August 2020

Reevaluating Order Fulfillment Decisions for E-Tailers Under True Simulated Operating Conditions

Amir H. Kalantari
University of Wisconsin-Milwaukee

REEVALUATING ORDER FULFILLMENT DECISIONS FOR E-TAILERS UNDER TRUE SIMULATED OPERATING CONDITIONS

by

Amir Kalantari

A Dissertation Submitted in

Partial Fulfillment of the

Requirement for the Degree of

Doctor of Philosophy

in Engineering

at

The University of Wisconsin – Milwaukee

August 2020

ABSTRACT

REEVALUATING ORDER FULFILLMENT DECISIONS FOR E-TAILERS UNDER TRUE SIMULATED OPERATING CONDITIONS

by

Amir Kalantari

The University of Wisconsin – Milwaukee, 2020
Under the Supervision of Professor Matthew Petering

This dissertation makes both a methodological and an applied contribution. From a methodological standpoint, this is among the very first works in the literature to explore the concepts of true simulated operating conditions and fully embedded decision-making algorithms. We illustrate the effectiveness of these concepts by applying them to an online retailer (i.e. e-tailer) order fulfillment decision making process.

Online shopping has completely transformed retail markets in recent years. For customers, it provides convenience, visibility and choice, and for retailers it provides market expansion opportunities, operational cost reduction, and many other advantages. There are fundamental differences between the supply chain design and operations of an online and traditional (i.e. brick and mortar) retailer. One of the key differences exists in customer order fulfillment which refers to the process of picking and packing order items from a retailer's warehouse or store and delivering them to customers. In traditional retail, order fulfillment happens in physical stores and by customers. In online retail, however, the tables are turned, and the retailer is responsible for this task.

The reliability, cost, and lead time of online order fulfillment have a direct impact on customer satisfaction and an e-tailer's overall success. In today's competitive market, excellence in fulfillment is critical and organizations are struggling with how best to accomplish this while remaining profitable. On one hand, order fulfillment accounts for a considerable amount of operational cost and reducing it directly improves an e-tailer's bottom-line. On the other hand, customers demand fast and cheap order delivery options. This constantly pushes e-tailers to make tough strategic and operational choices to stay competitive.

An e-tailer's order fulfillment process begins with a fulfillment decision which assigns a customer order to one or more fulfillment centers (FCs). E-tailers typically put an order fulfillment policy (i.e. fulfillment strategy) in place that determines how those decisions must be made. Identifying the best policy is extensively studied in the literature. However, most of the proposed policies focus on minimizing the fulfillment cost for individual customer orders by finding an optimal assignment at the time an order is placed. In this dissertation we show that this policy leads to a suboptimal decision at the system level. In other words, when a collection of these myopic fulfillment decisions is analyzed together, total fulfillment cost can be further reduced by optimizing the decisions for that group collectively.

Since e-tailers receive customer orders around the clock and at a fast pace, order fulfillment decisions are made automatically using an algorithm. Additionally, from an operational perspective, making fulfillment decisions on the fly for individual customer orders enables e-tailers to keep an updated available-to-promise inventory record for each stock keeping unit (SKU) and FC combination. It also allows them to provide an estimated delivery window to their customers in real time. Therefore, although in theory optimizing fulfillment decisions for a group of customer

orders reduces costs, there are practical challenges in deploying this policy in a real-world e-tailer environment.

In order to address these challenges, we propose a reevaluation strategy that does not fully replace the automated order fulfillment decision making process. Instead, it periodically reevaluates and optimizes the fulfillment decisions for a group of orders that are waiting in the system to be processed and shipped to customers. We develop an integer programming-based reevaluation algorithm that can be triggered for a fixed number of customer orders or at regular time intervals. Our integer program considers several dimensions such as on-hand and on-order inventory, customer delivery preferences, shipping methods, and the number of boxes to minimize total fulfillment cost while maintaining the delivery time and service level for all customer orders. Additionally, since the large instances of the proposed model are mathematically difficult to solve to optimality, we develop a decomposition-based heuristic for those instances.

As noted, our proposed reevaluation algorithm must be triggered regularly during an e-tailer's operations without interrupting other important processes relating to new customer orders, shipment of orders, and inventory replenishment. Therefore, in addition to reevaluation decisions, the computation time used by a reevaluation algorithm needs to be considered when designing an effective strategy. For example, for customer orders that need to be shipped on a given day, reevaluation decisions must be finalized before the shipping deadline.

To study the complex relationship between reevaluation and other processes, we embed our reevaluation algorithm inside a discrete event simulation model in such a way that both the decisions produced and computation time used by the algorithm are fed back to the simulation model. This novel method which was first presented by Petering (2015), enables us to study the tradeoff between the quality of the decisions produced and computation time used by the algorithm

in order to recommend the overall best reevaluation strategy for an e-tailer according to its operational characteristics.

Finally, we conduct more than two hundred experiments in which the reevaluation algorithm is fully embedded in the DES model. The results confirm the effectiveness of reevaluation algorithm in reducing total fulfillment cost by an average of 5% for our test instances. It also illustrates the tradeoff between decision quality and computation time and allows us to perform scenario analysis to find the best overall reevaluation strategy for an e-tailer.

To

My parents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my gratitude towards many individuals whose help and assistance made it possible for me to accomplish this work. First and foremost, I would like to formally thank my PhD advisor and committee chair, Dr. Matthew Petering for helping and supporting me over the years. He has been very inspirational to me and without his perpetual guidance, support and motivation, this would not have been possible. I would also like to thank and acknowledge the effort of Dr. Hamid Seifoddini, Dr. Jaejin Jang, Dr. Wilkistar Otieno and Dr. Christine Cheng for their invaluable insight, kind help and great comments. I have learned a great deal from them, and I feel very much indebted. I am grateful to all of those with whom I have had the pleasure to work during my PhD. Last but not the least, I would like to thank my family. There are no proper words to convey my deep gratitude and respect for their heart-warming emotional support.

Chapter 1

Introduction

The online retail (e-tail) industry has grown substantially during the past few decades. In the second quarter of 2019, the U.S. Department of Commerce reported that the estimate of U.S. retail e-commerce sales was $146.2 billion, which shows an increase of 4.2 percent from the first quarter of that year. This is while the total retail sales for the second quarter of 2019 was reported at $1,361.8 billion which means that e-commerce retail accounted for about 10.7 percent of the total retail sales in the U.S. Although this market share seems small, e-tail has been steadily growing year over year and is projected to continue with the same trend in the coming years (U.S. Department of Commerce, 2019). A similar retail transformation seems to be taking place in other parts of the world (O'Grady and D'Costa, 2019; Khan et al., 2013; Geng and Li, 2019) According to Statista, in 2019 retail e-commerce sales worldwide amounted to $3.53 trillion and its revenue is projected to grow to $6.54 trillion in 2022 (Statista, 2019).

The internet enables retailers to increase their sales and market share and to generate new business by offering new services (De Koster, 2003). It also provides consumers with more information and alternatives to help them with their product discovery and final purchase (Gao and Su, 2016). The availability, convenience and competitive pricing of e-tailers have also contributed to their growing popularity. Although the early e-tailers, such as Amazon and eBay, operated their entire business online, this market has evolved over time and nowadays many traditional retailers and manufacturers such as Walmart and Apple have moved a significant portion of their sales to the e-commerce channel. De Koster (2003) identifies four types of companies that sell products online to consumers: (i) product manufacturers such as DELL,

Unilever and Numico; (ii) traditional retailers and wholesalers, such as Barnes & Noble, Albert Heijn and Tesco; (iii) new internet companies without physical assets such as eBay; (iv) new internet companies, with physical assets such as Amazon, Peapod, and Maxfoodmarkets.

As companies move their sales to the e-commerce channel, one of the key decisions is how to design an effective supply chain network to deliver goods to customers with minimum cost and maximum reliability and service. There are fundamental differences between the supply chain structure of an e-tailer and a traditional retailer which need to be considered while making this decision. One of the main differences is in their delivery policy (De Koster, 2003). While in most traditional retail settings, customers pick up their orders from physical stores at the time of making a purchase, e-tailers are responsible for delivering orders to their customers. This has several implications for designing an effective supply chain network and strategy for an e-tailer. When placing an online order, customers provide the following information:

- Items that are ordered
- Quantity of each item
- Delivery preference
- Shipping address
- Payment method

Although customers indicate their delivery preference and shipping address, they do not control how and when their order is shipped to them. E-tailers typically operate several fulfillment centers (FCs) that are strategically positioned in different geographical locations within their area of operation. Those FCs are responsible to hold inventory and to ship customer orders to their shipping addresses using a courier. When a customer makes an order, the e-tailer assigns

fulfillment responsibility to one FC or a combination of FCs based on their available inventory, customer delivery preference and other criteria. Those FCs are responsible for picking order items from their warehouse, packing them into one or multiple boxes and shipping those boxes to customers within their desired delivery window. This process is referred to as the order fulfillment process. Since e-tailers control the order fulfillment process, they can decide the responsible FCs, number of boxes, shipping time and shipping method to satisfy customer orders. In the e-tail industry, this decision is called the order fulfillment decision. There are several order fulfillment policies that can be adopted by e-tailers for making fulfillment decisions. Those policies govern how fulfillment responsibilities must be delegated to FCs to reduce e-tailers' operating cost. The following sections describe the order fulfillment process, order fulfillment decision and order fulfillment policies in detail.

1.1. Order fulfillment process

When a customer places an online order, e-tailers make order fulfillment decisions to specify which FCs are responsible for fulfilling that order. In this section we examine how FCs fulfill customer orders that are assigned to them.

Figure 1.1 illustrates the order fulfillment process and its timeline using a simple example. As shown in this figure, the order fulfillment process consists of several steps and events that are explained below.

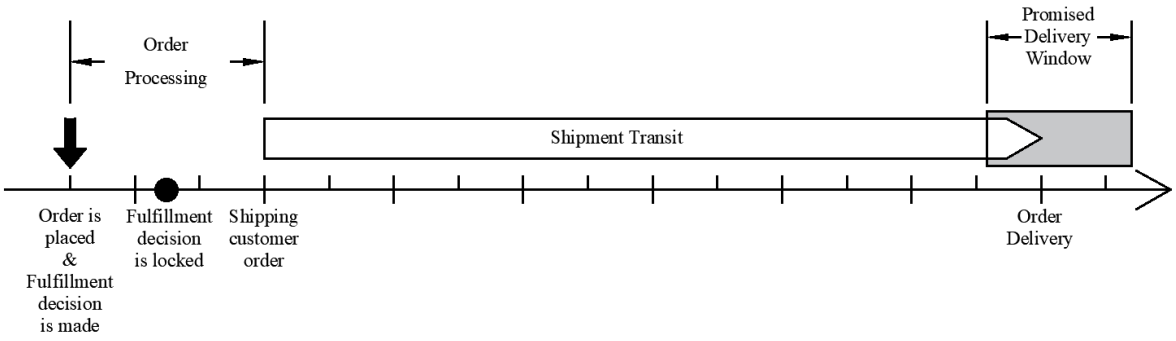

Figure 1.1: Order fulfillment process

- **Order is placed:** This is defined as the time at which a customer places an order. Unlike traditional retailers who normally have specific working hours, e-tailers receive customer orders around the clock.
- **Fulfillment decision is made:** E-tailers make fulfillment decisions on the fly and immediately after a customer order is placed. This decision assigns the customer order to one or multiple FCs which will be responsible for shipping the order items to the customer in one or more boxes.
- **Promised delivery window:** After customer places an order, e-tailer confirms the order and provides an estimated delivery time which is calculated based on order submission time and customer delivery preference. Because of the inherent variability and uncertainties in order fulfillment process, calculating an exact delivery time is not possible and instead, e-tailers provide an estimated time range which we refer to as the promised delivery window.
- **Fulfillment decision is locked:** Although fulfillment decisions are made immediately after customers place online orders, they are not executed in real-time and are added to a queue that contains a list of fulfillment decisions for all customer orders that are waiting in the system for shipment. Fulfillment decisions remain in the queue until few hours before the

customer order is shipped. At that point, the e-tailer locks the decision and begins to prepare the order items for shipment by picking them from the warehouse and placing them into boxes. While fulfillment decisions are waiting in the queue, the e-tailer can review and change them. However, once a fulfillment decision is locked, that decision is finalized, and no changes are allowed.

- **Shipping customer order:** There is a fixed cost associated with shipping customer orders from FCs. In order to break down this cost among multiple customer orders, e-tailers batch several shipments and pick up a group of them together at predetermined times during the day. We refer to this event as shipping a customer order. Time and cadence of this event depends on e-tailer's order volume and other variables.
- **Order processing:** To prepare customer orders that are assigned to them for shipment, FCs need to pick each item from their warehouse and pack them into boxes. This step which occurs between order placement and shipment pick-up is referred to as order processing.
- **Shipment transit time:** Shipment transit time is defined as the elapsed time between shipment pick-up and order delivery. The length of shipment transit time depends on the shipping method that is used at FCs for delivering customer orders.
- **Order delivery:** Order delivery is the actual time at which customers receive their orders. On-time delivery is one of the key performance indicators (KPI) for e-tailers which is measured as the percentage of customer orders that are delivered within their promised delivery windows. Although the e-tailer's goal is to maximize on-time delivery, in some cases due to supply chain related challenges such as inventory shortages and logistical problems, promised delivery window is missed and order delivery happens outside of that.

In addition to the order fulfillment process, it is important to understand the difference between customer delivery options and shipping methods that are used by e-tailers.

- **Customer delivery options:** Customer wait time for receiving their online orders, and the associated delivery cost, are among top e-tail performance measures (Kacen et al., 2013). To improve these metrics, e-tailers offer various delivery options to give customers flexibility in tradeoff between delivery cost and wait time. In this dissertation, we consider four delivery options that are most common in e-tail industry namely: $One\ Day\ Delivery$, $Two\ Day\ Delivery$, $Five\ Day\ Delivery$ and $Seven\ Day\ Delivery$. Intuitively, the faster delivery options are more expensive.
- **Shipping methods:** Most e-tailers outsource their outbound transportation to 3rd party logistic providers (3PLs) such as USPS, UPS and FedEx who are responsible for picking up customer shipments from FCs and delivering them to their shipping addresses. 3PLs offer several shipping methods that vary in shipping cost and transit time. For instance, UPS provides $Next\ Day\ Air$, $Second\ Day\ Air$, $Three\ Day\ Select$ and $UPS\ Ground$ which on average takes five days to deliver a shipment (UPS website, 2019). Like customer delivery options, shipping methods with shorter delivery times are more expensive.

Figure 1.2 depicts the difference between customer delivery options and shipping methods. Consider a scenario where a customer places an order with a $Two\ Day\ Delivery$ preference. The promised delivery window for that order is estimated as a time range between noon and 6 p.m. two days after the order is placed. Assuming customer orders are shipped once every day at noon and all ordered items are available in e-tailer's inventory at the time the order is placed, this order can be fulfilled using one of the following two alternatives. The first alternative is to process the order in the same day and use a $Second\ Day\ Air$ shipping method to send it to the customer. The second

alternative is to wait until the following day and use a $Next\ Day\ Air$ shipping method instead. Although shipping cost for the first alternative is lower, in some cases e-tailer might decide to use the second alternative because of inventory shortage or other constraints.

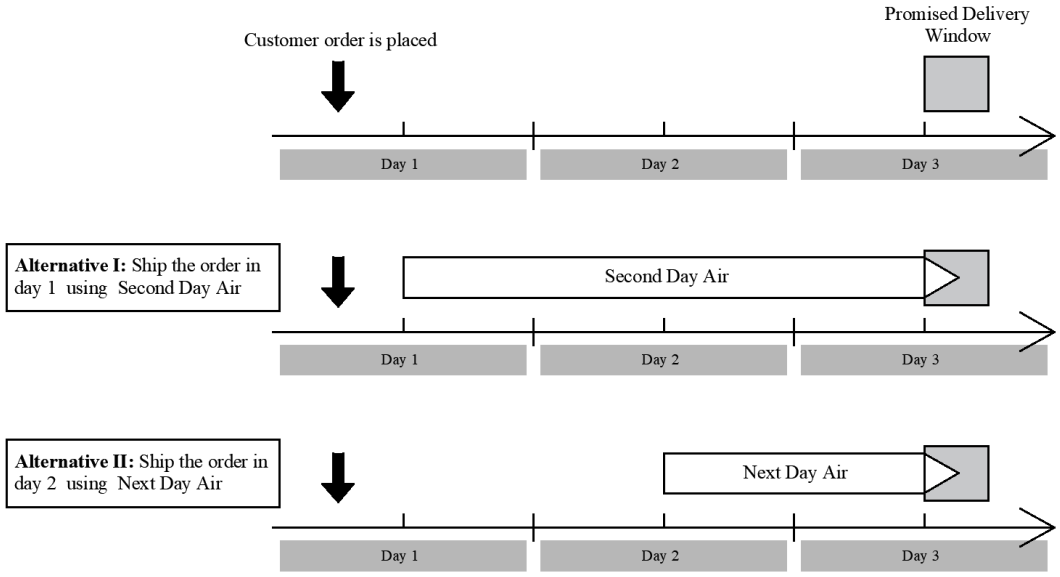

Figure 1.2: Difference between customer delivery options and shipping methods

The order fulfillment process is a critical part of e-tailer operations that not only accounts for a significant portion of overall operating cost, but also has a direct impact on customer service and satisfaction. Since shipping cost is considerably higher than the cost of picking and packing orders, in this dissertation we use it as an estimation of total order fulfillment cost. Acimovic and Graves (2015) report that an e-tailer's shipping cost could amount to 3.2% to 4.6% of its total annual sales. On the other hand, e-tailers often charge a fixed delivery fee for online orders which on some occasions is waived for loyalty program members or large customer orders. Therefore, reducing shipping costs has a direct and major impact on e-tailers' bottom-line.

A detailed analysis of shipping cost is provided in Chapter 3 using a sample dataset from the UPS website. The results of this analysis indicate that the shipping cost for a box is comprised of

two components. The first component is a fixed cost and only depends on the shipping method while the second component also depends on distance traveled and box weight.

1.2. Order fulfillment decision

After a customer places an order and before the order fulfillment process begins, the e-tailer needs to determine how to fulfill the order. We refer to this as the order fulfillment decision in which the following important questions are answered:

- Which FC or FCs are responsible for fulfilling the order?
- Which items and how many of each item are assigned to each FC?
- When should the order be shipped to the customer?
- What shipping method should be used at each FC?

The outcome of the order fulfillment decision is stored in an order fulfillment plan and is sent to the responsible FCs. In order to make this decision, e-tailer needs real-time visibility into available inventory at each FC as well as a list of other customer orders that are assigned to them. Computer information systems such as enterprise resource planning (ERP) and warehouse management systems (WMS) provide this visibility by allowing FCs to share information with each other and enabling the e-tailer to make order fulfillment decisions at a global level.

To understand the order fulfillment decision, consider a simple example in which an e-tailer operates three FCs and has three $SKUs$ in its product catalog (Figure 1.3). Assume a customer places an order on Day_1 before the shipment pick-up time requesting one unit of SKU_1 and SKU_2 with a $Two\ Day\ Delivery$ preference. The available inventory at each FC when the order is placed and their distance to customer shipping address are listed below:

- FC_1 is located 100 miles west of the customer and holds one unit of SKU_1

- FC_2 is located 100 miles north of the customer and holds one unit of SKU_2
- FC_3 is located 200 miles east of the customer and holds one unit of SKU_1, SKU_2 and SKU_3

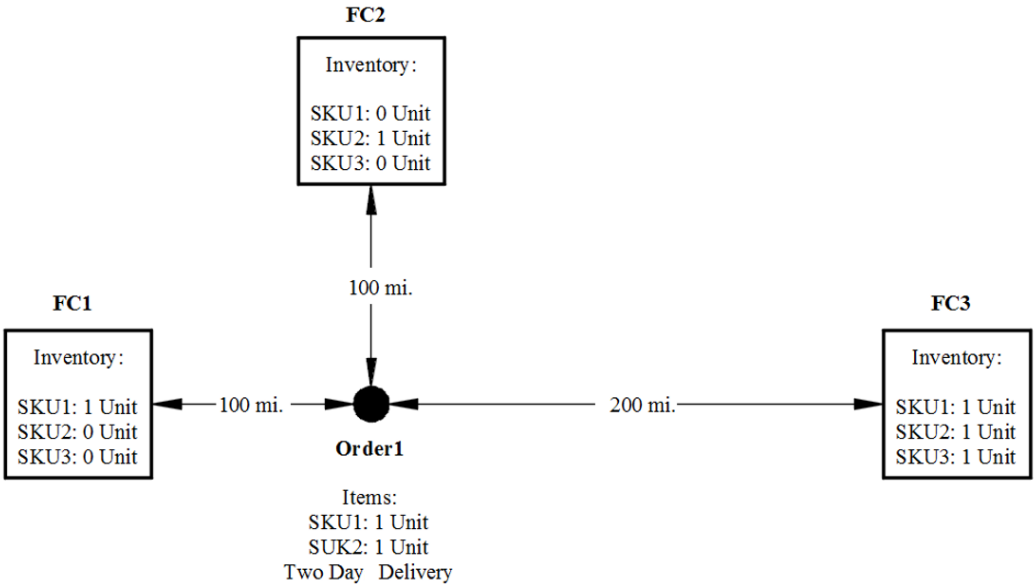

Figure 1.3: Order fulfillment decision for Order 1

In this example, there are several feasible alternatives to fulfill the customer order. Table 1.1 lists all feasible alternatives with their details. Note that since the customer has requested a $Two\ Day\ Delivery$ and the FCs have on-hand inventory for all $SKUs$, the e-tailer can either decide to use a $Second\ Day\ Air$ shipping method on the day the order is placed (Day_1) or wait until following day (Day_2) and use a $Next\ Day\ Air$ shipping method instead.

