a customer outside of their selected time slot. This can be done with or without explicitly coordinating this with the customer. However, without explicit coordination, there is a risk that nobody will be available to receive the delivery. Again, the question is which customers to serve later or earlier and by how much, to create a time feasible delivery schedule. This flexibility leads to new routing problems taking customer availability into account. This is related to a recent stream of work that focuses on maximizing the success rate of deliveries while minimizing the cost of additional attempts needed if the first attempt was not successful (Özarik et al. 2021, 2023; Voigt et al. 2023).

We can also try to actively reschedule customers to different time windows to create a feasible routing schedule. This gives rise to novel route optimization problems that are loosely related to the area of vehicle routing problems with soft time windows (Figliozzi 2010). Moreover, some customers are more flexible than others in terms of rescheduling their time slot appointment. One interesting question in this realm relates to finding the best sequence in which to approach customers to reschedule. Moreover, we may also offer monetary incentives or discounts to persuade customers to switch to another time window or accept a longer time window. How to best deploy incentives to maximize the effectiveness is another interesting area of study.

A customer may accept an occasional cancellation, rescheduling, or late delivery. An interesting line of research is to design time slot offer procedures and algorithms that generate the final route schedule to ensure that such actions do not occur too often for the same customer. This links the planning and control across multiple booking periods and thus creates significant computational challenges.

9.2.2. Minimizing the Risk of Invalid Orders.

If there are historical data available on order patterns and time slot preferences of customers, it may be possible to exploit this information. Although pessimistic control strategies prevent all possible concurrent interactions, even if they are unlikely to happen, we can also take a more probabilistic approach focusing on likely conflicts. This involves explicitly modeling customer and system behavior. Consider a simple example, in which we know that all customers currently in the system prefer morning slots. In that case, we may confidently offer a newly arriving customer an afternoon slot such that the probability of conflicts is low. The opposite logic also applies. If an incoming customer prefers only afternoon slots, we can confidently block the morning slots for this customer without the risk of losing this customer due to lack of appropriate choices. We can then either optimize the system given certain chance constraints or by taking into account the costs and awards in the objective function. It is not obvious how to model the different states and uncertainties in such a setting to build models and methods that are tractable and fast. One interesting avenue for future research is exploring the most suitable customer choice models and their level of detail and accuracy. Also here, we see promising opportunities for data-driven machine learning approaches to predict both customer and system behavior and to identify useful patterns and properties for concurrency control. Given the need for extremely fast methods, it may be especially relevant to explore "predict and optimize" models that take into account the time slot offering decisions in the prediction process (Elmachtoub and Grigas 2022, Vanderschueren et al. 2022).

On the capacity side, there may also be ways to reduce the risk of invalid orders. For example, we can anticipate potentially invalid orders by reserving some backup vehicles that are not visible to the system during the order intake phase. Determining the appropriate level of backup vehicle capacity is an interesting stochastic problem in this context.

9.3. Alternative Mechanisms

Some of the fundamental tradeoffs and concurrency issues associated with letting customers choose a time slot for attended home delivery in real time can be prevented by different ways of interacting with the customers.

First, online grocery retailers have started to offer subscription services (Wagner, Pinto, and Amorim 2021). Some of these subscription models allow frequent customers to subscribe to a specific delivery window, such as every Monday 0600–2000 hours for the next three months. In a hybrid system, where some customers subscribe and others arrive dynamically and sporadically, we would still encounter the concurrency problems discussed in this paper. That is, the subscription customers can be considered early-booking customers, and we still see concurrent interaction for the dynamically arriving customers. In a subscription-only system, we would likely not see concurrent interaction because subscription customers would typically not all arrive at the same time, would be willing to accept longer wait times, and may be more flexible in their choice of time slots. Such a full subscription model would only work with a repeat-purchase product such as groceries and a large base of loyal customers. One of the challenges of the subscription system is customer churn. Another challenge is subscription customers who do not use their subscribed time slot. This raises new questions. How do you deal with subscribers who do not order? Customers with a subscription can still choose not to place an order. When this happens, capacity is freed up to accommodate other customers. To know when this capacity will become available, the subscription may specify that if no order is placed before a certain time, the reservation of the time slot will be canceled. Optimistic concurrency control strategies may anticipate that this will happen for some customers and therefore allow invalid orders from dynamically arriving customers before this specified time. In this case, it remains uncertain whether an order is valid or invalid until after the specified time. New time slot offer procedures must be designed to accomplish this. Even outside the subscription service model, these research questions are relevant in settings in which it is common that customers cancel their order.