Table 1.1: Alternatives to fulfill example customer order

	Item assignment			Day of shipment			Shipping method		
Alt.	FC_1	FC_2	FC_3	FC_1	FC_2	FC_3	FC_1	FC_2	FC_3
1	SKU_1	SKU_2	-	Day_1	Day_1	-	*Second Day Air*	*Second Day Air*	-
2	SKU_1	SKU_2	-	Day_2	Day_2	-	*Next Day Air*	*Next Day Air*	-
3	SKU_1	SKU_2	-	Day_1	Day_2	-	*Second Day Air*	*Next Day Air*	-
4	SKU_1	SKU_2	-	Day_2	Day_1		*Next Day Air*	*Second Day Air*	-
5	SKU_1	-	SKU_2	Day_1	-	Day_1	*Second Day Air*	-	*Second Day Air*
6	SKU_1		SKU_2	Day_2	-	Day_2	*Next Day Air*	-	*Next Day Air*
7	SKU_1	-	SKU_2	Day_1	-	Day_2	*Second Day Air*	-	*Next Day Air*
8	SKU_1	-	SKU_2	Day_2	-	Day_1	*Next Day Air*	-	*Second Day Air*
9	-	SKU_2	SKU_1	-	Day_1	Day_1	-	*Second Day Air*	*Second Day Air*
10	-	SKU_2	SKU_1	-	Day_2	Day_2	-	*Next Day Air*	*Next Day Air*
11	-	SKU_2	SKU_1	-	Day_1	Day_2	-	*Second Day Air*	*Next Day Air*
12	-	SKU_2	SKU_1	-	Day_2	Day_1	-	*Next Day Air*	*Second Day Air*
13	-	-	SKU_1, SKU_2	-	-	Day_1	-	-	*Second Day Air*
14	-	-	SKU_1, SKU_2	-	-	Day_2	-	-	*Next Day Air*

The goal of the order fulfillment decision is to find the best alternative based on the e-tailer's order fulfillment policy which is explained in detail in Section 1.3. In this example, we assume the order fulfillment decision aims to find the alternative with the minimum shipping cost. As mentioned earlier, the cost of shipping one box is comprised of two components: a fixed component that depends on the shipping method and a variable component that also depends on box weight and shipping distance. Since all alternatives in this example use either *Next Day Air* or *Second Day Air* shipping methods, assume the shipping cost for these methods are as listed in Table 1.2.

Table 1.2: Cost of shipping one box for different shipping methods

Shipping Method	Fixed shipping cost ($)	Variable shipping cost per pound/mile ($)
Next Day Air	10	0.02
Second Day Air	5	0.01

Assuming each SKU weights exactly one-pound, shipping cost for each alternative is calculated in Table 1.3. Since objective of order fulfillment decision in this example is to minimize the shipping cost for the given order, Alternative 13 will be selected by the e-tailer. The fulfillment plan derived from this decision suggests that both $SKUs$ should be shipped to the customer from FC_3 using a $Second\ Day\ Air$ shipping method on Day_1. This fulfillment plan will be sent to FC_3 to be processed accordingly.

Table 1.3: Shipping cost for all alternatives

Alt.	Number of boxes shipped using *Next Day Air*	Number of boxes shipped using *Second Day Air*	Total pound/mile shipped using *Next Day Air*	Total pound/mile shipped using *Second Day Air*	Fixed shipping cost ($)	Variable shipping cost ($)	Total shipping cost ($)
1	0	2	0	200	10	2	12
2	2	0	200	0	20	4	24
3	1	1	100	100	15	3	18
4	1	1	100	100	15	3	18
5	0	2	0	300	10	3	13
6	2	0	300	0	20	6	26
7	1	1	200	100	15	5	20
8	1	1	100	200	15	4	19
9	0	2	0	300	10	3	13
10	2	0	300	0	20	6	26
11	1	1	200	100	15	5	20
12	1	1	100	200	15	4	19
13	0	1	0	400	5	4	9*
14	1	0	400	0	10	8	18

As e-tailers receive orders around the clock they need to make an order fulfillment decision for each order. The order fulfillment decision described in this example follows a policy that determines the decision should be made in such a way that the shipping cost for the order is minimized. In the next section we describe different order fulfillment policies and how they impact the decision making process.

1.3. Order fulfillment policy

The order fulfillment decision that was described in the previous example evaluates alternatives based on their shipping cost and selects the one with the minimum value to fulfill the customer order. However, this is not a generic approach and e-tailers might have a different strategy for making order fulfillment decisions. Order fulfillment policy determines how the fulfillment decisions should be made and the objective of those decisions.

Some e-tailers fix fulfillment responsibilities ahead of time by assigning all customer orders received from each geographical region to a designated FC. This strategy is called static order fulfillment policy. Although the static policy is relatively simple to implement and maintain, it requires the e-tailer to hold inventory for all $SKUs$ at all FCs. E-tailers typically have a very large product catalog which includes millions of $SKUs$ from various categories. This makes it almost impossible to hold all those $SKUs$ at each location. Instead, they develop an inventory policy that distributes the $SKUs$ among FCs based on their capacity, total customer demand for each SKU, geographical distribution of that demand and other factors. However, a static order fulfillment policy does not work with such inventory system, and e-tailers usually need to take a different approach. A dynamic order fulfillment policy assigns orders as they are placed, to the FC or a combination of FCs that can satisfy them with the minimum shipping cost. Since fulfillment responsibilities in dynamic policy are not decided a priori, and are determined after orders are placed, it can work with a distributed inventory system.

Although dynamic order fulfillment policies work well with a distributed inventory system and minimize shipping cost for individual orders as they are placed, it is possible for them to make a series of myopic optimal decisions which collectively lead to a sub-optimal decision at the system level. This is mainly because dynamic policies make order fulfillment decisions merely based on

the current system state without accounting for future customer orders and inventory replenishments (Xu et al., 2009). For example, a fulfillment decision that optimally assigns an order to FCs based on current information could change the system state in such a way that future orders are fulfilled with sub-optimal assignments due to lack of inventory at certain locations.

To understand the myopic nature of order fulfillment decisions that are made by a dynamic order fulfillment policy, consider the example provided in Section 1.2 and assume after making order fulfillment decision for the first customer order and assigning both SKU_1 and SKU_2 to FC_3 a second customer order ($Order_2$) is placed in the same day requesting one unit of each SKU_1, SKU_2 and SKU_3 with a $One\ Day\ Delivery$ preference (Figure 1.4). If no inventory replenishment happens in Day_1 and since FC_3 has already assigned its inventory of SKU_1 and SKU_2 to fulfill $Order_1$, the only option for fulfilling $Order_2$ is to send a separate box from each FC to the second customer using a $Next\ Day\ Air$ shipping method. The shipping cost for this decision can be calculated using Table 1.2 as $38.80, and the e-tailer's total shipping cost for the two orders is $47.80 collectively.

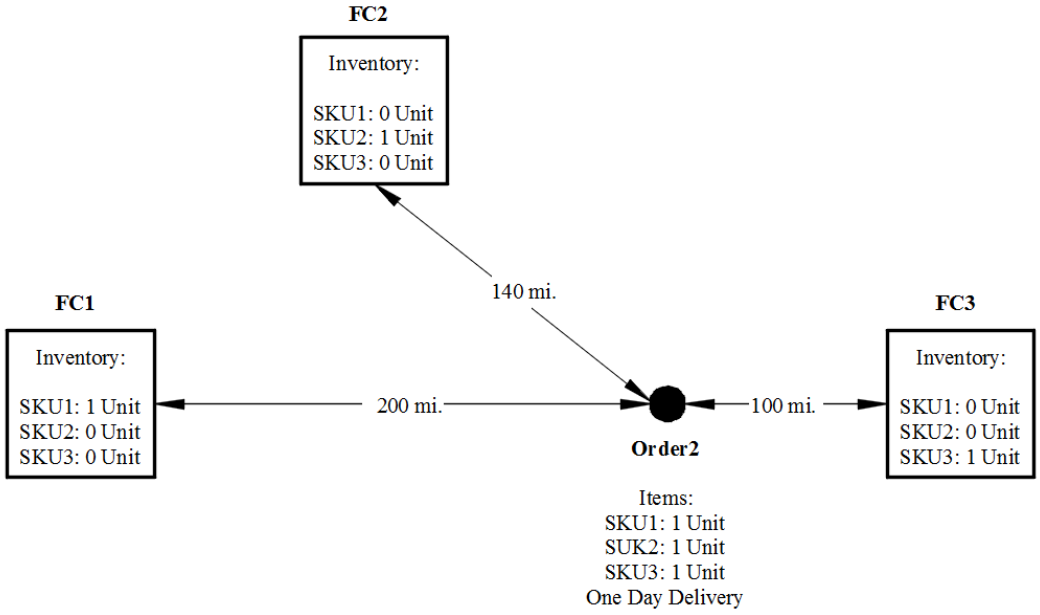

Figure 1.4: Order fulfillment decision for Order 2

Fulfillment decisions for $Order_1$ and $Order_2$ follow a dynamic policy which minimizes the shipping cost for individual orders as they are placed. Now consider a scenario where one fulfillment decision is made for both orders at the same time. Figure 1.5 illustrates the optimal fulfillment decision for this scenario that minimizes the total shipping cost for both orders. In this case, the optimal decision is to assign $Order_1$ to FC_1 and FC_2 and $Order_2$ to FC_3. Total shipping cost for this assignment is \$28 which is \$19.80 less than the previous assignment. This new decision increases the shipping cost for $Order_1$ by \$3 by splitting its shipment into two boxes. However, this adjustment allows all items in $Order_2$ to be shipped in a single box.

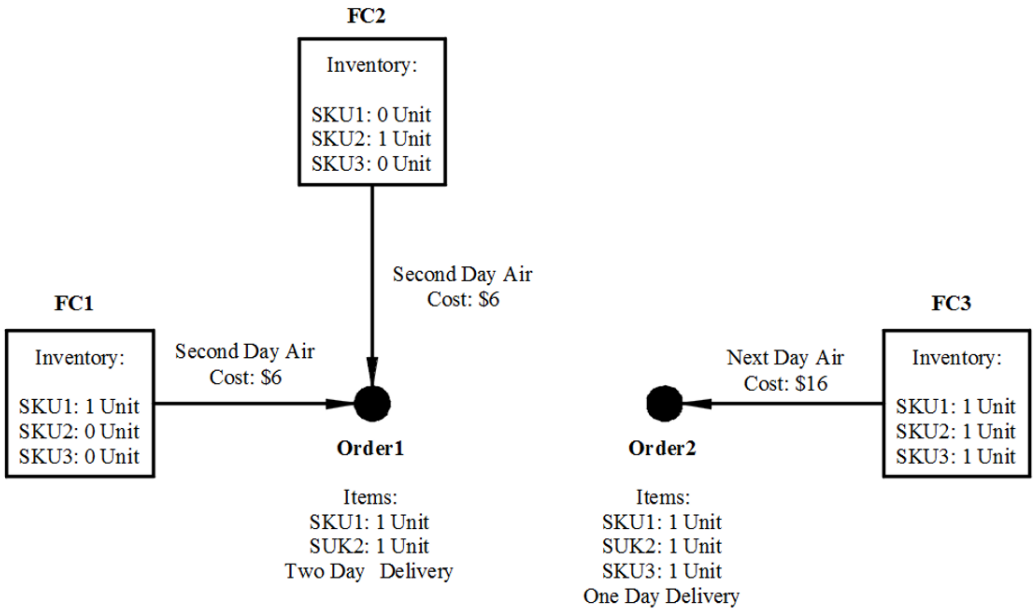

Figure 1.5: Making order fulfillment decisions for both orders simultaneously

As shown in this example, by making fulfillment decisions for multiple orders together, e-tailers can significantly reduce their total shipping costs and avoid myopic decisions that only consider individual orders. However, in practice fulfillment decisions need to be made as orders are placed in order to provide an estimated delivery window to customers and to update on-hand

inventory information at the FCs. In other words, the fulfillment decision for $Order_1$ may not be postponed until $Order_2$ is placed. There are two general techniques to address this problem. The first technique, which is called an *adjusted dynamic order fulfillment policy,* follows the same principles as the dynamic policy except it also accounts for future orders by forecasting them based on historical order information. For instance, Acimovic and Graves (2015) use the dual values of a transportation linear program to estimate future expected shipping cost and apply those estimates in the objective function of a heuristic algorithm that makes fulfillment decisions by minimizing the immediate shipping cost for the current order plus expected shipping cost for future orders.

The second technique is called an *order fulfillment reevaluation policy* which, as suggested by its name, reevaluates fulfillment decisions that have been made using a dynamic policy for a group of orders and optimizes them globally. As mentioned earlier, when a customer places an order a fulfillment decision needs to be made immediately to update inventory status at FCs and to provide an estimated delivery window to the customer. However, there is usually a lag between when a fulfillment decision is made and when the order is shipped to the customer. During this lag orders are processed at designated FCs to get them ready for shipment. While an order is waiting for shipment, more customer orders are placed, and the same process is followed to make a myopic fulfillment decision for them. This lag can be leveraged to reevaluate the initial fulfillment decisions for all the orders that are queued in the system and to make a decision that minimizes their overall shipping cost. For example, Xu et al. (2009) develop a heuristic algorithm that reduces total shipping cost by minimizing total number of customer shipments. Their algorithm reevaluates myopic fulfillment decisions and reduces the number of split shipments by shuffling the assignments. Mahar and Wright (2009) develop a similar approach that assigns accumulated online orders to FCs based on expected inventory, shipping, and customer wait cost.

Figure 1.6 illustrates an order fulfillment reevaluation policy for an e-tailer with three $SKUs$ and two FCs. At time $t = 11{:}35$, FC_1 holds one unit of each SKU while FC_2 only has one unit of SKU_1 and one unit of SKU_2 in its inventory. $Order_1$ is placed at $t = 11{:}40$ requesting one unit of $SKU_1$and one unit of SKU_2 with a $One\ Day\ Delivery$ preference. $Order_1$ distance to FC_1 and FC_2 is 50 miles and 125 miles respectively. Assuming that shipment pick up happens at $t = 12{:}00$, both FCs can fulfill $Order_1$ by sending a single box to the customer using a $One\ Day\ Air$ shipping method. However, since it is cheaper to send the shipment from FC_1, the order is assigned to this FC. Shipping cost for this assignment, calculated based on the shipping rates in Table 1.2, is \$12. After making this fulfillment decision $Order_2$ is placed at $t = 11{:}45$ requesting one unit of each SKU with a $One\ Day\ Delivery$ preference. Since FC_1 has already assigned its inventory of SKU_1 and SKU_2 to $Order_1$, the only alternative to fulfill $Order_2$ is to split it into two shipments and send two separate boxes to the customer from FC_1 and FC_2. The cost of this assignment is \$24.5, increasing the e-tailer's total shipping cost for satisfying both orders to \$36.5. At $t = 11{:}55$ and before shipment pick-up time, the e-tailer can reevaluate the fulfillment decisions for both orders. The decisions generated by the reevaluation suggests that by assigning $Order_1$ to FC_2 and $Order_2$ to FC_1, the e-tailer can satisfy both orders with a total shipping cost of \$32.5 which is \$4 less than the previous assignments.

In this example, we assumed the reevaluation algorithm instantly finds the optimal assignment and did not consider its computation time. However, as we show in Chapter 7 order fulfillment reevaluation is a complex problem and finding an optimal decision for real-world e-tailers with millions of $SKUs$ and tens of FCs in a reasonable amount of time may not be possible. On the other hand, e-tailers operate around the clock 24 hours a day, 7 days a week and since decisions produced by the reevaluation algorithm impact other processes, they need to execute it during their

operations in such a way that it does not halt the system and finds the answer in a timely manner. Therefore, when designing a reevaluation strategy, both algorithm decisions and computation time need to be considered. In the following sections we introduce the concept of an intense unending real-time operational challenge (IURTOC) and explain how combining optimization and simulation techniques allows the e-tailer to objectively compare different reevaluation strategies and find the one that best fits its needs.

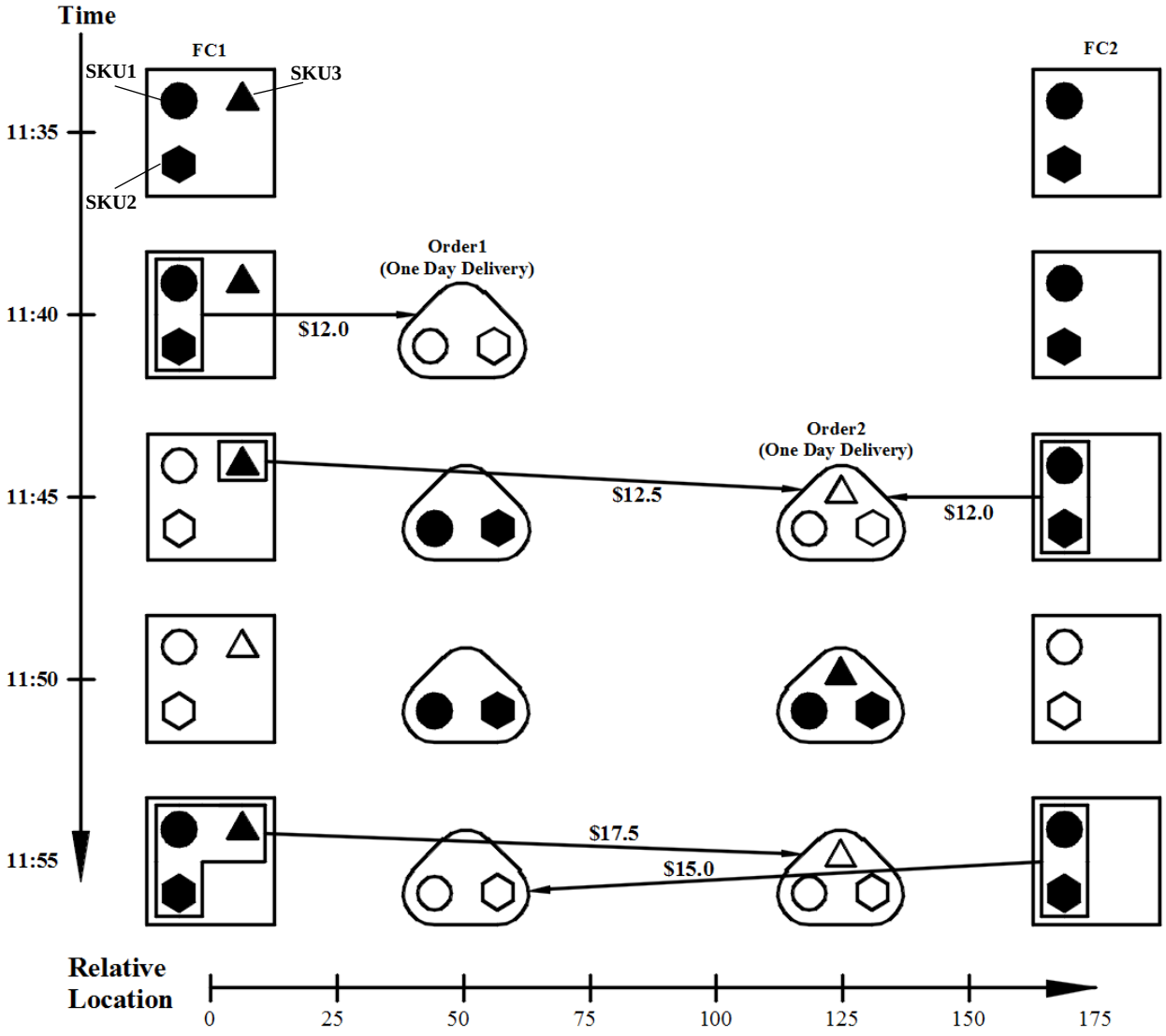

Figure 1.6: Reevaluating order fulfillment decisions

1.4. Intense unending real-time operational challenge (IURTOC)

Petering (2015) defines an intense unending real-time operational challenge (IURTOC) as "a business problem whose goal is to create an algorithm for automatically making operational decisions on a continual basis so as to maximize the productivity of an industrial system whose operations never cease and whose evolution is characterized by incomplete and/or changing second-by-second information regarding process times and new job arrivals from time 0 to time infinity." E-tailer order fulfillment is an IURTOC in which operations never stop as customers place orders around the clock and fulfillment decisions are made one order at a time using an algorithm that automatically assigns them to a set of FCs based on the e-tailer's fulfillment policy. Furthermore, if an e-tailer uses a reevaluation policy, there will be another algorithm to shuffle the assignments for a set of orders to reduce the e-tailer's total shipping cost.