Second, it may be possible to reduce the negative impact of concurrent interactions by increasing the time flexibility in final route planning. This can be done by having a sufficient number of customers select a wider time slot or provide flexibility for multiple time slots (Strauss, Gülpinar, and Zheng 2021). This creates more wiggle room to find a feasible routing schedule after the cutoff time. This gives rise to several research questions on how to model and operate such a system and on how to preserve sufficient flexibility during the order intake phase while maximizing choices offered to the customer and thereby the number of valid orders. One possible idea would be to offer various lengths of time windows to customers at different stages of the planning period. Köhler, Ehmke, and Campbell (2020) study a system in which early in the planning stage only long time windows are offered, whereas later short time windows are offered. From a concurrency control perspective, it may be better to actually use an opposing strategy in which we offer longer time slots in the busy period just before the cutoff time. It is interesting to explore the tradeoffs that arise in such a situation in terms of routing efficiency versus the risks of invalid orders.

9.4. Concluding Remarks

Despite the growing attention for dynamic time-slot management in the academic literature, the problems of concurrent interactions in online time-slot reservation systems are only now surfacing with large-scale implementation in practice. In this paper, we introduce the problems of concurrent interaction, show the need for new methods to prevent and deal with these problems, and point to relevant future research directions.

References

Abdollahi M, Yang X, Nasri MI, Fairbank M (2023) Demand management in time-slotted last-mile delivery via dynamic routing with forecast orders. *Eur. J. Oper. Res.* 309(2):704–718.

Agatz N, Fan Y, Stam D (2021) The impact of green labels on time slot choice and operational sustainability. *Production Oper. Management* 30(7):2285–2303.

Agatz N, Campbell AM, Fleischmann M, Savelsbergh MWP (2011) Time slot management in attended home delivery. *Transportation Sci.* 45(3):435–449.

Agatz N, Campbell AM, Fleischmann M, van Nunen J, Savelsbergh MWP (2013) Revenue management opportunities for Internet retailers. *J. Revenue Pricing Management* 12(2):128–138.

Ausseil R, Pazour JA, Ulmer MW (2022) Supplier menus for dynamic matching in peer-to-peer transportation platforms. *Transportation Sci.* 56(5):1304–1326.

Bernstein PA, Hadzilacos V, Goodman N (1987) *Concurrency Control and Recovery in Database Systems*, vol. 370 (Addison-Wesley, New York).

Campbell AM, Savelsbergh MWP (2005) Decision support for consumer direct grocery initiatives. *Transportation Sci.* 39(3): 313–327.

Campbell AM, Savelsbergh MWP (2006) Incentive schemes for attended home delivery services. *Transportation Sci.* 40(3):327–341.

Chen HR (2009) An evaluation of real-time transaction services in web services e-business systems. *Advances in Data and Web Management* (Springer, Berlin), 532–537.

Cleophas C, Ehmke JF (2014) When are deliveries profitable? *Bus. Inform. Systems Engrg.* 6(3):153–163.

Ehmke JF, Campbell AM (2014) Customer acceptance mechanisms for home deliveries in metropolitan areas. *Eur. J. Oper. Res.* 233(1):193–207.

Elmachtoub AN, Grigas P (2022) Smart "predict, then optimize". *Management Sci.* 68(1):9–26.

Figliozzi MA (2010) An iterative route construction and improvement algorithm for the vehicle routing problem with soft time windows. *Transportation Res. Part C Emerging Tech.* 18(5):668–679.