Since in an IURTOC an algorithm is embedded in a system to make operational decisions without human intervention in a continual basis, and since each decision impacts future ones by changing system state, when designing an effective decision-making algorithm, it is important to consider its computation time as well as the quality of the decisions it recommends. A sophisticated mixed integer programming (MIP) algorithm that finds the optimal solution for an IURTOC may not be an ideal option for that system if its computation time takes longer than when the decision is needed. The same rule applies when choosing the right heuristic algorithm. When comparing two heuristic algorithms for an IURTOC, if the first algorithm always finds a solution within 10% of optimal but requires one hour to compute while the second algorithm always finds a solution within 20% of optimal in 10 minutes, and the operational requirements of the system require a solution in less than 15 minutes, the second algorithm should be selected, although it produces inferior decisions.

Consider the example shown in Figure 1.6 where an e-tailer reevaluates fulfillment decisions for $Order_1$ and $Order_2$ at $t = 11{:}55$, five minutes before shipment pick-up time. As it was shown, without considering reevaluation algorithm computation time, the revised assignments become available before shipments are picked up and can be implemented to reduce the total shipping cost. Figure 1.7 depicts the same example but instead of assuming the reevaluation algorithm computation time is negligible, we assume it takes 10 minutes for the algorithm to find an optimal decision. In other words, the new fulfillment decision becomes available at $t = 12{:}05$, five minutes after shipments are sent out. Therefore, the e-tailer needs to keep the original assignments that are made by myopic decisions and although reevaluation finds a better assignment, at the time of execution completion, those decisions are invalid and lead to an infeasible solution.

Most order fulfillment reevaluation studies in the literature focus on designing an optimization algorithm that finds the best decision for customer order fulfillment by minimizing total outbound shipping cost (Mahar and Wright, 2009; Xu et al., 2009; Acimovic and Graves, 2015). To the best of our knowledge, those studies do not address the computation time of those algorithms and its possible impact on the overall system productivity. In some papers in the operations management literature, computer simulation techniques are used for comparing the performance of different algorithms by embedding them in a simulation model. However, in those studies only the decisions produced by the reevaluation algorithm are fed back to the simulation model and their computation time is ignored. In other words, the algorithms are only partially embedded within a simulation model. Petering (2015) proposes an alternative method called a fully embedded decision making algorithm that combines optimization and discrete event simulation (DES) in a novel way to enable a more accurate and objective comparison between different decision-making algorithms for an

IURTOC. This method allows managers and other decision makers to better analyze and evaluate the performance of different algorithms in a test environment before deploying them in the field. This helps to prevent system shutdowns and interruptions which could occur as a result of long running algorithms that in theory produce favorable decisions but in practice halt the system by not providing a decision when it is needed. This technique is explained in the next section.

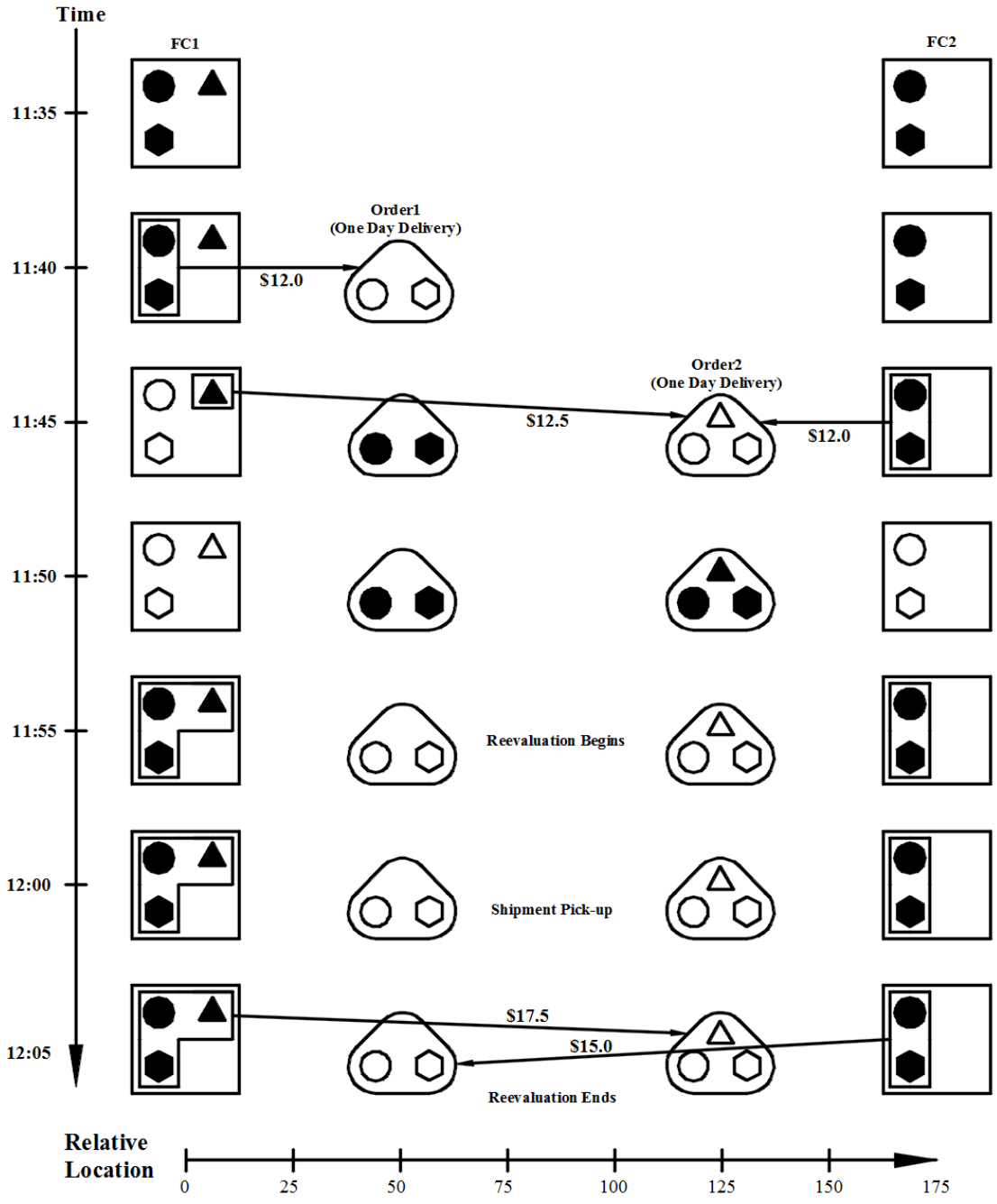

Figure 1.7: Importance of time in reviewing order fulfillment decisions

1.5. Fully embedded decision-making algorithm (FEDMA)

In the previous section, we explained that, in an IURTOC, decisions are made continuously and around the clock in response to dynamic events that could not be predicted ahead of time. When evaluating a decision-making algorithm (DMA) for an IURTOC, it is important to consider the DMA's computation time in addition to the quality of the decisions it recommends. However, most studies in the decision science literature either (i) consider static problems or (ii) design a DMA for an IURTOC without considering how much time is needed for solutions to be found. Regarding case (ii), in some studies in experimental decision science, a discrete event simulation (DES) technique is used to test and compare the performance of different DMAs by embedding them in a simulation model (Petering, 2015). In those studies, only DMA decisions are fed back into a DES model; the computation times are not fed back into the DES model and are assumed to be zero. Petering (2015) calls this technique a *partially embedded decision-making algorithm* (PEDMA) but states that it is preferable to use a *fully embedded decision-making algorithm* (FEDMA) technique that considers both the DMA's decisions and computation time when embedding it into a DES model.

The FEDMA technique allows DMAs to be tested under *true simulated operating conditions* which is more representative of real-world dynamic conditions than the PEDMA technique. Petering (2015) explains that using a FEDMA is particularly important if the average runtime of a DMA is nontrivial compared to the average time that elapses between consecutive calls to the DMA. Additionally, in this dissertation we show that if other types of decisions are made within the same system, using a FEDMA is critical to ensure that the DMA's computation time and decisions do not interfere with these other decisions.

1.6. Contribution and novelty of the research

This research applies the FEDMA technique to the e-tailer order fulfillment reevaluation problem. We develop a DES framework within which different reevaluation algorithms can be fully embedded, and we compare their performance under true simulated operating conditions. This allows managers to find the best reevaluation algorithm for their order fulfillment decisions that not only finds the best order-to-FC assignment but also can be operationalized in real-time without negatively impacting system productivity and performance.

Additionally, we develop an IP and a heuristic algorithm to reevaluate order fulfillment decisions and fully embed them in our DES framework. Using the FEDMA concept we compare the performance of the two algorithms and show that although IP is proven to find an optimal decision, when problem size increases, it is not a viable option because of its computation time. In those cases, the heuristic algorithm should be selected to reevaluate fulfillment decisions.

Beside the reevaluation algorithm itself, execution frequency is an important aspect of the order fulfillment reevaluation policy. For example, using the same algorithm, the e-tailer could reevaluate order fulfillment decisions for 10 orders or 100 orders at a time. We show that although reevaluating more fulfillment decisions together increases the potential savings, it also increases the complexity of the problem instances that are considered, and the computation time needed to identify good decisions. This might have a negative impact on the e-tailer operations. Using the FEDMA technique we can study the tradeoff between savings and computation time and recommend the best reevaluation execution frequency accordingly. We are not aware of any study in the literature that is able to (1) compare different order fulfillment reevaluation algorithms under true simulated operating conditions or (2) recommend the reevaluation algorithm execution frequency that best balances total savings and computation time.

Chapter 2

Review of literature

2.1. FEDMA, PEDMA, and true simulated operating conditions

The concept of combining optimization and simulation has been extensively studied in the literature. Pflug (2012) suggests that combining simulation and optimization is the only practicable way of getting insight into stochastic models and obtaining optimal decisions for them. This is because, while optimization techniques recommend an optimal decision for a system, most of them assume the system under study is static and has deterministic parameters. This is while the vast majority of the real-world systems and phenomena have a stochastic nature. Simulation models, on the other hand, capture stochasticity by using probability distributions that measure the likelihood of various events. Therefore, an effective combination of these two techniques can provide a framework for analyzing many real-world systems.

One of the proposed methods for combining simulation and optimization is called simulation-based optimization or "simulation optimization". In this method, in order to obtain an optimal design for a stochastic system a simulation model is run iteratively, each time with different values for the parameters that define the system. These parameters are the decision variables needing to be optimized, and simulation is used to (i) compute the objective value of and/or (ii) determine if constraints are satisfied by each particular set of parameter values. As the process unfolds, the parameter values gradually move closer to the optimum solution. In this case, simulation is used to set the value of different parameters in each iteration. Gosavi (2015), describes this method in detail and provides a comprehensive overview of different techniques for developing a simulation-based optimization model. Many researchers have utilized simulation-based optimization to solve

problems in a wide range of applications. For instance, Marbach and Tsitsiklis (2001) propose a simulation-based optimization algorithm for optimizing the average reward in a Markov Reward Process where optimization takes place within a parameterized set of policies. Nguyen et al. (2014) provide an overview of simulation-based optimization methods applied to building performance analysis. Becerril-Arreola et al. (2013) apply this method to study an e-tailer's promotional pricing, free-shipping threshold and inventory decisions. Other studies that use simulation-based optimization include Mele et al. (2006), Zeng and Yang (2009), Huang et al. (2010) and Keramydas et al. (2017).

Another method for combining simulation and optimization is embedding an optimization algorithm (i.e. decision-making algorithm) within a simulation model in order to study the impact of the decisions produced by the decision-making algorithm (DMA) on the system's performance. This method has also been extensively leveraged in the field to study a variety of systems. For instance, Sivakumar (1999) develops a discrete event simulation (DES) model of the complex manufacturing environment of a semiconductor test facility. He then embeds an optimization algorithm within this DES model for online and near real-time scheduling. Using this method, he achieves a world-class cycle time, improved machine utilization and more predictable and highly repeatable manufacturing performance. Hillstrom (1977) utilizes this technique to develop a methodology to evaluate the performance of unconstrained nonlinear optimization algorithms. This methodology enables decision makers to compare the performance of multiple optimization algorithms in a simulated environment. In a similar study, Beiranvand et al. (2017) describe this method as one of the best practices for benchmarking the performance of different optimization algorithms. Other studies that use this method include Azadivar and Wang (2000), Marques et al. (2014) and Hare et al. (2018).

When embedding a DMA within a DES model, most studies in the literature only feed the decisions produced by the DMA to the DES; and they do not account for its computation time. However, the computation time of a DMA can have a significant impact on a system's operation. For instance, in designing a DMA to assign drivers to passengers in a ride-sharing app, if a DMA produces high quality decisions but takes one-hour of computation time, it may not fit the operational requirements of the real-world system. Therefore, in order to effectively evaluate a DMA, both the decisions produced and computation time must be taken into consideration.

Petering (2015) refers to the technique in which only the decisions produced by the DMA are fed back to a DES model as *partially embedding the DMA* and the algorithm itself as a *partially embedded DMA* (PEDMA). He proposes a novel technique to embed a DMA in a discrete event simulation (DES) model so that both the decisions produced, and the computation time used, by the DMA are fed back to the DES model. In this case the algorithm is called *a fully embedded decision-making algorithm* (FEDMA). The FEDMA technique allows decision makers to study a system under true simulated operating conditions.

Using a FEDMA is particularly important when the system under study operates at a fast pace and its operations never halt. In these systems, the DMA must be executed during normal operations without negatively impacting the system's performance, so it is critical to consider the DMA's computation time. Petering (2015) defines an intense unending real-time operational challenge (IURTOC) as "a business problem whose goal is to create an algorithm for automatically making operational decisions on a continual basis so as to maximize the productivity of an industrial system whose operations never cease and whose evolution is characterized by incomplete and/or changing second-by-second information regarding process times and new job arrivals from time 0 to time infinity."

In this dissertation we show that an e-tailer's order fulfillment process is an IURTOC and develop a FEDMA to reevaluate an e-tailer's order fulfillment plans under true simulated operating conditions.

2.2. Online retailing

In this section we review the literature on online retail (e-tail) order fulfillment processes and related topics. The tremendous growth of the e-tail industry over the past two decades has attracted researchers and practitioners from various disciplines which has resulted in numerous research articles. Within this rich literature, we focus on e-tail success factors, supply chain management, transportation planning, and order fulfillment and delivery.

2.2.1. Enablers and success factors

The explosion of the e-tail sector has drastically transformed customer behavior and shopping habits in the last few decades. While shopping in a physical store was once the primary way to shop, e-tail is quickly becoming a preferred way to shop for customers around the world. Many researchers have developed quantitative and qualitative methods to identify enablers and facilitators for this rapid growth. Sahney (2008) follows an empirical study to conceptualize key e-tail enablers and uses a quality function development technique to identify performance indicators that are critical to the success of an e-tailer. The model determines clear transaction policies, online interactivity between buyer and seller, transaction safety and transaction privacy as main facilitators/enablers of e-tail systems. An effective website design which includes functionality, usability, ease-of-navigation and interface is another critical success factor that has been extensively studied. Constantinides (2004) studies the web experience components and their role as inputs in the online customer decision making process. This study shows that e-tail firms delivering superior web experience influence their client's perceptions and attitudes and drive

additional traffic to their online store. Yen et al. (2007) develop an analytical model for effective web store design that can measure website accessibility in a systematic and quantitative manner. Other success factors that have been proposed include consumer traits, sense of freedom and control, convenience, customized service, access to wider variety of products, trust and shopping experience (Grewal et al., 2004; Dabholkar and Bagozzi, 2002; Wolfinbarger and Gilly, 2001; Elliot and Fowell, 2000; Shim et al., 2001; Eastlick and Lotz, 1999; Yoon, 2002; Lee and Turban, 2001).

2.2.2. Supply chain network

There are fundamental differences in the supply chain of traditional retailers and e-tailers. According to Xu (2009), the common characteristics of the e-tail supply chain that distinguishes it from traditional retail include:

(i) Large scale: Since e-tailers are not limited by the size of their stores, they can operate multiple fulfillment centers and use large physical spaces to store their products. This enables them to offer a more diverse product catalog compared to traditional retailers.

(ii) Logistics as a matter of trust: trust and timely delivery are two of the most critical success factors for an e-tailer. Brynjolfsson and Smith (2000) compare e-tailers and traditional retailers and conclude that branding, awareness and trust are determining success factors for online stores. Keeney (1999) conducts a survey to analyze the advantages and disadvantages of e-tail from the customer standpoint. This study finds that timely delivery of products is a major factor contributing to the success or failure of an e-tailer. The results of a survey conducted by Torkzadeh and Dhillion (2002) confirms Keeney's analysis.

(iii) High visibility: e-tailers collect a large amount of data about customer orders and buying behavior. They also share a lot of information with customers in their website. This not

only allows e-tailers to improve their online store but also helps customers throughout the purchasing process.

(iv) Assemble to order system: when customers place an online order, the e-tailer can decide to send that order to the customer in one or multiple shipments. Given the large number of SKUs in e-tailers website, the number of possible combinations is enormous. Therefore, it is critical for the e-tailer to make good decisions about how to assemble an order and ship it to the customer. There is a significant opportunity to reduce shipping cost by making the right decision.

(v) Delay in demand fulfillment: there is a delay between the time when a customer places an online order and when it is delivered to their address. The length of this delay differs based on customers' delivery preferences. E-tailers can leverage this delay to improve their fulfillment decisions and minimize their overall shipping cost.

(vi) Retailer directed demand allocation: e-tailers control how customer orders should be assigned to fulfillment centers or drop-shippers.

Because of these differences, retailers who add an online channel to their existing physical channel need to re-design their supply chain. Additionally, e-tailers who operate completely online without a physical store, need to take these differences into account for designing an effective supply chain. Retailers can treat their online channel as a separate business unit and designate a dedicated supply chain to fulfill online customer orders. Hovelaque et al. (2007) study different organizational models for traditional retailers who decide to add an online sales channel. They use a newsboy order policy model to compare the performance of three different organizational models: "store-picking", "dedicated warehouse-picking" and "drop-shipping". Their analysis indicates that, retailers can increase their profit by using a "store-picking" or "drop-shipping"

models when compared to “warehouse-picking”. Ma et al (2017) develop a news vendor model to analyze the value of drop-shipping for retailers with online and physical channels. Their results show that drop-shipping can significantly reduce store inventory, streamline returns and increase overall profit.

In addition to drop-shipping, retailers can designate distribution centers for their online channel. De Koster (2003) proposes a model that establishes a positive association between operational complexity and the distribution structure of food e-tailers. Based on this analysis, complex operations with large product assortment tend to have special distribution centers for online orders and new internet-only companies tend to use special internet-orders only warehouses. In a similar article, Bendoly et al. (2007) propose that if percentage of total demand that is online exceeds a threshold, it is best to assign a dedicated warehouse for online channel. Maher et al. (2015) propose that retailers can save up to 18% in total cost by presenting only a subset of their stores to online customers as potential pick-up locations. Xiao et al. (2009) use a discrete-time dynamic programming model to analyze the impact of demand seasonality on an e-tailer’s inventory management policy.

The primary reason for separating the supply chain network of online and physical channels is lack of preconditions for integration which includes know-how, resources, infrastructure and requirements for picking (Hübner, 2015). On the other hand, there is a significant value in integration between different channels. Integration allows retailers to use existing infrastructure, increase synergy and leverage inventory pooling and transshipment. Therefore, although combining the online and physical channel into one compelling seamless customer experience is one of the biggest challenges for retailers and manufacturers (Tetteh and Xu, 2014) we are now observing a move from multichannel to omnichannel which is an emerging channel integration

strategy aiming to address this challenge (Ansaripour and Trafalis, 2013; Piotrowicz, 2014; Verhoef et al., 2015; Mena et al. 2016). In an omnichannel model, customers can shop online and offline at the same retailer (Bett et al., 2013). Some retailers also allow customers to buy from the online channel and pick up their products from a physical store (Gao and Su, 2017). In addition to providing value for retailers, an omnichannel model also allows customers to use channels in parallel and simultaneously (Parker and Hand, 2009; Ortis and Casoli 2009; Rajendran et al., 2019) and enables a better and more streamlined return process (Akturk et al., 2018).

2.2.3. Order delivery

Order delivery is a key service element for an e-tailer (Boyer and Hult, 2005; Agatz et al., 2008). Delivery encompasses any activities that physically move the product from the e-tailer to the customer. In the case of home delivery, this is known as the last mile. The last mile can be divided into customer pick-up versus home delivery (Daduna and Lenz, 2005) which can be further subdivided into attended and unattended delivery (Kamarainen and Punakivi, 2002).

Most e-tail customers request home delivery (Devari et al., 2017). In an attended home delivery, the customer and e-tailer need to agree on a delivery time window. The length and timing of this window as well as delivery lead time are among the key customer service indicators (Agatz et al., 2008). On the other hand, they have a direct impact on the e-tailer's delivery costs. The last mile delivery cost can account for 13% to 75% of total supply chain costs (Gevaers et al., 2009). Finding the right balance between cost and service is a challenging problem (Boyer et al., 2003; Esper et al., 2003) that needs to be carefully examined for an e-tailer based on its customer expectations, competitors and other determining factors.

Although last mile delivery has traditionally been handled by commercial carriers such as UPS, e-tailers are looking for options that can make their order delivery process more efficient. The highly advertised Amazon Prime Air delivery service using UAVs which can reduce delivery lead time from multiple days to a few hours is an example of such an attempt (Jung and Kim 2017). In the same vein, a stream of research has explored the idea of crowd logistics for resolving the last mile delivery issues in urban areas (Devari et al., 2017). Crowd logistics provides economic benefit for all parties involved by designating the outsourcing of logistic services to a crowd (Mehmann et al., 2015). Crowd logistics is massively supported by the increasing digitization of the society (Unterberg, 2010) and end-to-end information sharing enabled by customers' smartphones throughout the process.

2.2.4. Order fulfillment decision

When a customer places an online order, the e-tailer makes an order fulfillment decision that assigns fulfillment responsibility to one or more FCs with available inventory and determines an estimated delivery date. In practice, most e-tailers optimize the order fulfillment decision for each customer order based on outbound transportation cost (Malykhina, 2005). Additionally, many e-tailers make order fulfillment decisions immediately after a customer places an order (Soars, 2003). Since order fulfillment decisions have a significant impact on an e-tailer's bottom-line, many researchers have proposed different models and techniques to improve the decision-making process.

One stream of research has explored improving the fulfillment decision for individual customer orders by improving the decision-making algorithm or combining that decision with other operational decisions made by the e-tailer. Jasin and Sinha (2015) formulate the online order fulfillment decision problem as a stochastic model and derive an approximation of that in form of

a deterministic linear program (DLP). They use two heuristic algorithms to solve this DLP and through numerical experiments illustrate that consolidating shipments for a customer order increases transportation cost savings. Bhargava et al. (2016) develop a Best Matching Protocol (BMP) for order fulfillment decisions in a collaborative and geographically distributed network. This protocol enables collaboration between multiple order fulfilling agents and provides a scalable solution for the increasing size of a supply network. Ardjmand et al. (2018) propose a genetic algorithm that integrates order cartonization into order fulfillment decisions to improve the overall shipping cost and fulfillment time. In a similar article, Govindarajan et al. (2018) propose a heuristic algorithm that combines the inventory policy with the order fulfillment decision. This combined approach outperforms a decentralized planning strategy that treats the inventory policy and order fulfillment as separate decisions. Other articles that consider improving fulfillment decisions for individual customer orders include Rambaran (2016), Acimovic and Graves (2017), Lei et al. (2018), Chen et al. (2019) and Li and Jia (2019).

Another group of researchers have developed models that consider making fulfillment decisions for a group of customer orders as opposed to making myopic decisions for individual orders. Mahar and Wright (2009) develop a quasi-dynamic allocation policy that postpones fulfillment decisions for individual orders and instead assigns accumulated orders to fulfillment centers. This model also considers expected inventory and customer wait costs in addition to outbound transportation cost and reduces overall operating cost by as much as 23%. Acimovic and Graves (2015) use the dual values of a transportation linear program to estimate the future expected shipping costs and apply those estimates in the objective function of a heuristic algorithm that makes fulfillment decisions by minimizing the immediate shipping cost for the current order plus the expected shipping cost for future orders. Xu et al. (2009) develop a heuristic algorithm that

reduces the total shipping cost by minimizing the total number of boxes that are shipped. Their algorithm reevaluates myopic fulfillment decisions and reduces the number of split (i.e. multi-box) shipments by shuffling the assignments.

Chapter 3

Discrete-event simulation model for an e-tailer order fulfillment process

Discrete-event simulation (DES) is one of the most popular modeling techniques in experimental decision science that is used to study the behavior and performance of a discrete system over a finite time horizon. In this chapter a DES model of an e-tailer order fulfillment process is presented. This model is utilized in the following chapters to compare the performance of different reevaluation algorithms that are fully embedded in the simulation model.

There are several out-of-the-box DES software packages on the market with relatively simple and intuitive paradigms for developing simulation models. Although those packages accelerate the model building process by simplifying and automating most of the tasks, they do not provide the required flexibility for fully embedding complicated reevaluation algorithms. Therefore, in this dissertation we build the DES model from the ground up in the C++ programming language which allows that flexibility. In Chapter 5 we leverage direct integration between C++ and the CPLEX optimization package to fully embed an integer programming-based reevaluation algorithm in this simulation model.

3.1. Generic DES model architecture

DES models a system as a series of events that occur over time and assumes no change in the system's state between those events. This is in contrast with continuous simulation in which the system state evolves at regular intervals of time. The choice between DES and continuous simulation depends on the characteristics of the system under study. For example, DES can be used to study average customer wait time in a bank by modeling the system using discrete events such as customer arrivals and departures. However, to study an electric circuit, since system

evolution cannot be modeled using discrete events, a continuous simulation model should be used instead (Alimeling et al., 1999).

Figure 3.1 illustrates the generic DES model architecture and its components. This section provides a brief description of each component and explores how they work together within the context of a DES model. There are many introductory books to DES that provide a comprehensive overview of this field including Banks et at. (1996). The specific DES model for an e-tailer order fulfillment process is described in Section 3.2.

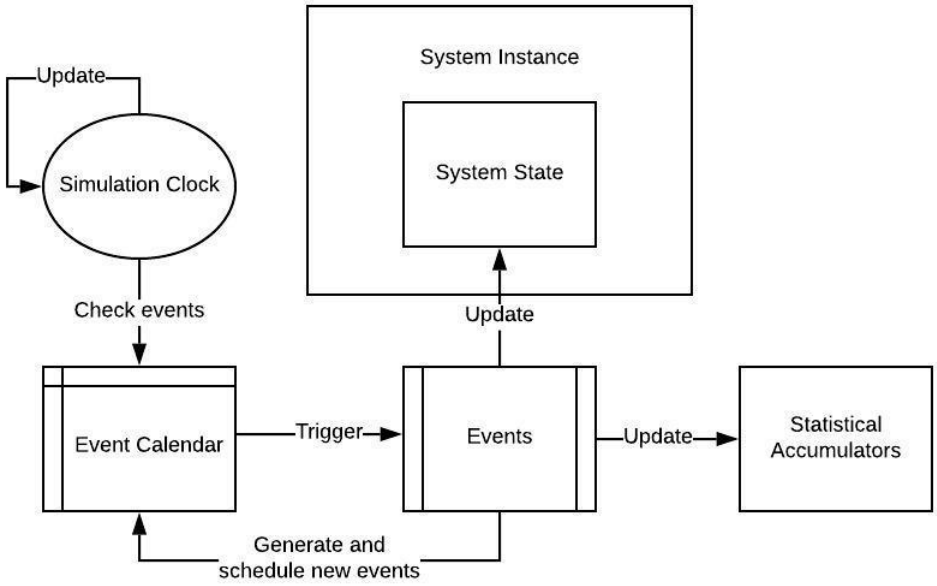

Figure 3.1: Conceptual diagram of a DES model architecture

3.1.1. System instance

The system instance is a mathematical representation of the real-world system under study. Building a DES model starts with defining a system instance that accurately captures the relevant operational behavior and characteristics of the real-world system. For example, a system instance for a DES model of a bank includes the customer interarrival time, service time, number of bank

tellers and other information that needs to be specified in order to fully define the system under consideration.

3.1.2. System state

The system state is a set of variables whose values capture the system status at a given moment in time. It is used to monitor the system's evolution over the simulation time horizon by capturing the impact of each event. In the bank example, the number of bank tellers is not considered part of the system state since it is static and does not change as events occur. The number of customers in the queue, on the other hand, is dynamic and changes during simulation and hence is included in the system state.

3.1.3. Events

The events are the key building blocks of a DES model that represent any activity that makes a change in the system state. As mentioned earlier, DES assumes that events occur at a point in time and that the system state does not change between two consecutive events. In the bank DES model, customer arrival is an example of an event which changes system state by adding a new customer to the queue. As illustrated in Figure 3.1, in some cases, execution of an event results in addition of a new event to the event calendar. This is a very important property of DES models that will be explained further in the following sections.

3.1.4. Simulation clock

The simulation clock is a virtual timer inside a DES model that keeps track of the current simulation time. DES models are developed to analyze a system within a finite time horizon. The length of this time horizon and its measurement units are determined by the analysis objectives. The simulation clock starts at time 0 (at the beginning of this time horizon), moves forward

incrementally and checks the event calendar for the next event that needs to be triggered at each point in time. In this dissertation we consider a next-event time progression which means that the simulation clock moves directly from the starting time of one event to the next one. Alternatively, a fixed-increment time progression can be used that moves the simulation clock forward in fixed increments.

3.1.5. Event calendar

The event calendar is a list of events (and their related information) that are scheduled to take place at known future times (after the current clock time). As the simulation clock progresses, if the starting time of an event in the event calendar matches the current simulation time, it is triggered by simulation. The event calendar is constantly being updated during a simulation as past events are deleted and future events are added.

3.1.6. Statistical accumulators

The statistical accumulators keep track of various system performance metrics during a simulation. The type and number of statistical accumulators are determined by the objectives of the analysis. The value of a statistical accumulator is updated by different events and are reported at the end to help analyze the simulation output. In the bank DES model, average customer wait time is a statistical accumulator that is updated as each customer enters and subsequently exits the bank.

3.2. DES model of an e-tailer order fulfillment process

In this section a DES model of an e-tailer order fulfillment process is presented. As mentioned earlier in this chapter, this DES model is developed in the C++ programming language following an object-oriented approach. Table 3.1 displays a list of indices and parameters used to define the

system instance for this model. This section provides an overview of this DES model and describes the modeling approach and assumptions.

Table 3.1: Indices and parameters for DES model

Indices	
s	SKU
f	FC
r	Customer order
i	Item: a particular SKU that is requested in a particular order
m	Shipping method
p	Delivery preference
a	Assignment
z	Shipping zone
g	Region
q	Order quantity
Parameters	
$Length$	Length of rectangular space that represents the e-tailer's area of operation, (real, > 0)
$Width$	Width of rectangular space that represents the e-tailer's area of operation, (real, > 0)
$NumSKUs$	Number of SKUs in e-tailer product catalog, (integer, > 0)
$NumFCs$	Number of FCs in e-tailer supply chain network, (integer, > 0)
$NumRegions$	Number of regions in e-tailer area of operation, (integer, > 0)
SKU_s	An object that stores a representation of SKU s, (instance of SKU class)
FC_f	An object that stores a representation of FC f, (instance of FC class)
$DeliveryDays_p$	Number of days a customer should wait to receive their order if delivery preference p is selected, (integer, > 0)
$DelProb_p$	Probability that delivery preference p is selected when an order is placed, (real, ≥ 0)
$AvgIntPlcTime$	Average inter-order-placement time, (real, > 0)
$MaxOrdLines$	Maximum number of order lines allowed in a customer order, (integer, > 0)
$OrdLinesProb_i$	Probability that i items are requested in a customer order, (real, ≥ 0)
$ShippingDays_m$	Number of days it takes for a box to reach its destination when it is shipped by shipping method m, (integer, > 0)
$SkuProb_s$	Probability that SKU s is selected in a customer order, (real, ≥ 0)
$MaxQuantity$	Maximum quantity of any SKU that can be requested in any order, (integer, > 0)
$QtyProb_q$	Probability that the order quantity is q in an order item, (real, ≥ 0)
$region_g$	An object that stores representation of region g, (instance of $Region$ class)
$RgnProb_g$	Probability that an order originates from region g, (real, ≥ 0)
$MaxBoxWeight$	Maximum weight capacity (in pounds) of a box, (real, > 0)

3.2.1. System instance for e-tailer DES model

The e-tailer order fulfillment process is a complex system that involves many entities and relationships. There are many ways to model this system using DES. Since the primary objective of this dissertation is studying different strategies and algorithms for reevaluating order fulfillment decisions, we model those system characteristics that are relevant to this analysis. In order to develop a generic simulation model that can be used by decision makers across a wide variety of e-tailers, we use several parameters in this model. The value of these parameters should be set

based on the characteristics of a real-world e-tailer. Figure 3.2 illustrates the main components of the system instance for this DES model.

Note that in this simulation, we use an object-oriented modeling approach in which each entity is modeled as a member of a class. The characteristics of that entity are modeled as attributes of that class. For example, each FC in this simulation is modeled as a member of the *FC* class. This class has several attributes including $invInfo_s$ that specifies the inventory policy for SKU_s at an FC. In this dissertation, to reference an attribute of a class we use the following terminology: ($member\_name.attribute$). For example, $FC_f.invInfo_s$ describes the inventory policy for SKU_s at FC_f. Note that an attribute of a class can be another class.

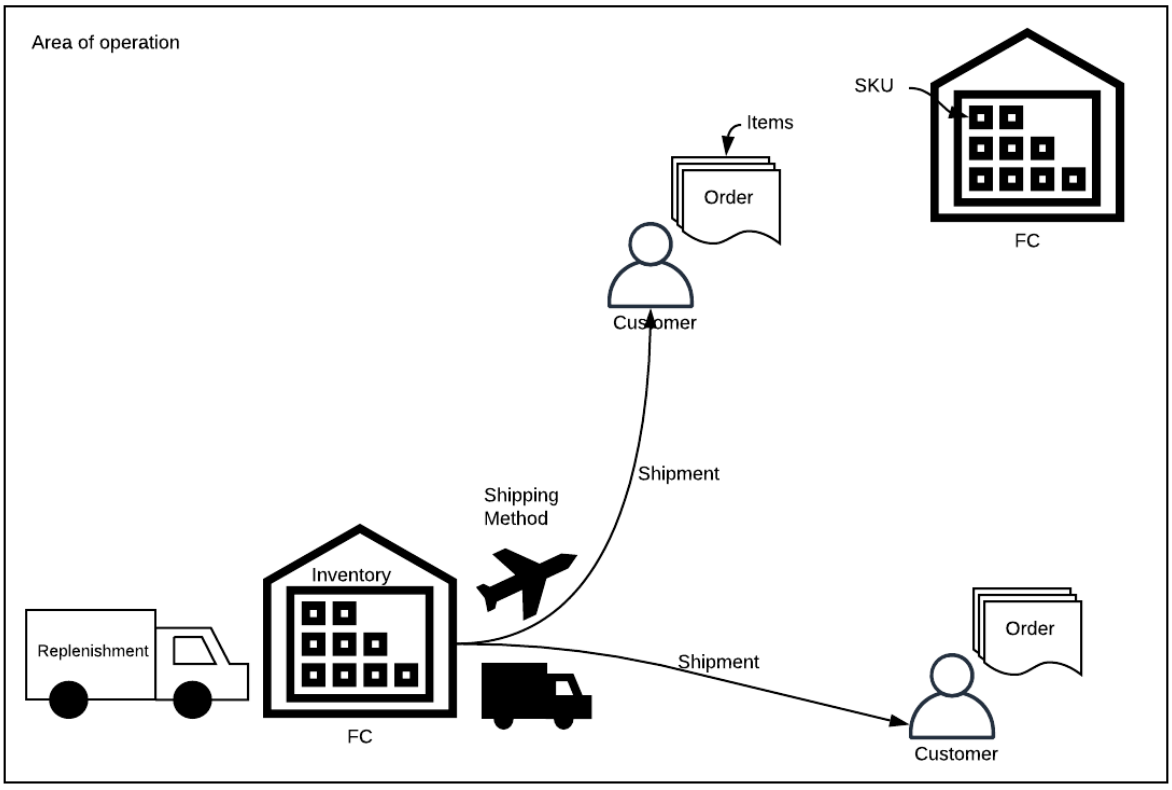

Figure 3.2: Visual representation of system instance for the DES model

Time measurement. For this DES model, time is measured in minutes. The timing of any event including customer order placement and order shipment is captured by measuring the number of minutes that elapsed between the simulation starting time and when that event occurs. Additionally, we assume that the e-tailer's operations never stop and customer orders are placed and processed by the e-tailer around the clock. The simulation time horizon is a model parameter that can be adjusted. For example, for simulating one month of the e-tailer's operations, the time horizon should be set to 43,200 minutes.

Area of operation. E-tailers typically serve customers within a certain geographical region which is referred to as their area of operation. Some e-tailers ship products to their customers in a single country while others operate internationally. In this DES model, the area of operation is assumed to be a rectangular area. The size of this area is modeled using the $Length$ and $Width$ parameters.

The location of each point within the e-tailer's area of operation is measured based on its relative position to the bottom-left corner which is assumed to be the origin of a two-dimensional coordinate plane. This coordinate plane is used to define the location of each FC and customer which allows the simulation model to measure the Euclidian distance between each FC and customer order. Table 3.2 provides the definition of the $Location$ class which is used in this simulation to store the x and y coordinates of a customer and FC location.

Table 3.2: Location class definition

$\boldsymbol{Class}$: $Location$	
$xCoord$	X coordinate of a location, $(real, 0 \leq xCoord \leq Length)$
$yCoord$	Y coordinate of a location, $(real, 0 \leq yCoord \leq Width)$

Stock keeping units (SKUs). In this simulation model, a SKU is defined as a distinct type of item for sale on the e-tailer's website. The $NumSKUs$ parameter represents total number of SKUs

in the e-tailer's product catalog. Characteristics of SKU s are modeled using the SKU_s object which is a member of the SKU class that is described in Table 3.3. As shown in this table, the SKU class includes two parameters that model the SKU ID and weight.

Table 3.3: SKU class definition

Class: SKU	
SKU_ID	SKU ID, (integer, $1 \leq SKU\_ID \leq NumSKUs$)
$weight$	SKU weight in pounds, (real, > 0)

Fulfillment centers (FCs). FCs are located inside the e-tailer's area of operation and are responsible for managing its inventory and fulfilling customer orders. The $NumFCs$ parameter represents total number of FCs in the e-tailer's supply chain network. The characteristics of FC f, are modeled using the FC_f object which is a member of the FC class that is described in Table 3.4.

Table 3.4: FC class definition

Class: FC	
FC_ID	FC ID, (integer, $1 \leq FC\_ID \leq NumFCs$)
$FCLoc$	FC location, (instance of $Location$ class)
$invInfo_s$	Inventory management information for SKU s, (instance of $InvInfo$ class)

The first parameter in this class is FC_ID which assigns a unique identifier to each FC object. In this model, an auto-increment number is used for generating values of FC_ID. The $FCLoc$ parameter captures location of FC_f based on the coordinate system that was described previously.

The inventory management policy for SKU_s at FC_f is modeled using the $invInfo_s$ object which is a member of the $InvInfo$ class. The definition of this class is provided in Table 3.5.

Table 3.5: InvInfo class definition

Class: $InvInfo$	
SKU_ID	SKU ID, (integer, $1 \leq SKU\_ID \leq NumSKUs$)
$maxLevel$	Maximum inventory level, (integer, ≥ 0)
$reviewCycle$	Inventory review cycle in minutes, (real, > 0)
$leadTime$	Lead time for receiving a replenishment order in minutes, (real, ≥ 0)

Two main aspects of the e-tailer inventory management policy are considered in this simulation model:

- Inventory placement: e-tailers typically offer a large variety of SKUs in their website. However, to reduce operational costs, they do not store all SKUs at all FCs and instead strategically position their inventory by spreading it among their locations. This is modeled in this simulation using the $maxLevel$ parameter in $InvInfo$ class. If in the real-world system, the e-tailer does not stock SKU_s in FC_f, the value of $FC_f.invInfo_s.maxLevel$ can be set to 0.
- Inventory replenishment policy: this refers to the frequency and size of replenishment orders for each SKU. In this simulation model, we assume that all FCs follow an order-up-to-level (OUTL) policy with periodic review for inventory replenishment which is illustrated in Figure 3.3. There are three parameters in the $InvInfo$ class that are used to model this policy. The $maxLevel$ parameter refers to the maximum inventory level for a SKU. The $reviewCycle$ specifies the frequency at which inventory level is reviewed at the FC. Since there usually is a lag between the time at which a replenishment order is placed and when it reaches the FC, the $leadTime$ parameter is used to capture that. The order quantity for each replenishment is calculated based on on-hand inventory and its difference with $maxLevel$. Note that the value of all these parameters could be different for each FC-SKU combination. For example, if $maxLevel$ for SKU_s at FC_f is 200 with a $reviewCycle$ of 2880 minutes (2 days) and $leadTime$ of 1440 minutes (1 day), when the simulation begins, the inventory level for this SKU and FC combination is set to 200. After that, the inventory level decreases as customers place orders and they are assigned to FC_f. At $t = 2880$, the inventory level is reviewed for the first time. Assuming that on-hand inventory for SKU_s is 50, a replenishment order is placed

with a quantity of 150 plus the expected demand for SKU_s at FC_f during the $leadTime$. For instance, if on average 75 units of SKU_s are shipped from FC_f in a day, since $leadTime$ is 1 day, 75 units are added to the replenishment order quantity and the total quantity becomes 175. Note that, this number does not immediately get added to the on-hand inventory level. Instead this is added to on-order inventory which will be turned into on-hand inventory after 1440 minutes.