Fleckenstein D, Klein R, Steinhardt C (2023) Recent advances in integrating demand management and vehicle routing: A methodological review. *Eur. J. Oper. Res.* 306(2):499–518.

Gendreau M, Ghiani G, Guerriero E (2015) Time-dependent routing problems: A review. *Comput. Oper. Res.* 64:189–197.

Graefe G (2019) On transactional concurrency control. *Synthetic Lecture Data Management* 14(5):1–404.

Ichoua S, Gendreau M, Potvin JY (2003) Vehicle dispatching with time-dependent travel times. *Eur. J. Oper. Res.* 144(2):379–396.

Khaing KK, Myint MM (2017) Data consistency and concurrency controlling in air ticket selling in different sites by using notification-reread method (NRM). PhD thesis, MERAL Portal, University of Computer Studies, Yangon, Myanmar.

Kindervater GAP, Savelsbergh MWP (1997) Vehicle routing: Handling edge exchanges. Aarts E, Lenstra JK, eds. *Local Search in Combinatorial Optimization*, 1st ed. (John Wiley & Sons, New York), 337–360.

Koch S, Klein R (2020) Route-based approximate dynamic programming for dynamic pricing in attended home delivery. *Eur. J. Oper. Res.* 287(2):633–652.

Köhler C, Ehmke JF, Campbell AM (2020) Flexible time window management for attended home deliveries. *Omega* 91:102023.

Lewandowski G, Bouvier DJ, McCartney R, Sanders K, Simon B 2007 Commonsense computing (episode 3) concurrency and concert tickets. *Proc. 3rd Internat. Workshop Comput. Ed. Res.* (ACM, New York), 133–144.

Liu N, Van De Ven PM, Zhang B (2019) Managing appointment booking under customer choices. *Management Sci.* 65(9): 4280–4298.

Özarik SS, Veelenturf LP, Van Woensel T, Laporte G (2021) Optimizing e-commerce last-mile vehicle routing and scheduling under uncertain customer presence. *Transportation Res. Part E Logist. Transportation Rev.* 148:102263.

Özarik SS, Lurkin V, Veelenturf LP, Van Woensel T, Laporte G (2023) An adaptive large neighborhood search heuristic for last-mile deliveries under stochastic customer availability and multiple visits. *Transportation Res. Part B Methodological* 170:194–220.

Strauss A, Gülpinar N, Zheng Y (2021) Dynamic pricing of flexible time slots for attended home delivery. *Eur. J. Oper. Res.* 294(3):1022–1041.

van der Hagen L, Agatz N, Spliet R, Visser TR, Kok L (2024) Machine learning–based feasibility checks for dynamic time slot management. *Transportation Sci.* 58(1):94–109.

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

Vanderschueren T, Verdonck T, Baesens B, Verbeke W (2022) Predict-then-optimize or predict-and-optimize? An empirical evaluation of cost-sensitive learning strategies. *Inform. Sci.* 594: 400–415.

Visser TR, Spliet R (2020) Efficient move evaluations for time-dependent vehicle routing problems. *Transportation Sci.* 54(4): 1091–1112.

Voigt S, Frank M, Fontaine P, Kuhn H (2023) The vehicle routing problem with availability profiles. *Transportation Sci.* 57(2):531–551.

Waßmuth K, Köhler C, Agatz N, Fleischmann M (2023) Demand management for attended home delivery: A literature review. *Eur. J. Oper. Res.* 311(3):801–815.

Wagner L, Pinto C, Amorim P (2021) On the value of subscription models for online grocery retail. *Eur. J. Oper. Res.* 294(3):874–894.

Yang X, Strauss AK (2017) An approximate dynamic programming approach to attended home delivery management. *Eur. J. Oper. Res.* 263(3):935–945.

Yang X, Strauss AK, Currie CSM, Eglese R (2016) Choice-based demand management and vehicle routing in e-fulfillment. *Transportation Sci.* 50(2):473–488.