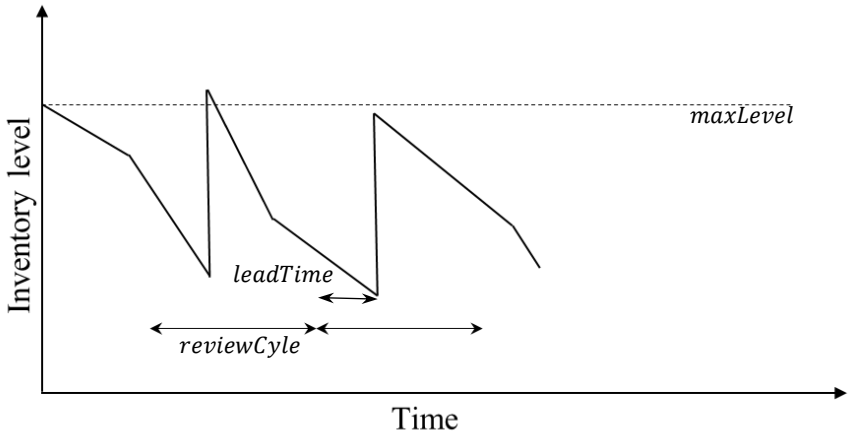

Figure 3.3: Order-up-to-level inventory policy with periodic review

Delivery options. Several studies indicate that the variety, cost and lead time of e-tailer delivery options is a highly influential part of the customer buying process (Esper et al., 2003; Ma, 2017) Most e-tailers offer several delivery options to their customers. In this DES model we consider four of the most common delivery options that are listed below:

- Delivery Option 1: *Seven Day Delivery*
- Delivery Option 2: *Five Day Delivery*
- Delivery Option 3: *Two Day Delivery*
- Delivery Option 4: *One Day Delivery*

Average lead time for delivery option p is stored in the $DeliveryDays_p$ parameter. For instance, value of $DeliveryDays_2$ is set to 5 which indicates that if a customer chooses delivery option 2, their order is delivered to them within 5 days after it is placed. In real-world systems, delivery options are not selected by customers with the same likelihood. For example, delivery options with shorter lead time are more expensive and therefore less likely to be selected by customers who are cost aware. $DelProb_p$ parameter defines the probability of delivery option p getting selected by a customer. This parameter allows decision makers to set the probability of different delivery options based on their customer buying behavior.

The last aspect of delivery options that is modeled in this simulation is the delivery window. The delivery window is a range of time during the day of order delivery when the customer should expect to receive their package. In real-world systems, premium delivery options such as *One Day Delivery* and *Two Day Delivery* have a tighter delivery window than basic options. This is modeled by setting a shorter delivery window for premium delivery options in this simulation.

Table 3.6 summarizes all information for delivery options. Based on this table, if a customer places an order at 10:00 a.m. on a Monday and selects a *One Day Delivery* option they can expect to receive their order between 3:00 p.m. and 7:00 p.m. on Tuesday. If the customer selects a *Seven Day Delivery* instead, their order will be delivered between 8:00 a.m. and 7:00 p.m. on the Monday of the following week. Note that in this simulation model, 1:00 p.m. is considered the shipment cutoff time (lock time) and customer orders that are placed after this time are not eligible to ship on the same day. In other words, if a customer places an order at 1:05 p.m. on a Monday and selects a *One Day Delivery* option, their order is delivered between 3:00 p.m. and 7:00 p.m. on Wednesday.

Table 3.6: List of delivery options

Delivery option (p)	Description	$DeliveryDays_p$ (Days)	$DelProb_p$ (%)	Delivery window (Time of Day)
1	*Seven Day Delivery*	7	31	8:00 a.m. to 7:00 p.m.
2	*Five Day Delivery*	5	27	8:00 a.m. to 7:00 p.m.
3	*Two Day Delivery*	2	23	12:00 p.m. to 7:00 p.m.
4	*One Day Delivery*	1	19	3:00 p.m. to 7:00 p.m.

Customer orders. Customer orders in this simulation are modeled using the $order_r$ object which is a member of the $Order$ class that is described in Table 3.7. Unlike SKUs and FCs, the number of customer orders is not a known value and is a random variable. The time that elapses between two consecutive orders being placed is referred to as the inter-order-placement time. We use a probability distribution to model the variability of the inter-order-placement time. The type of this distribution is a model parameter that can be configured based on a specific e-tailer customer order placement rate. Additionally, we use the $AvgIntPlcTime$ parameter as the mean value of this probability distribution that can also be adjusted as needed.

As shown in Table 3.7, each customer order is assigned an $ordID$ that uniquely identifies that order in the system. The time an order is placed is stored in the $ordTime$ parameter. In addition, the day and hour of order placement are stored in the $ordDay$ and $ordHour$ parameters respectively.

The location of customer orders is modeled using the $ordLoc$ object which is a member of the $Location$ class that was described in Table 3.2. In other words, $order_r.ordLoc.xCoord$ and $order_r.ordLoc.yCoord$ store the x and y coordinates of the customer who places order r.

Table 3.7: Order class definition

$\boldsymbol{Class}$: $Order$	
$ordID$	Order ID, (integer, > 0)
$ordTime$	Clock time (in minutes) when order is placed, (real, ≥ 0)
$ordHour$	Hour of order placement time, (integer, $0 \leq ordHour \leq 23$)
$ordDay$	Day of order placement, (integer, ≥ 0)
$ordLoc$	Customer location, (instance of $Location$ class)
$delPref$	Delivery preference, (integer, $1 \leq delPref \leq NumDelPref$)
$estDel$	Estimated delivery time, (real, ≥ 0)
$mustLockTime$	Simulation clock time by which all assignments for the order should be locked, (real, ≥ 0)
$estLockTime$	Estimated time at which all assignments for the order will be locked, (real, ≥ 0)
$numItems$	Number of line items (i.e. unique SKUs) in the order, (integer, $1 \leq numItems \leq MaxOrdLines$)
$item_i$	Information pertaining to item i in customer order, (instance of $Item$ class)

In a real-world e-tailer system when a customer places an order, they select one of the delivery options that are available in the website. The $delPref$ parameter in the $Order$ class stores the delivery option selected by a customer in this simulation. For example, if a *Two Day Delivery* option is selected by the customer who places $order_r$, the value of $order_r.delPref$ is set to 3 based on the delivery options listed in Table 3.6.

The estimated delivery time for an order is calculated based on when it is placed and its delivery preference, and it is stored in $estDel$ parameter. Additionally, estimated delivery time determines how much time the e-tailer has before they must lock their fulfillment decision for the order. We assume that boxes are shipped once every day at 2:00 p.m. and the fulfillment decision for all orders that will be shipped that day must be locked at 1:00 p.m. This gives FCs enough time to prepare customer orders for shipment. Based on this assumption, the final deadline for locking customer order r ($mustLockTime$) is 1:00 p.m. on the day before its estimated delivery. This allows the e-tailer to use a *Next Day Air* shipping method to fulfill that customer order. This final deadline is stored in the $mustLockTime$ parameter. For example, if a customer places an order at 10:00 a.m. on a Monday and selects a *Five Day Delivery* preference, their estimated delivery time ($estDel$) is calculated as a random time between 8:00 a.m. and 7:00 p.m. on Saturday of the same week and the value of $mustLockTime$ for their order is set to 1:00 p.m. on Friday.

Although waiting until the final deadline and using a *Next Day Air* shipping method is a feasible decision, in a real-world system the e-tailer could find a more economical way to fulfill that order. For instance, in the previous example, if the e-tailer has enough inventory, it can use a *UPS Ground* (i.e. 5-day) shipping method to ship the customer order at 2:00 p.m. on Monday which is a cheaper option than waiting until Friday and using a *Next Day Air* shipping method. In following sections, we explain the fulfillment decision process that focuses on finding the best decision to minimize the shipping cost of each customer order. The $estLockTime$ parameter stores the estimated time at which the e-tailer will lock its fulfillment decision for order r. By definition, $estLockTime$ is less than or equal to $mustLockTime$.

The $numItems$ parameter stores the number of items (i.e. number of unique SKUs) requested in customer order r. This is modeled using the $Item$ class that is described in Table 3.8. Some customers order a single item while others order multiple items.

Table 3.8: Item class definition

$\boldsymbol{Class}$: $Item$	
SKU_ID	SKU ID, $(integer, 1 \leq skuID \leq NumSKUs)$
qty	Order quantity, $(integer, 1 \leq qty \leq MaxQuantity)$

In a real-world e-tailer system, the number of items varies from one customer order to another. An example dataset from OList, a Brazilian e-tailer, is presented in Table 3.9 (Kaggle website, 2020). This table shows the number of items requested in 98,000 historical customer orders. It can be observed that almost 90% of OList customer orders include a single item while less than 1% of them include more than four items. In addition, the maximum number of items in a single customer order in this sample dataset is 21 which occurs only in one instance. To model this, we use the $MaxOrdLines$ parameter to represent the maximum number of items in this simulation.

Additionally, $OrdLinesProb_i$ is the probability that i items are requested in an order. Values of $MaxOrdLines$ and $OrdLinesProb_i$ can be adjusted by decision makers based on real-world e-tailer historical demand patterns. For example, to model OList customer order patterns, $MaxOrdLines$ should be set to 21 and value of $OrdLinesProb_1$ through $OrdLinesProb_{21}$should be calculated based on the distribution that is presented in Table 3.9. Note that since $OrdLinesProb_i$ represents a probability distribution, the sum of its values must be equal to 1.

Table 3.9: Distribution of number of items in OList customer orders

Number of items	Frequency
1	88863
2	7516
3	1322
4	505
5	204
6	198
7	22
8	8
9	3
10	8
11	4
12	5
13	1
14	2
15	2
20	2
21	1

Another important aspect of customer orders that is modeled in this simulation is the distribution of SKUs in customer orders. Some SKUs are fast-moving and are ordered very frequently while others are slow-moving and have a sporadic ordering pattern. $SkuProb_s$ represents the probability that SKU s is the next SKU requested in any order. The value of $SkuProb_s$ can be calculated for all SKUs based on historical demand records of a real-world e-tailer. This allows decision makers to set a higher probability value to fast-moving SKUs. For example, consider a scenario in which the e-tailer has three SKUs in its product catalog. The e-

tailer's historical demand records indicate that in the past 1000 customer orders SKU_1, SKU_2, and SKU_3 have been ordered by 800, 500 and 200 customers respectively. According to this data, value of $SkuProb_1$, $SkuProb_2$ and $SkuProb_3$ should be set to 0.8, 0.5 and 0.2. Note that since $SkuProb_s$ is not a probability distribution, the sum of its values does not have to be equal to 1.

In order to model variability of order quantities, $QtyProb_q$ is used in this simulation which defines the probability that quantity associated with an item is equal to q. Like $SkuProb_s$, the values of $QtyProb_q$ could be set based on e-tailer historical demand. For instance, if 90% of a real-world e-tailer's single item orders have an order quantity of 1, value of $QtyProb_1$should be set to 0.9.

Regions. All customer orders are placed from within the e-tailer's area of operation. In a real-world e-tailer system, customer locations are not evenly distributed inside the area of operation. For instance, more orders are submitted from highly populated metropolitan areas than rural areas. Figure 3.4 illustrates the geographical distribution of a sample of OList customer orders that was presented earlier in this chapter. As the map shows, the number of orders that are submitted from eastern parts of the country is higher than the west. Additionally, it can be observed that certain regions in the east have a higher order density than others.

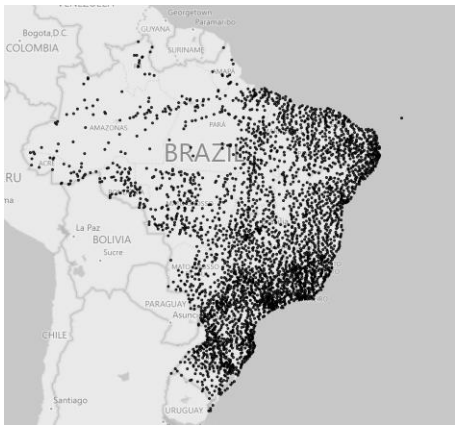

Figure 3.4: Geographical distribution of a sample of OList customer orders

To model the geographical distribution of customer orders, we split the e-tailer's area of operation into multiple rectangular regions. $NumRegions$ is a model parameter that allows decision makers to control the granularity of this division. By setting a higher value for this parameter, the area of operation is split into more regions and vice versa. The size and position of region g is modeled using the $region_g$ object which is a member of the $Region$ class that is described in Table 3.10. Each region is assigned a $regionID$ that uniquely identifies it in the system. The $llLoc$ and $urLoc$ parameters store the coordinates of the lower left and upper right corner of $region_g$ respectively.

Table 3.10: Region class definition

Class: $Region$	
$regionID$	Region ID, (integer, $1 \leq regionID \leq NumRegions$)
$llLoc$	Location of lower left corner of region, (instance of $Location$ class)
$urLoc$	Location of upper right corner of region, (instance of $Location$ class)

$RgnProb_g$ is a probability distribution that specifies the likelihood that a customer order is placed from $region_g$. By setting a higher value of $RgnProb_g$ for regions that represent highly populated areas within an e-tailer's area of operation, we can model the geographical distribution of customer orders.

A simple example is presented in Figure 3.5. As displayed in this figure, the e-tailer's area of operation is split into four regions. A $regionID$ is assigned to each region as well as a $RgnProb_g$ value. In this example, region 1 has the highest order rate with a probability value of 0.4 which indicates that 40% of the e-tailer's customer orders are submitted from a location within this region. The locations of customer orders are assumed to be uniformly distributed within each region.

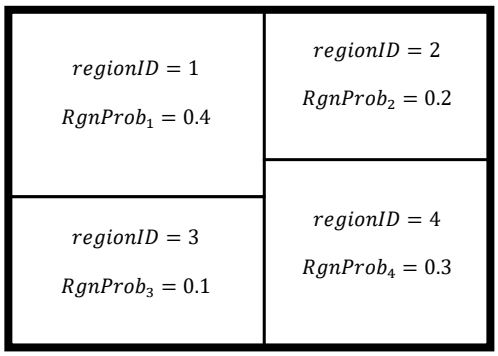

Figure 3.5: Modeling geographical distribution of customer orders using regions

Shipping methods. The difference between customer delivery options and shipping methods that are used by e-tailers for order delivery is explained in Chapter 1. Some e-tailers handle their own outbound transportation while most of them outsource it to 3PLs who offer various shipping methods for order delivery. In this model, the following UPS shipping methods are used for analysis.

- Shipping Method 1: *UPS Ground*
- Shipping Method 2: *Three Day Select*
- Shipping Method 3: *Second Day Air*
- Shipping Method 4: *Next Day Air*

We use $ShippingDays_m$ parameter to store the transit time for shipping method m. Table 3.11 provides a summary of these four shipping methods. We assume that the time it takes to ship a box is independent of the distance it is shipped. That is, the times shown in Table 3.11 are valid for shipping times from any FC to any customer. When a customer places an order, the e-tailer can choose a shipping method that meets the customer's delivery preference. For example, for an order with a *Seven Day Delivery* preference, the e-tailer can use any of the four shipping methods above. In the following sections, we present the shipping cost of different methods and explain how the e-tailer takes that into account for selecting a shipping method that minimizes its total shipping cost.

Table 3.11: List of shipping methods

Shipping method (m)	Description	$ShippingDays_m$ (days)
1	*UPS Ground*	5
2	*Three Day Select*	3
3	*Second Day Air*	2
4	*Next Day Air*	1

Shipments. In the simulation model, orders can be rejected if not enough open inventory is available to satisfy the order. If they can be satisfied, they are shipped in one or more boxes. As we explain in the next section, there is a fixed cost for shipping each box. This incentivizes the e-tailer to reduce the total number of boxes for fulfilling a customer order. In a real-world e-tailer system, there is a maximum weight capacity for a single box. If the total weight of the items ordered exceeds this threshold, the e-tailer must break the order shipment into multiple boxes. The maximum box weight varies from one e-tailer to another and is dependent on product types. In this simulation parameter $MaxBoxWeight$ is used to give the decision maker flexibility in setting this value based on its business requirements.

All shipments must be delivered to customers within their expected delivery window which is determined based on their delivery preference. As mentioned earlier, there are four shipping methods that can be used for order delivery. Since there is a fixed cost associated with picking orders from FCs, orders are not typically shipped one at a time and instead FCs batch multiple orders and ship them at a pre-determined time referred to as the shipment pick-up time. We assume shipment pick-up happens once a day at 2:00 p.m. Since picking items from inventory and getting them packed into boxes for shipment requires time, we assume that all fulfillment decisions for customer orders must be fixed by 1:00 p.m. on the day of shipment. This gives FCs enough time to meet the shipment deadline. All orders that are placed after 1:00 p.m. are therefore not eligible for being shipped on that day and must wait until following days.

Note that we assume all inventory replenishments reach their destination FC at 12:00 a.m. The inventory units that are received at an FC can be used to satisfy an order that is shipped from that FC at 2:00 p.m. on the same day. For instance, consider a scenario where a customer places an order at 11:00 p.m. on a Monday and requests 1 unit of SKU_1 with a *One Day Delivery* preference. If there is no on-hand inventory of SKU_1at any FC, but FC_1 expects a replenishment for that SKU at 12:00 a.m. on Tuesday, the e-tailer can accept the order and assign it to FC_1. In this case, the customer order is shipped at 2:00 p.m. on Tuesday and is delivered to the customer between 3:00 p.m. and 7:00 p.m. on Wednesday.

Consider another example in which a customer places an order at 1:01 p.m. on a Monday and requests 1 unit of SKU_1 with a *One Day Delivery* preference. In this case we assume that the e-tailer has enough inventory on-hand to satisfy this order. However, since the order is placed after the locking time (1:00 p.m.) it is not shipped on Monday and instead is sent out to the customer on

Tuesday at 2:00 p.m. The customer will receive their order between 3:00 p.m. and 7:00 p.m. on Wednesday.

Shipping cost. In this simulation model, shipping cost is calculated based on UPS shipping rates (UPS website, 2019). UPS uses a three-step process to calculate shipping cost.

Step 1: Specify shipping method. The first step in calculating shipping cost is identifying the shipping method that is used by the e-tailer. The list of available shipping methods and their details are provided in previous sections. Intuitively, shipping methods with shorter transit times have a higher shipping cost.

Step 2: Identify a UPS shipping zone. A UPS Shipping zone is specified based on the selected shipping method and distance between the origin and destination. A list of shipping zones for all shipping methods as well as the distance range for each is provided in Table 3.12. For instance, if a box is sent from a FC in Chicago to a customer in Detroit using a *Three Day Select* shipping method, since the total distance is approximately 280 miles the UPS shipping zone is identified as 303.

Table 3.12: List of UPS shipping zones based on shipping method and distance (in miles)

Shipping method (m)	Shipping method description	Minimum distance	Maximum distance	Shipping zone
1	*UPS Ground*	0	165	002
1	*UPS Ground*	166	308	003
1	*UPS Ground*	309	607	004
1	*UPS Ground*	608	1,020	005
1	*UPS Ground*	1,021	1,440	006
1	*UPS Ground*	1,441	2,020	007
1	*UPS Ground*	2,021	NA	008
2	*Three Day Select*	0	165	302
2	*Three Day Select*	166	308	303
2	*Three Day Select*	309	607	304
2	*Three Day Select*	608	1,020	305
2	*Three Day Select*	1,021	1,440	306
2	*Three Day Select*	1,441	2,020	307
2	*Three Day Select*	2,021	NA	308
3	*Second Day Air*	0	165	202
3	*Second Day Air*	166	308	203
3	*Second Day Air*	309	607	204
3	*Second Day Air*	608	1,020	205
3	*Second Day Air*	1,021	1,440	206
3	*Second Day Air*	1,441	2,020	207
3	*Second Day Air*	2,021	NA	208
4	*Next Day Air*	0	165	102
4	*Next Day Air*	166	308	103
4	*Next Day Air*	309	607	104
4	*Next Day Air*	608	1,020	105
4	*Next Day Air*	1,021	1,440	106
4	*Next Day Air*	1,441	2,020	107
4	*Next Day Air*	2,021	NA	108

Step 3: Calculate shipping cost. After identifying the shipping zone, UPS uses a step function to calculate the final shipping cost based on the box weight. This step function rounds up total box weight to full pounds and then uses a cost matrix to specify the final shipping cost. Table 3.13 displays a sample dataset from this cost matrix for UPS shipping zone 303. Based on this table, the total cost of shipping a box that weighs 8 pounds from a FC in Chicago to a customer in Detroit using the *Three Day Select* shipping method is $19.11.

Table 3.13: Sample shipping rates from UPS cost matrix

Shipment weight (lb.)	Shipping cost ($)
1	11.43
2	12.12
3	12.76
4	14.06
5	15.07
6	16.35
7	17.63
8	19.11
9	19.98
10	20.62

As can be observed from this sample dataset, the shipping cost per pound is not a constant value. In order to simplify this step function for the simulation model, linear regression is used to fit a linear model to shipping cost based on box weight. Table 3.14 summarizes the information for all fitted models and reports their accuracy using the R^2 metric. As shown in this table, the linear models estimate shipping cost with a very high degree of accuracy. A scatter plot of all UPS shipping costs and the fitted linear model for all the shipping zones is provided in Appendix A.

Table 3.14: Summary of fitted linear models for all shipping zones

Shipping Method	Shipping Zone	Fitted Model	Estimated Fixed Cost ($/box)	Estimated Variable Cost ($/lb)	R^2
UPS Ground	002	$y = 0.467x + 0.500$	0.500	0.467	0.93
UPS Ground	003	$y = 0.483x + 0.900$	0.900	0.483	0.96
UPS Ground	004	$y = 0.513x + 1.370$	1.370	0.513	0.97
UPS Ground	005	$y = 0.502\,x + 6.011$	6.011	0.502	0.99
UPS Ground	006	$y = 0.543\,x + 9.707$	9.707	0.543	0.99
UPS Ground	007	$y = 0.576\,x + 13.629$	13.629	0.576	0.99
UPS Ground	008	$y = 0.636\,x + 15.849$	15.849	0.636	0.98
Three Days Select	302	$y = 0.808\,x + 7.23$	7.23	0.808	0.99
Three Days Select	303	$y = 1.013\,x + 9.874$	9.874	1.013	0.99
Three Days Select	304	$y = 1.328\,x + 9.284$	9.284	1.328	0.99
Three Days Select	305	$y = 1.745\,x + 9.296$	9.296	1.745	0.99
Three Days Select	306	$y = 2.525\,x + 12.312$	12.312	2.525	0.99
Three Days Select	307	$y = 2.970\,x + 11.164$	11.164	2.970	0.99
Three Days Select	308	$y = 3.329\,x + 11.692$	11.692	3.329	0.99
Two Days Air	202	$y = 1.181\,x + 10.348$	10.348	1.181	0.99
Two Days Air	203	$y = 1.338\,x + 15.010$	15.010	1.338	0.99
Two Days Air	204	$y = 1.772\,x + 15.223$	15.223	1.772	0.99
Two Days Air	205	$y = 2.590\,x + 17.495$	17.495	2.590	0.99
Two Days Air	206	$y = 4.265\,x + 19.846$	19.846	4.265	0.99
Two Days Air	207	$y = 4.525\,x + 24.643$	25.643	4.525	0.99
Two Days Air	208	$y = 4.760\,x + 24.013$	24.013	4.760	0.99
Next Day Air	102	$y = 2.073\,x + 11.562$	11.562	2.073	0.99
Next Day Air	103	$y = 3.012\,x + 16.57$	16.57	3.012	0.99
Next Day Air	104	$y = 4.998\,x + 35.206$	35.206	4.998	0.99
Next Day Air	105	$y = 5.410\,x + 41.148$	41.148	5.410	0.99
Next Day Air	106	$y = 5.543\,x + 49.559$	49.559	5.543	0.99
Next Day Air	107	$y = 6.039\,x + 45.559$	45.559	6.039	0.99
Next Day Air	108	$y = 6.345\,x + 45.379$	45.379	6.345	0.99

Using these linear models, the shipping cost for each zone is broken down into two separate components. The first component is a fixed charge which does not depend on box weight and the second component is a variable cost which is dependent on box weight. For instance, for shipping an 8-pound box from a FC in Chicago to a customer in Detroit using a *Three Day Select* shipping method, there is a fixed cost of $9.874 and a variable cost of $1.103 per pound. Therefore, total shipping cost for this box is estimated as $18.698 using the regression model which has less than 3% error when compared with the actual shipping cost of $19.11 from the UPS cost matrix.

Several important patterns can be detected in Table 3.14. First, the fixed shipping cost components are considerably higher than the variable components. Therefore, minimizing the number of boxes that are shipped to customers can reduce total shipping cost. Xu et al. (2009) use this principle in designing an order fulfillment reevaluation algorithm that minimizes total number of boxes by shuffling order assignments. However, there are other patterns in this data that need

to be considered. There is a significant difference between the cost of various shipping methods. In the previous example of sending an 8-pound box from a FC in Chicago to a customer in Detroit, if the e-tailer uses *UPS Ground* shipping method instead of *Three Day Select*, they can reduce total shipping cost from $19.11 to $4.77 and save $13.94. This indicates that minimizing number of boxes does not necessarily minimize the total cost. In some cases, sending more boxes using cheaper shipping methods could result in an overall lower shipping cost. In the following sections, we discuss additional aspects of the online order fulfillment process that must be considered by our proposed algorithms.

3.2.2. System state for e-tailer DES model

The system state is the dynamic aspect of a simulation model that evolves over time as new events occur. The system state for an e-tailer order fulfillment process is composed of several entities and relationships that are explained in this section. To model these entities, we use a class called $SysState$. Table 3.15 provides a definition of this class.

Table 3.15: SysState class definition

$\boldsymbol{Class}$: $SysState$	
$orderQueue_r$	List of all orders, (instance of $Order$ class)
$openOrderNumbers$	Order numbers for open orders, (integer, > 0)
$closedOrderNumbers$	Order numbers for closed orders, (integer, > 0)
$numOrders$	Total number of orders that have been placed, (integer, ≥ 0)
$numOpenOrders$	Total number of open orders, (integer, ≥ 0)
$numClosedOrders$	Total number of closed orders, (integer, ≥ 0)
$numAssignments$	Total number of assignments, (integer, ≥ 0)
$assignments_a$	List of current assignments, (instance of $Assignment$ class)
$inv_{s,f}$	Inventory status for SKU s at FC f, (instance of $Inventory$ class)
$timeOfMostRecentEvent$	Time of most recent event, (real, > 0)

Order queue. The order queue keeps a record of all customer orders that have been placed. Figure 3.6 provides a conceptual view of the order queue. As shown in this figure, each customer order is modeled as an independent entity. The order queue consists of a collection of these entities that together capture the relevant information of all customer orders that have been placed since

the beginning of the simulation. We use $orderQueue_r$ to store the information of customer order r. The $numOrders$ parameter is also used to capture total number of customer orders that have been placed.

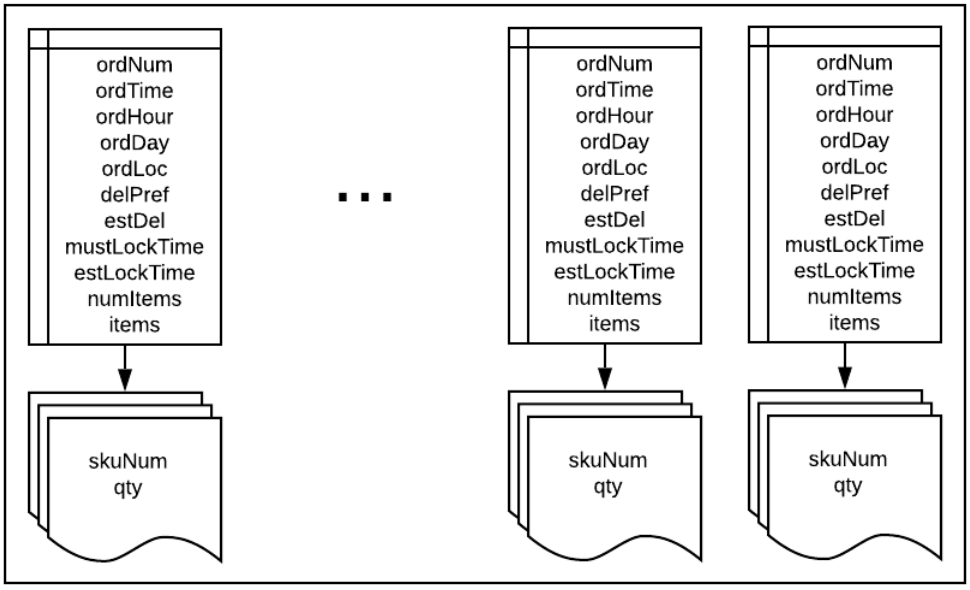

Figure 3.6: Conceptual diagram of customer order queue

Open orders and closed orders list. In addition to the order queue, two other lists are used in the system state to separate open orders from closed orders. When a new order is placed, its order number is added to $openOrderNumbers$ list and it remains there until it is either rejected or all its items are shipped out from FCs. At that point, its order number is removed from the $openOrderNumbers$ list and is added to the $closedOrderNumbers$ list. Note that since all information about an order is already stored in $orderQueue$, the $openOrderNumbers$ and $closedOrderNumbers$ lists only include the order numbers.

Inventory level. The inventory level is the amount of inventory that is available at the e-tailer at a particular time during the simulation. In this model, inventory is tracked using six inventory

types for each SKU and FC combination. Figure 3.7 illustrates these inventory types. As shown in this figure, all inventory units that are physically present at an FC are referred to as *on-hand*, while the units that are ordered but have not reached the destination FC are considered *on-order*. Inventory units that are not assigned to a customer order are tagged as *unassigned* and otherwise they are considered *assigned*. If the order that an inventory unit is assigned to is locked for shipment, that inventory unit is referred to as *locked.* Inventory units that are assigned to orders that are not locked for shipment are considered *open.*

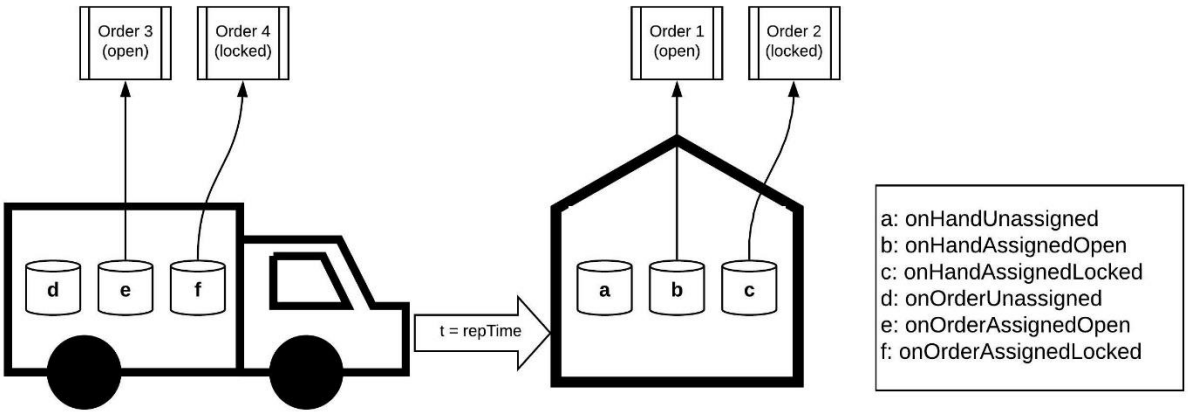

Figure 3.7: Inventory types in the system state

To store inventory information for SKU s at FC f in the system state, we use the $inv_{s,f}$ object which is a member of $Inventory$ class that is described in Table 3.16. As shown in this table, there is a variable for each of the six inventory types in this class. These variables are used to track inventory levels during the simulation.

Table 3.16: Inventory class definition

$\boldsymbol{Class}$: $Inventory$	
FC_ID	FC number, (integer, $1 \leq FC\_ID \leq NumFCs$)
SKU_ID	SKU number, (integer, $1 \leq SKU\_ID \leq NumSKUs$)
$onHandUnassigned$	On hand inventory units that are not assigned to any order, (integer, ≥ 0)
$onHandAssignedOpen$	On hand inventory units that are assigned to an order but can be reevaluated, (integer, ≥ 0)
$onHandAssignedLocked$	On hand inventory units that are assigned to an order and cannot be reevaluated, (integer, ≥ 0)
$onOrderUnassigned$	On order inventory units that are not assigned to any order, (integer, ≥ 0)
$onOrderAssignedOpen$	On order inventory units that are assigned to an order but can be reevaluated, (integer, ≥ 0)
$onOrderAssignedLocked$	On order inventory units that are assigned to an order and cannot be reevaluated, (integer, ≥ 0)
$repTime$	Expected simulation clock time of next replenishment, (real, ≥ 0)

Figure 3.8 shows how inventory types change as different events occur. Inventory is first created when an FC places a replenishment order for a SKU and all inventory units in that replenishment order are tagged as *on-order-unassigned*. If the e-tailer assigns some of those units to customer orders before they reach the FC, their type changes to o*n-order-assigned-open.* If the order that those inventory units are assigned to gets locked for shipment while the replenishment order is still in transit, their type becomes o*n-order-assigned-locked.* When the replenishment order reaches the destination FC, all inventory units with *on-order* types are added to the corresponding *on-hand* type. For example, inventory units that are *on-order-unassigned* are added to existing *on-hand-unassigned* inventory units. The inventory type transition ends when an inventory unit is shipped to a customer as a result of shipment event. More information about events are provided in the following sections.

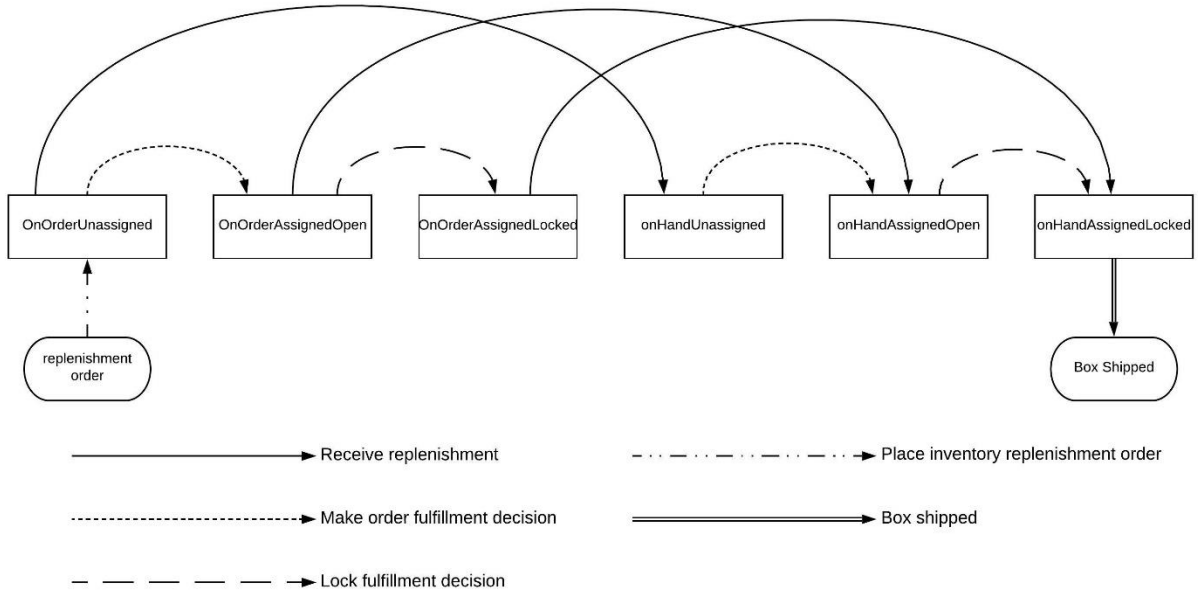

Figure 3.8: Transition logic between different inventory types

Assignments. When the e-tailer makes a fulfillment decision for an order, the decision is stored in the system state as a set of assignments. Each assignment specifies the responsible FC for fulfilling an item in the customer order in addition to information about when and how that item will be shipped. In other words, if a customer order contains multiple items, a separate assignment is generated and stored in the system state for each item. In some cases, if the e-tailer decides to split a single-item order between multiple FCs, multiple assignments will be generated for that item. In this simulation we use the $Assignment$ class to store this information in the system state. Table 3.17 provides the definition for this class as well as the variables that are used to capture different aspects of an assignment.

Table 3.17: Assignment class definition

Class: *Assignment*	
$asgID$	Assignment ID, (integer, > 0)
$ordID$	Order ID, (integer, > 0)
$skuID$	SKU ID, (integer, $1 \leq skuID \leq NumSKUs$)
$fcID$	FC ID, (integer, $1 \leq fcNumber \leq NumFCs$)
$qtyFromOnHand$	Quantity that is assigned from on hand inventory, (integer, ≥ 0)
$qtyFromOnOrder$	Quantity that is assigned from on order inventory, (integer, ≥ 0)
$shippingTime$	Scheduled shipping time for assignment, (real, > 0)
$lockingTime$	Scheduled locking time for assignment, (real, > 0)
$shippingMethod$	Shipping method, (integer, $1 \leq shippingMethod \leq 4$)
$locked$	Is this assignment locked or it can be modified? (binary)

Figure 3.9 illustrates the relationship between customer orders, fulfillment decisions and assignments. In this example, a customer order is placed at time $t = 0$ with two items. The first item is one unit of SKU_1 and the second item is two units of SKU_2. This customer has requested $delPref_1$ which, as described previously, is *Seven Day Delivery*. According to this delivery preference, the $mustLockTime$ for this order is estimated as $t = 9420$ which is 1:00 p.m. of the day before the delivery deadline. The e-tailer makes a fulfillment decision to identify the best option to fulfill the order given its current system state. The results of this decision are stored in two assignment objects. The first assignment indicates that SKU_1 will be fulfilled by FC_1 using on-hand inventory. This assignment will be locked at $t = 3660$ minutes and shipped at $t = 3720$ minutes. Order locking time and shipping time correspond to 1:00 p.m. and 2:00 p.m. two days after the order is placed. Additionally, the e-tailer has decided to use $shippingMethod_1$ which is *UPS Ground* and has an average transit time of 5 days. The second assignment indicate that SKU_2 will also be fulfilled by $FC_1$with a similar shipping method, locking time and shipping time. The only difference in the second assignment is that the e-tailer has decided to fulfill SKU_2 by using one unit of on-hand inventory and one unit of on-order inventory. This might be because the e-tailer does not have enough on-hand inventory at FC_1 but is expecting a replenishment for SKU_2 before order locking time. Note that both assignments are not locked at this time and can be

changed if needed. Finally, the value of $estLockTime$ for customer order is set according to these assignments. In this case, since both assignments for this order are locked at $t = 3660$, the $estLockTime$ for the order is also set to the same value.

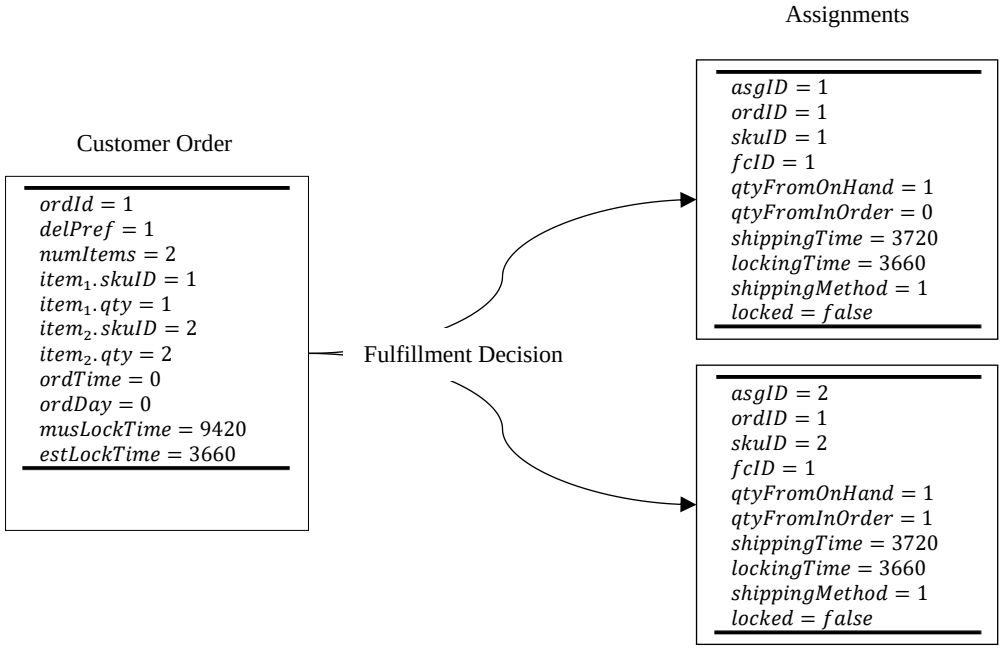

Figure 3.9: Relationship between customer orders, fulfillment decisions and assignments

3.2.3. Events for e-tailer DES model

The e-tailer order fulfillment process is a highly complex system that involves many activities and events. An event triggering diagram is provided in Figure 3.10 which illustrates the main events that are considered in this DES model and the relationships between them. A solid arrow indicates possible instant triggering of the downstream event resulting from the occurrence of the upstream event. A dashed arrow indicates a guaranteed placement of the downstream event into the calendar whenever the upstream event occurs.

The overall flow of this DES model begins with a *customer places order* event. When an order is placed, the e-tailer immediately checks inventory levels at all FCs to determine whether it can be fulfilled. A customer order is accepted if there is enough inventory to satisfy all order items; otherwise the e-tailer rejects that order. The next step for accepted orders is making a fulfillment decision that determines the best option to fulfill the order with minimum cost while meeting customer expectations including the requested delivery deadline. Note that, when a customer order is placed, it triggers a future order placement which is added to the event calendar.

Inventory replenishment at FCs are independent of customer order placement and occur periodically based on the e-tailer's inventory policy. The periodicity of this event means that every replenishment order triggers the next one and this cycle continues throughout the simulation. There is a lead time for receiving a replenishment order; hence this event is added to the event calendar to be executed in the future.

It is assumed that shipments are picked up from all FCs at 2:00 p.m. every day. In order to prepare for this event, all fulfillment decisions for orders that are due for shipment that day are locked at 1:00 p.m. This gives the e-tailer enough time to pick items from inventory and pack them into boxes that will be shipped out to customers. This is depicted in Figure 3.10 by showing "lock fulfillment decision" as a recurring event that schedules customer order shipments to take place in the future.

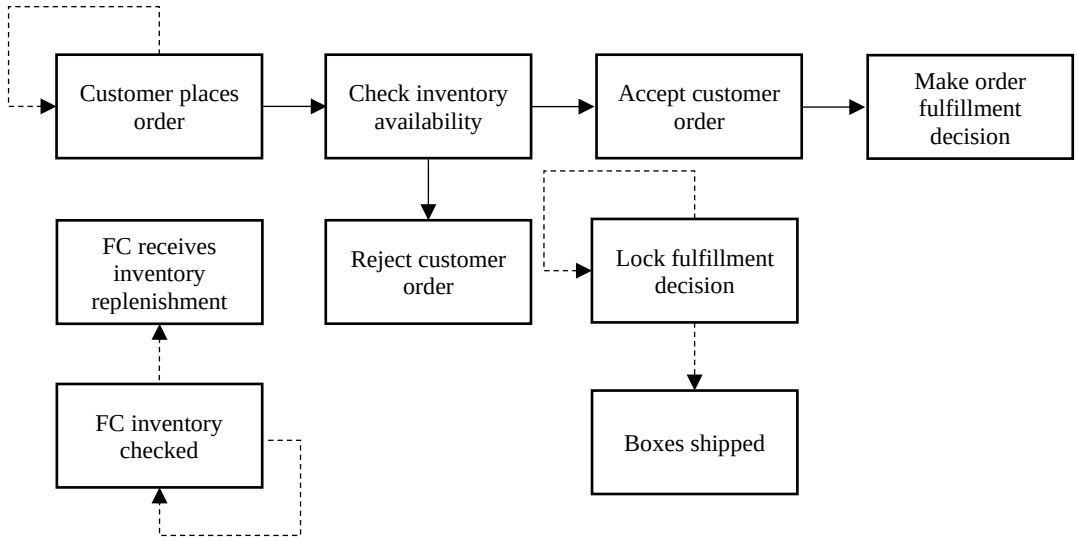

Figure 3.10: Event triggering diagram for discrete event simulation model

The remainder of this section explains these events in more detail. The pseudocode for all events is provided in Appendix B.

Customer places order. This is the beginning of main simulation flow. When a new customer places an order, the first step is to gather and generate information about it. This includes the order ID, order placement time, customer location, delivery preference, expected delivery window, number of items and SKU and quantity of each item.

- Order ID: a sequential numbering system is used in this simulation to generate an integer order ID for each customer order.
- Placement time: this is same as the current simulation time and can be directly derived from that. In addition to order placement time, the day and hour of this event is calculated and stored in the system state.
- Customer location: orders are assumed to be placed from a location within the e-tailer's area of operation. As described previously, the area of operation is split into multiple regions and $RgnProb_g$ is the probability that a customer order originates from region g. When the *customer places order* event is triggered, a random number is generated to determine the location of the new order based on the values of $RgnProb_g$ for all g.
- Delivery preference: Four delivery options are considered in this simulation. To determine which delivery option is selected by $order_r$, we use a random number generator and $DelProb_p$ which specifies the probability that a customer selects delivery option p for their order.
- Expected delivery window: this is calculated based on when the order is placed and the customer's selected delivery preference. If an order is placed after 1:00 p.m., which is the locking time for all assignments, it is not eligible for same day shipment and must wait until the following day.
- Number of items: a discrete probability distribution is used to determine number of items in the customer order.

- SKUs: a different SKU is selected for each item that is ordered. Some SKUs are ordered more frequently than others. To model these differences a probability distribution is used that gives popular SKUs a higher likelihood of being included in a customer order.
- Order quantity: The quantity of an item is correlated to the SKU in that item. For instance, it is more likely for a customer to order multiple pens in a single order than it is for them to order multiple computers. To model this, a probability distribution is used whose value changes based on the SKU. This allows e-tailers to set a different value for this distribution for each of their SKUs to accurately capture their customer order pattern.

Once the order information is generated, the next step is to update the statistical accumulators and system state. The only statistical accumulator that is updated by this event is $TotalDemand$. The new order is added to the end of customer order queue in the system state and is considered as an open order. At the end of this event, the *check inventory availability* event is triggered to decide whether this order can be fulfilled. A future *customer places order* event is also added to the event calendar.

Check inventory availability. When a customer places an order, the e-tailer immediately checks its inventory levels to decide if all items in that order can be fulfilled. The *check inventory availability* event performs this task by calculating total eligible inventory for each SKU in the customer order at all FCs. While all *on-hand-unassigned* inventory units are eligible to satisfy a customer order, *on-order-unassigned* inventory units are only eligible if they are scheduled to arrive at a FC before the order must be locked.

Once the total eligible inventory units for all SKUs in the customer order are calculated, they are compared with customer order quantity to determine whether the order should be accepted or

rejected. If there is enough eligible inventory to satisfy all order items, the order is accepted; if any of the items in the order cannot be fulfilled, the order is rejected.

Accept customer order. This event increases the $NumOrdersAccepted$ statistical accumulator by one unit and subsequently calls the *make order fulfillment decision* event to determine how this order is fulfilled.

Reject customer order. This event increases the $NumOrdersRejected$ statistical accumulator by one unit. It then updates the system state by moving the rejected order from the open orders list to the closed orders list.

Make order fulfillment decision. As described previously, the *check inventory availability* event determines if an order can be fulfilled given current inventory levels at all FCs. However, it does not specify how the order is fulfilled. This decision is made when the *make order fulfillment decision* event is triggered. This event uses a greedy heuristic algorithm for assigning a customer order to FCs.

The algorithm begins with ranking the FCs based on number of the items they can fulfill. FCs that can fulfill more items are ranked higher. If two FCs can fulfill the same number of items, the FC that is closer to the customer's location gets a higher rank. After ranking FCs, the algorithm assigns each item to the FC with the highest rank that can fulfill it. If one of the order items cannot be fulfilled by a single FC, it is split between multiple FCs. By following this process, if the FC with the highest rank can fulfill all order items, then the customer order is shipped from a single FC and vice versa.

When assigning order items to FCs, the algorithm also determines the shipping method and shipping day for the assignment based on the customer delivery preference and inventory

replenishment lead time. Table 3.18 summarizes all combinations that are considered. As shown in this table, the algorithm finds the cheapest shipping method that can satisfy the customer's delivery preference while considering the inventory replenishment lead time constraint. It then finds the latest time for shipping out the customer order based on the selected shipping method and delivery preference. For example, if a customer selects a *Seven Day Delivery* preference and the FC has on-hand inventory to fulfill the order, the algorithm picks a *UPS Ground* shipping method (M1) which takes an average of five days to deliver a shipment. It then decides to hold this shipment for 2 days before sending it out. The locking time for the assignment is always set on the same day as the shipment. This delay not only allows the e-tailer to tend to more urgent customer orders in the meantime, but also increases the opportunity to reevaluate assignments for this order within these two days. After all assignments are determined by the algorithm, they are added to the list of active assignments in the system state which makes them available to subsequent events in the simulation.

Table 3.18: Shipping method and shipping day determination
LT = inventory replenishment lead time, MX|D indicates order will be shipped D days after it is placed using method X

Delivery preference	LT: 0	LT: 1	LT: 2	LT: 3	LT: 4	LT: 5	LT: 6
Seven Day Delivery	M1 | 2	M1 | 2	M1 | 2	M2 | 4	M2 | 4	M3 | 5	M4 | 6
Five Day Delivery	M1 | 0	M2 | 2	M2 | 2	M3 | 3	M4 | 4	N/A	N/A
Two Day Delivery	M3 | 0	M4 | 1	N/A	N/A	N/A	N/A	N/A
One Day Delivery	M4 | 0	N/A	N/A	N/A	N/A	N/A	N/A

Lock fulfillment decision. All assignments that are generated by the *make order fulfillment decision* event have a locking time associated with them that is determined based on their shipping method and shipping time. *Lock fulfillment decision* is a periodic event that occurs every day in the simulation at 1:00 p.m. This event checks all assignments that are stored in system state and locks those that have a locking time that matches current simulation time. In addition to locking

assignments, this event locks inventory units in the FCs that are used in those assignments. Just before it ends, this event puts a *boxes shipped* event in event calendar.

Boxes shipped. As mentioned earlier, *boxes shipped* event occurs every day at 2:00 p.m. This event reviews all assignments that are currently stored in system state and finds the ones that are locked for shipment. It then calculates the shipping cost for each order based on the fixed and variable shipping rate that is presented in Table 3.14. The calculated cost is added to the $TotalShippingCost$ statistical accumulator. At the end, this event updates inventory levels at all FCs by removing inventory units that are in boxes that are shipped.

FC inventory replenishment. This DES model follows a periodic inventory review policy to replenish the inventory at all FCs. Each SKU at each FC has a review cycle associated with it that specifies the periodicity of its replenishment. There is also a $maxLevel$ parameter that determines the size of replenishment. When the *FC inventory replenishment* event is triggered, it reviews the current inventory level for each SKU at each FC and then places replenishment orders based on this information. There is a lead time associated with receiving replenishment orders. This lead time also depends on the SKU and FC for which the replenishment order is placed. As a result of this event, a *receive replenishment order* event is placed in the event calendar which will be triggered when the lead time is reached. The inventory units that are ordered are added to the list of on-order inventory units in the system state.

Receive replenishment. Once a replenishment order reaches its destination FC, the *receive replenishment event* is triggered to update the system state accordingly. For every SKU-FC combination in the replenishment order, this event updates inventory levels by adding the on-order inventory units to on-hand inventory and subsequently setting the on-order inventory units to zero.

It also checks all the assignments in the system state that use the arriving on-order inventory units and updates them by changing that value to on-hand inventory instead.

3.2.4. Statistical accumulators for e-tailer DES model

There are four statistical accumulators in this DES model that are listed in Table 3.19. These accumulators are updated during the simulation to monitor system performance. The primary performance metric in this study is $TotalShippingCost$ which is the combined cost of fulfilling all customer orders that are placed over the course of simulation. This metric is used to compare different fulfillment strategies.

Table 3.19: Statistical accumulators

Statistical accumulators	
$TotalDemand_s$	Total demand for SKU s, (integer, ≥ 0)
$NumOrdersAccepted$	Number of orders that are accepted, (integer, ≥ 0)
$NumOrdersRejected$	Number of orders that are rejected, (integer, ≥ 0)
$TotalShippingCost$	Total shipping cost, (real, ≥ 0)

In addition to $TotalShippingCost$ other statistical accumulators are used in this model which help decision makers to analyze the system from different standpoints beside shipping cost. $NumOrdersAccepted$ and $NumOrdersRejected$ provide insight into the e-tailer's service level which is a very important KPI. The e-tailer service level is computed as $\left(\frac{NumOrdersAccepted}{NumOrdersAccepted+NumOrdersRejected}\right)$. If a fulfillment strategy reduces total shipping cost but negatively impacts service level by rejecting more orders, it might not be an ideal strategy for an e-tailer whose primary objective is to increase the service level. In this dissertation we assume total shipping cost is the main focus, but this model can also be used to study service level and

find a fulfillment strategy to optimize that. Lastly, $TotalDemand_s$ is an accumulator which represents the total number of units of SKU s that have been ordered.

This chapter provided a comprehensive overview of the DES model of an e-tailer order fulfillment process. The rule-based algorithm used in this model to make order fulfillment decisions in the *order fulfillment decision event* only considers one order at a time which leads to a set of myopic assignments. We also observed that e-tailers do not immediately ship a customer order and there is a window of time between when a customer order is placed and when it is locked and shipped out of the FCs. The next chapter presents a math model that takes advantage of this window to revise fulfillment decisions by optimizing them for a group of customer orders. This model has the potential to generate significant cost savings for the e-tailer while maintaining an adequate customer service level.

Chapter 4

Integer program for reevaluating order fulfillment plans

4.1. Problem definition

When a customer places an order, the e-tailer makes an order fulfillment decision to determine the best option to fulfill that order with minimum cost. Since this decision is made independently for each customer order, it leads to a myopic decision that might not be optimal at the system level. In the previous chapter a simulation model of an e-tailer order fulfillment process was presented which includes a greedy rule-based algorithm for making individual order fulfillment decisions. In Chapter 1 we analyzed the fulfillment process and identified a window of opportunity between the time a fulfillment decision is made for an order and the time the items are processed by the e-tailer and shipped to the customer. This window can be used to reevaluate fulfillment decisions for a group of active customer orders to find a decision that reduces total shipping cost.

In this chapter an integer program is presented for reevaluating order fulfillment decisions for a set of customer orders. This integer program is intended to be executed several times during a day to reduce an e-tailer's total shipping cost. In this program, we assume R customer orders are reevaluated together. A total of F FCs exist in the e-tailer supply chain that hold inventory and ship customer orders. There are S distinct SKUs within the customer orders that are reevaluated. Since the set of customer orders might not include all SKUs in the e-tailer's product catalog, S is less than or equal to the total number of SKUs on the e-tailer's website. Each customer order includes one or more items and has a promised delivery day that is determined by the customer delivery preference. All items in each order must be delivered to customer before the promised

delivery day. In other words, reevaluating order fulfillment decisions should not negatively impact an e-tailer's customer service level.

Inventory units that are assigned to individual orders by the myopic fulfillment decisions are added to a shared inventory pool and used by the reevaluation algorithm for improving assignments. This includes both on-hand as well as on-order inventory units. This guarantees that the reevaluation algorithm will always be able to find at least one feasible decision which is the same as the myopic decisions. This is critical because, as mentioned earlier, reevaluation must not impact service level; all customer orders and commitments must be met. In addition to this, a portion of the un-assigned inventory units at each FC gets locked and is made available to the reevaluation algorithm to improve its decisions. That inventory may not be used to fulfill other customer orders that are placed while reevaluation is being executed.

Since this integer program is used to improve the myopic decisions that are made by a greedy rule-based algorithm, the integer program uses the same shipping methods and shipping rates to keep the results consistent and comparable. There are four shipping methods that are adopted from the UPS website: *UPS Ground*, *Three Day Select*, *Two Day Air* and *Next Day Air*. The shipping rates for these methods are presented in Table 3.14.

The math model assumes that ordered items are packaged into boxes and shipped to customers at a predetermined time each day which is called the shipment pick-up time. In our math model we assume that there is a maximum weight limit for a single box and if the total weight of items shipped to a customer from a FC exceeds this limit, the shipment must be split into multiple boxes. Additionally, as in the simulation model we assume that the locking time is 1:00 p.m. and the shipment pick-up time is 2:00 p.m. every day.

Note that this mathematical model is executed regularly and at different times during the e-tailer's operations. When the reevaluation is triggered, the e-tailer takes a snapshot of the orders and inventory levels at all FCs and uses that information to construct an instance of this mathematical model to optimize the assignments for those orders. In this dissertation we assume that the e-tailer can choose to trigger the reevaluation algorithm either for a fixed number of orders or at fixed time intervals. The first method enables the e-tailer to find a batch size (i.e. number of orders in a single reevaluation) that performs best for its business and use that to specify when reevaluation should be triggered. The second method enables the e-tailer to find a time interval that produces the best results and use that to trigger the reevaluation. Intuitively, if the e-tailer uses the first method, the elapsed time between two consecutive reevaluations will be variable; by following the second method on the other hand the elapsed time between two consecutive reevaluations is constant but number of customer orders that are reevaluated together will be variable.

4.2. Mathematical formulation

In this section the integer programming model is presented. The set of indices, parameters and decision variables used in the mathematical program, and their respective explanations, are given in Table 4.1.

Table 4.1: Indices, parameters and decision variables in integer program

Indices	
s	SKU ($1 \leq s \leq S$)
d	Day ($0 \leq d \leq D$)
m	Shipping method ($1 \leq m \leq M$)
f	FC ($1 \leq f \leq F$)
r	Customer order ($1 \leq r \leq R$)
Parameters	
S	Number of unique SKUs requested within the orders being reevaluated
F	Number of FCs
R	Number of customer orders being reevaluated
M	Number of shipping methods
D	Number of days in reevaluation horizon
$weight_s$	Weight of SKU s in pounds
$maxBoxWeight$	Maximum weight in pounds allowed in a single box, (integer, > 0)
$cBox_{mfr}$	Fixed shipping cost for sending a box from FC f to the customer who placed order r using shipping method m
$cPound_{mfr}$	Shipping cost per pound from FC f to the customer who placed order r using shipping method m
$ordQty_{sdr}$	Number of units of SKU s in customer order r that are requested to be delivered by day d
$cQty_{sdr}$	Total (cumulative) number of units of SKU s that must be delivered by day d to the customer who placed order r
$inventory_{sdf}$	Number of units of SKU s that arrive at FC f on day d
$cInventory_{sdf}$	Cumulative number of units of SKU s that arrive at FC f on or before day d
$transitDays_{dm}$ $\begin{cases}1\\0\end{cases}$	If transit time for shipping method m is exactly d days Otherwise
$delivery_r$	Promised delivery day for customer order r, (integer, $1 \leq delivery_r \leq D$)
Decision Variables	
X_{sdmfr}	Number of units of SKU s that are shipped on day d using shipping method m from FC f to the customer who placed order r, ($0 \leq d \leq D-1$)
B_{dmfr}	Number of boxes shipped out of FC f to the customer who placed order r using shipping method m on day d, ($0 \leq d \leq D-1$)
W_{mfr}	Total weight of shipment from FC f to the customer who placed order r using shipping method m, ($0 \leq d \leq D-1$)
U_{sdf}	Total units of SKU s that are shipped out of FC f on day d, ($0 \leq d \leq D-1$)
V_{sdf}	Total units of SKU s that are shipped out of FC f on or before day d, ($0 \leq d \leq D-1$)
Y_{sdr}	Total units of SKU s that are delivered on day d to the customer who placed order r, ($1 \leq d \leq D$)
Z_{sdr}	Total units of SKU s that are delivered on or before day d to the customer who placed order r, ($1 \leq d \leq D$)

The input data consists of several primary parameters as well as secondary parameters that are derived from primary parameters. A detailed description of some of these parameters is provided below.

- D: This is a primary parameter that indicates time horizon for reevaluation. In this dissertation, since the longest delivery option is *Seven Day Delivery*, we set the value of D to 7.

- $cBox_{mfr}$: This is a primary parameter that indicates the shipping cost for sending a box from FC f to customer order r using shipping method m. The value of this parameter is calculated using the procedure that was explained in Chapter 3.
- $cPound_{mfr}$: This is a primary parameter that indicates the shipping cost per pound from FC f to customer order r using shipping method m. The value of this parameter is calculated using the procedure that was explained in Chapter 3.
- $ordQty_{sdr}$: This is a primary parameter that indicates the number of units of SKU s in customer order r that are requested to be delivered by day d. The value of this parameter is set based on information about the order. In this case, d refers to the promised delivery day for the order.
- $cQty_{sdr}$: This is a secondary parameter which is derived from $ordQty_{sdr}$ and indicates the total (cumulative) number of units of SKU s that must be delivered by day d to the customer who placed order r. If the value of $ordQty_{sbr}$ is greater than 0, that value will be applied to $cQty_{sdr}$ for all d which is less than or equal to b. For example, if $ordQty_{131}$ is 10 and $ordQty_{161}$ is 2, then the value of $cQty_{1d1}$ for d between 0 and 7 is calculated as: {0,0,0,10,10,10,12,12}
- $inventory_{sdf}$: This is a primary parameter that indicates total number of inventory units for SKU s that become available on day d at FC f. In this case on-hand inventory units are assumed to be available on day 0 and any on-order inventory units are assumed to be available on the day of replenishment order delivery.
- $cInventory_{sdf}$: This is a secondary parameter which is derived from $inventory_{sdf}$ and indicates the cumulative number of inventory units of SKU s that arrive at FC f on or before day d. The value of $cInventory_{sbf}$ is the summation of $inventory_{sdf}$ for all d less

than or equal to b. For example, if the value of $inventory_{101}$ is 5 and the value of $inventory_{131}$is 10 then the value of $cInventory_{1d1}$ for d between 0 and 7 is calculated as: {5,5,5,15,15,15,15,15}

- $transitDays_{dm}$: This is a primary parameter that indicates the transit time for each shipping method. If shipping method m has a transit time of d days value of $transitDays_{dm}$ is equal to 1 otherwise, it is set to 0.
- $delivery_r$: This is a primary parameter that indicates promised delivery day for customer order r.

In addition to these parameters, there are seven integer decision variables in the model forming our integer program. X_{sdmfr} is the primary decision variable that indicates the number of units of SKU s that are shipped on day d using shipping method m from FC f to the customer who placed order r. B_{dmfr} and W_{mfr} are secondary decision variables that are derived from X_{sdmfr}. The B_{dmfr} decision variable is defined as the number of boxes that are shipped out of FC f to the customer who placed order r using shipping method m on day d. The W_{mfr} decision variable is defined as the total weight of shipments from FC f to the customer who placed order r using shipping method m. U_{sdf} and V_{sdf} are used in the integer program to ensure that the total units of SKU s that are shipped out of FC f on day d do not exceed available inventory. More specifically, U_{sdf} represents the total units of SKU s that are shipped out of FC f on day d while V_{sdf} represents the total units of SKU s that are shipped out of FC f on or before day d. Finally, Y_{sdr} and Z_{sdr} capture the total units of SKU s that are delivered to the customer who placed order r and are used in the integer program to ensure all customer demands are satisfied. More specifically, Y_{sdr} is defined as the total units of SKU s that are delivered on day d to the customer who placed order r

and Z_{sdr} is defined as the total units of SKU s that are delivered on or before day d to the customer who placed order r.

When reevaluation is triggered, the required information is captured from the e-tailer's system state to set the value of parameters for the math model. The generic steps that are followed for each reevaluation are outlined below:

Step 1: A snapshot of all orders that need to be reevaluated is captured which includes the number of orders, order items, promised delivery days and customer locations.

Step 2: The list of unique SKUs that are ordered at least by one customer is computed.

Step 3: For those SKUs, a portion of on-hand-unassigned and on-order-unassigned inventory at all FCs is locked and is made available to the reevaluation.

Step 4: The inventory assignments for all orders that are reevaluated are cancelled and the assigned inventory units are made available to the reevaluation.

This provides all necessary information for the reevaluation. Note that if reevaluation is triggered before 1:00 p.m. (i.e. locking time) we consider the day the reevaluation is triggered to be $d = 0$ and the value of d for all other parameters is calculated based on this day. For example, if a reevaluation is triggered at 10:00 a.m. on day 5 of the simulation to reevaluate three customer orders that have a delivery deadline of day 7, 8 and 10 respectively, in constructing the math model, the value of d for $ordQty_{sdr}$ for these orders is calculated as 2, 3 and 5 respectively. If the reevaluation is triggered after 1:00 p.m. on the other hand, the day after the reevaluation is triggered, is considered $d = 0$. In the previous example, if reevaluation is instead triggered at 1:05 p.m. on day 5 of the simulation, the value of d for the three orders is calculated as 1, 2 and 4. The same principle applies to calculating the value of d for inventory. If reevaluation is triggered at

10:00 a.m. on day 5 and FC_1 expects a replenishment for SKU_1 at 12:00 a.m. on day 6, those inventory units are added to $inventory_{111}(d = 1)$. If the reevaluation is triggered at 1:05 p.m. on day 5 on the other hand, those inventory units are added to $inventory_{101}(d = 0)$.

Minimize

$$\sum_{m=1}^{M}\sum_{f=1}^{F}\sum_{r=1}^{R} W_{mfr} \times cPound_{mfr} + \sum_{d=0}^{D-1}\sum_{m=1}^{M}\sum_{f=1}^{F}\sum_{r=1}^{R} B_{dmfr} \times cBox_{mfr} \quad (4\text{-}1)$$

Subject to

$$\sum_{s=1}^{S} X_{sdmfr} \times weight_s \leq maxBoxWeight \times B_{dmfr} \qquad \forall d \in [0, D-1], \forall m, \forall f, \forall r \quad (4\text{-}2)$$

$$W_{mfr} = \sum_{s=1}^{S}\sum_{d=0}^{D-1} X_{sdmfr} \times weight_s \qquad \forall m, \forall f, \forall r \quad (4\text{-}3)$$

$$U_{sdf} = \sum_{m=1}^{M}\sum_{r=1}^{R} X_{sdmfr} \qquad \forall s, \forall d \in [0, D-1], \forall f \quad (4\text{-}4)$$

$$V_{sdf} = \sum_{b=0}^{d} U_{sbf} \qquad \forall s, \forall d \in [0, D-1], \forall f \quad (4\text{-}5)$$

$$V_{sdf} \leq cInventory_{sdf} \qquad \forall s, \forall d \in [0, D-1], \forall f \quad (4\text{-}6)$$

$$Y_{sdr} = \sum_{m=1}^{M}\sum_{f=1}^{F}\sum_{b=0}^{d-1} X_{sbmfr} \times transitDays_{(d-b),m} \qquad \forall s, \forall d \in [1, D], \forall r \quad (4\text{-}7)$$

$$Z_{sdr} = \sum_{b=1}^{d} Y_{sbr} \qquad \forall s, \forall d \in [1, D], \forall r \quad (4\text{-}8)$$

$$Z_{sdr} \geq cQty_{sdr} \qquad \forall s, \forall d \in delivery_r, \forall r \quad (4\text{-}9)$$

In the mathematical model above, the objective function (4-1) minimizes total shipping cost. Constraint (4-2) calculates total number of boxes that are shipped from FC f to customer order r on day d using shipping method m based on assignments, SKU weights, and the maximum box weight. This constraint assumes that a collection of objects weighting less than or equal to $maxBoxWeight * B$ can fit into B boxes without violating the weight limit of any individual box. Although somewhat unrealistic, this constraint allows us to estimate the number of boxes shipped without explicitly specifying the items that are placed in individual boxes. Constraint (4-3) calculates total shipment weight from FC f to customer order r using shipping method m. Constraint (4-4) calculates the total amount of inventory units of SKU s that are shipped out from FC f on day d. Constraint (4-5) calculates the total amount of inventory units of SKU s that are shipped out from FC f on or before day d. Constraint (4-6) ensures the total number of units of SKU s that are shipped from FC f on day d does not exceed available inventory. Constraint (4-7) calculates the total number of units of SKU s that are delivered on day d to the customer who placed order r; this constraint acknowledges the transit time for each shipping method considered. Constraint (4-8) calculates the total number of units of SKU s that are delivered on or before day d to the customer who placed order r. Finally, constraint (4-9) ensures that all items in customer orders are delivered on or before their promised delivery days.

Chapter 5

Heuristic algorithm for reevaluating order fulfillment plans

The large instances of the math model proposed in Chapter 4 are mathematically difficult to solve to optimality. Therefore, a decomposition based heuristic algorithm is presented in this chapter that can quickly solve very large problems and provide a sub-optimal decision that may still be better than the combined decisions generated by the myopic fulfillment decision. It should be noted that several heuristic algorithms were considered including simulated annealing and Tabu search. After an extensive number of tests, the following decomposition-based heuristic demonstrated superior performance to these alternatives.

Our proposed heuristic algorithm reevaluates the fulfillment decisions for a batch of customer orders in four steps. In the first step, customer orders are randomly split into two subsets of equal size. In the second step, the fulfillment decisions for orders in $subset_1$ are fixed based on the original assignments. Third, the sequence of orders in $subset_2$ is randomly shuffled. Finally, a simple integer program is used to optimize the fulfillment decision for each order in $subset_2$ one at a time and sequentially as opposed to optimizing them together.

This heuristic can improve the fulfillment decisions and reduce shipping costs in two primary ways. First, as noted earlier, an integer program applied to a single order outperforms the rule based myopic decision for that order since it considers a more complex set of criteria to mathematically optimize the assignments. Second, shuffling the sequence in which fulfillment decisions are made for individual orders allows the heuristic to reconsider the inventory allocation without being constrained to make those allocations chronologically. In Chapter 7, we illustrate the effectiveness of this algorithm through a set of examples.

Table 5.1 provides a list of indices, parameters and decision variables, and their respective explanations, for the simplified integer program used in the heuristic algorithm. Note the absence of index r in Table 5.1. As shown in this table, by solving the integer program for an individual customer order, the problem complexity reduces significantly. The decomposition reduces number of dimensions for key decision variables in the integer program. Besides, the range of the remaining dimensions particularly index s is also reduced resulting in additional simplification of the problem. For example, number of SKUs in this case reflects the number of items in the specific customer order which is significantly lower than number of SKUs in a collection of customer orders.

Table 5.1: Indices, parameters and decision variables in decomposed integer program

Indices	
s	SKU ($1 \leq s \leq S$)
d	Day ($0 \leq d \leq D$)
m	Shipping method ($1 \leq m \leq M$)
f	FC ($1 \leq f \leq F$)
Parameters	
S	Number of unique SKUs requested in the customer order
F	Number of FCs
M	Number of shipping methods
D	Number of days in reevaluation horizon
$weight_s$	Weight of SKU s in pounds
$maxBoxWeight$	Maximum weight in pounds allowed in a single box, (integer, > 0)
$cBox_{mf}$	Fixed shipping cost for sending a box from FC f to the customer who placed the order using shipping method m
$cPound_{mf}$	Shipping cost per pound from FC f to the customer who placed the order using shipping method m
$ordQty_{sd}$	Number of units of SKU s in the customer order that are requested to be delivered by day d
$cQty_{sd}$	Total (cumulative) number of units of SKU s that must be delivered by day d to the customer who placed the order
$inventory_{sdf}$	Number of units of SKU s that arrive at FC f on day d
$cInventory_{sdf}$	Cumulative number of units of SKU s that arrive at FC f on or before day d
$transitDays_{dm}$ $\begin{cases}1\\0\end{cases}$	If transit time for shipping method m is exactly d days Otherwise
$delivery$	Promised delivery day for the customer order, (integer, $1 \leq delivery \leq D$)
Decision Variables	
X_{sdmf}	Number of units of SKU s that are shipped on day d using shipping method m from FC f to the customer who placed the order, ($0 \leq d \leq D-1$)
B_{dmf}	Number of boxes shipped out of FC f to the customer who placed the order using shipping method m on day d, ($0 \leq d \leq D-1$)
W_{mf}	Total weight of shipment from FC f to the customer who placed the order using shipping method m, ($0 \leq d \leq D-1$)
U_{sdf}	Total units of SKU s that are shipped out of FC f on day d, ($0 \leq d \leq D-1$)
V_{sdf}	Total units of SKU s that are shipped out of FC f on day d or before that, ($0 \leq d \leq D-1$)
Y_{sd}	Total units of SKU s that are delivered on day d to the customer who placed the order, ($1 \leq d \leq D$)
Z_{sd}	Total units of SKU s that are delivered on or before day d to the customer who placed the order, ($1 \leq d \leq D$)

The formulation of this decomposed integer program for a single order is as follows:

Minimize

$$\sum_{m=1}^{M}\sum_{f=1}^{F} W_{mf} \times cPound_{mf} + \sum_{d=0}^{D-1}\sum_{m=1}^{M}\sum_{f=1}^{F} B_{dmf} \times cBox_{mf} \quad (5\text{-}1)$$

Subject to

$$\sum_{s=1}^{S} X_{sdmf} \times weight_s \leq maxBoxWeight \times B_{dmf} \qquad \forall d \in [0, D-1], \forall m, \forall f \quad (5\text{-}2)$$

$$W_{mf} = \sum_{s=1}^{S}\sum_{d=0}^{D-1} X_{sdmf} \times weight_s \qquad \forall m, \forall f \quad (5\text{-}3)$$

$$U_{sdf} = \sum_{m=1}^{M} X_{sdmf} \qquad \forall s, \forall d \in [0, D-1], \forall f \quad (5\text{-}4)$$

$$V_{sdf} = \sum_{b=0}^{d} U_{sbf} \qquad \forall s, \forall d \in [0, D-1], \forall f \quad (5\text{-}5)$$

$$V_{sdf} \leq cInventory_{sdf} \qquad \forall s, \forall d \in [0, D-1], \forall f \quad (5\text{-}6)$$

$$Y_{sd} = \sum_{m=1}^{M}\sum_{f=1}^{F}\sum_{b=0}^{d-1} X_{sbmf} \times transitDays_{(d-b),m} \qquad \forall s, \forall d \in [1, D] \quad (5\text{-}7)$$

$$Z_{sd} = \sum_{b=1}^{d} Y_{sb} \qquad \forall s, \forall d \in [1, D] \quad (5\text{-}8)$$

$$Z_{sd} \geq cQty_{sd} \qquad \forall s, d = delivery \quad (5\text{-}9)$$

In the mathematical model above, the objective function (5-1) minimizes total shipping cost for the order. Constraint (5-2) calculates total number of boxes that are shipped from FC f to the

customer order on day d using shipping method m based on assignments, SKU weights, and the maximum box weight. This constraint has the same limitations as constraint (4-2). Constraint (5-3) calculates total shipment weight from FC f to the customer order using shipping method m. Constraint (5-4) calculates the total amount of inventory units of SKU s that are shipped out from FC f on day d. Constraint (5-5) calculates the total amount of inventory units of SKU s that are shipped out from FC f on or before day d. Constraint (5-6) ensures that the total number of units of SKU s that are shipped from FC f on day d does not exceed available inventory. Constraint (5-7) calculates the total number of units of SKU s that are delivered on day d to the customer who placed the order; this constraint acknowledges the transit time for each shipping method is considered. Constraint (5-8) calculates the total number of units of SKU s that are delivered on or before day d to the customer who placed the order. Finally, constraint (5-9) ensures that all items in customer orders are delivered on or before their promised delivery days.

The structure of the above integer program closely resembles that of the integer program presented in Chapter 4. The main difference is that the subscript r has been removed from the parameters, variables, constraints and objective function. Therefore, this integer program has many fewer variables and constraints than the integer program presented in Chapter 4. Overall, our experiments indicate that the computation time required to reevaluate the fulfillment decisions for n customer orders separately using n instances of the decomposed formulation is significantly lower that the computation time required to reevaluate them together using the original formulation. The details of these experiments are presented in Chapter 7.

Chapter 6

Fully embedded order fulfillment reevaluation algorithm

In Chapter 1 we defined an intense unending real-time operational challenge (IURTOC) and explained why the e-tailer order fulfillment process is an example of such a challenge. Since in an IURTOC, a system's operation never halts and events occur around the clock, when designing a decision-making algorithm (DMA) for it, it is important to not only consider the decisions made by the algorithm, but also its computation time and execution cadence. This is because operations of an IURTOC should not be interrupted while executing a DMA; the system continues to evolve as the algorithm searches for decisions. Since the decisions that are generated by a DMA must be fed back to the real-world system by a certain time to determine the future course of events, it is critical to design a DMA that can meet that timeline. Additionally, if the decision-making process must be repeated regularly in an IURTOC, the cadence of this event must be determined based on the DMA decision quality and computation time.

Because of this complexity, in order to design and evaluate a DMA for an IURTOC, we need a framework that enables us to analyze a DMA from three important dimensions:

1. Decision quality: does the DMA provide good decisions?
2. Computation time: does the DMA provide the decision when it is needed?
3. Execution cadence: how often should the DMA be executed?

While almost all articles in the literature investigate decision quality, the other two dimensions have not been studied extensively. In other words, the vast majority of articles focus on designing a DMA that finds high quality decisions for static problem instances, but they do not explore whether the computation time of that DMA meets the operational requirements of a real-world

IURTOC. Additionally, they do not study how often the DMA should be executed to maximize its effectiveness without causing any interruptions in real-world system operations.

6.1. Definition of a fully embedded decision-making algorithm (FEDMA)

In some studies, computer simulation techniques are used to run experiments with a DMA. However, in those studies only the decisions produced by a DMA are fed back to the simulation model and its computation time is assumed to be zero. We refer to this technique as *partially embedding the DMA within the DES model,* and in this case, we have a *partially embedded DMA* (PEDMA). In this dissertation however, we use the novel technique introduced by Petering (2015, 2018) to embed a DMA in a discrete event simulation (DES) model so that both the decisions produced, and the computation time used by the DMA are fed back to the DES model. In this case we say we have a *fully embedded decision-making algorithm* (FEDMA). Using the FEDMA, we can study different execution cadences for the DMA in order to find the best strategy.

In this chapter we use the FEDMA concept as a framework to embed the integer program that was presented in Chapter 4 and the heuristic method described in Chapter 5 within the DES model that was described in Chapter 3. This allows us to analyze the performance of these decision-making algorithms holistically and from all dimensions that are important to real-world practitioners.

6.2. FEDMA for reevaluating order fulfillment plans

In order to fully embed the integer program from Chapter 4 within the DES model described in Chapter 3, two main events are added to this model. The first event is called *Reevaluate fulfillment decisions.* This event gathers all required information for formulating the integer program from the system state; constructs the mathematical model; and calls the CPLEX solver to solve the integer program. The second event, called *Apply reevaluation decisions,* takes the decisions identified by the solver and feeds them back to the simulation model. Note that there is a time lag between these two events. During this lag other parts of simulation model are not stopped and continue to evolve. For instance, new customer orders are placed and the *Make order fulfillment decision* event is triggered to assign those orders to FCs using the myopic, rule-based fulfillment algorithm. This is one of the key differences between a FEDMA and PEDMA that allows us to analyze impact of DMA computation time on system operations.

In order to identify the best cadence for executing the reevaluation algorithm, we consider two different strategies. In the first strategy, which is illustrated in Figure 6.1, the reevaluation algorithm is triggered for a fixed number of open customer orders. This fixed number is a model parameter that can be adjusted based on the e-tailer's operational characteristics. For instance, if value of this parameter is set to 20, the simulation model waits until there are 20 open customer orders in the queue before executing the reevaluation algorithm. The decision maker might experiment with different values of this parameter to find an optimum value that maximizes cost savings without negatively impacting customer service level. Note that, in this strategy, the elapsed time between two subsequent reevaluations varies. In the second strategy, which is illustrated in Figure 6.2, the reevaluation algorithm is triggered at regular time intervals (e.g. every 30 minutes). In this case, the length of the time interval is a model parameter that can be adjusted. Unlike the

first strategy, the number of customer orders that are reevaluated together is variable. In Chapter 7 we experiment with these strategies to compare their performance under different settings.

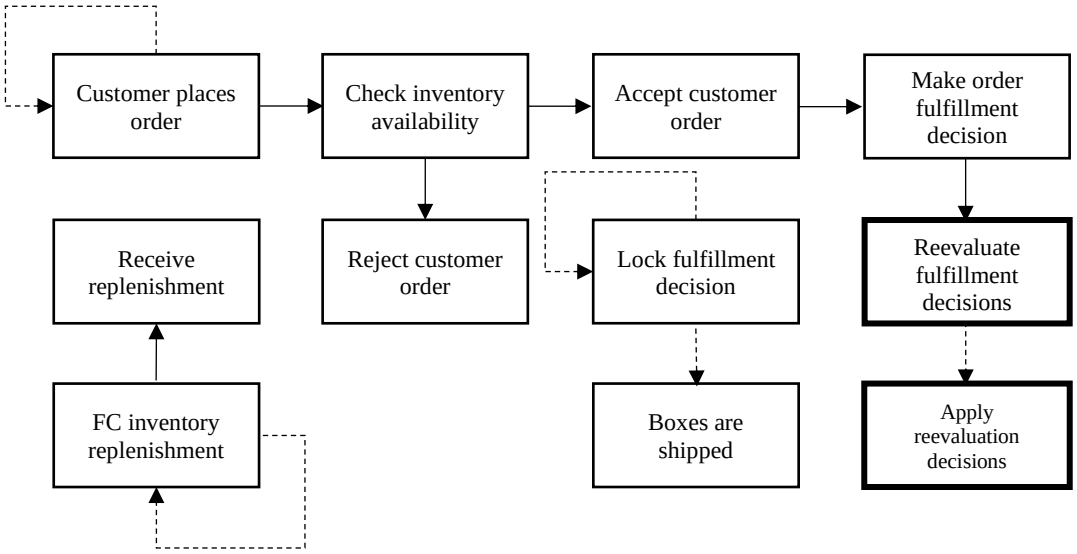

Figure 6.1: Event triggering diagram for DES model with FEDMA - (fixed reevaluation batch size)

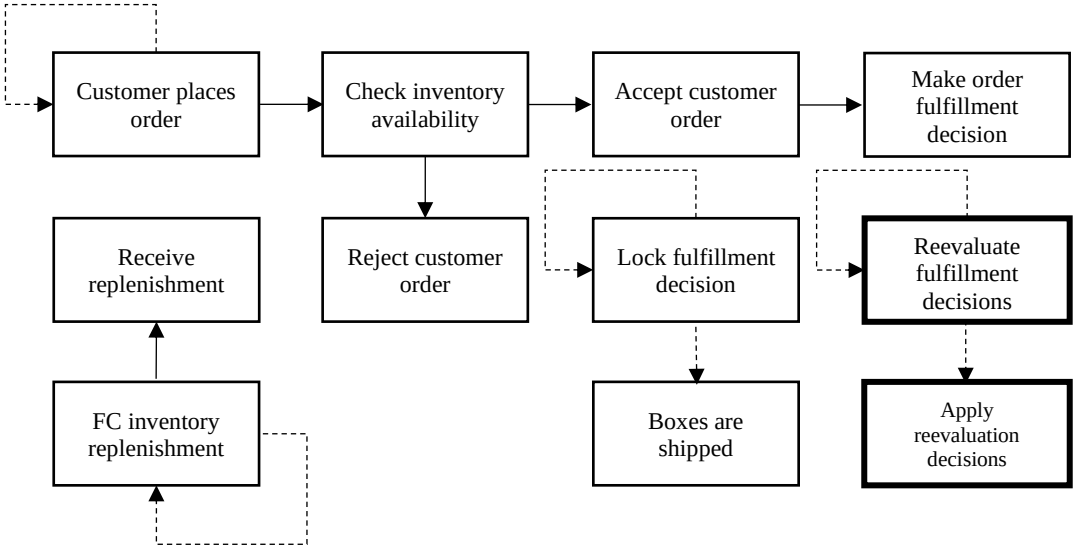

Figure 6.2: Event triggering diagram for DES model with FEDMA - (fixed reevaluation cycle time)

6.3. Challenges when fully embedding the order fulfillment DMA in the DES model

Designing a FEDMA for reevaluating order fulfillment decisions involves various complexities and challenges that are described in this section.

6.3.1. Structural differences between optimization and simulation

Decision making algorithms (e.g. heuristic and integer programming algorithms) assume that the system under study is static. In other words, they assume that the system state does not change between the time the algorithm begins searching for a decision and when that decision is found. Simulation models, on the other hand, assume that the underlying system is dynamic, and its state evolves over time as new events occur. When fully embedding a DMA in a simulation model, the simulation model runs in parallel while the DMA is searching for a decision, so the system state in the DES model will be different when the DMA finds a decision compared to when it had started looking for a decision. The FEDMA needs to take this into account and ensure that the decision it finds is viable given the DES model's new system state when the DMA terminates.

In designing a FEDMA for reevaluating order fulfillment decisions, we can ensure that the integer program results are interpreted based on the updated system state. For instance, as described in Chapter 4, one key decision variable in the integer program is X_{sdmfr}which indicates the number of units of SKU s that are shipped on day d using shipping method m from FC f to satisfy customer order r. When running the reevaluation algorithm, if the algorithm is called on day $\hat{d}$ and finishes on the same day, the values of X_{sdmfr} should be fed back to the simulation model without any adjustments. However, if the algorithm is called on day $\hat{d}$ and ends on day $\hat{d}+1$, then the X_{sdmfr}values must be adjusted to reflect this shift in the d index before they are fed back to the simulation model.

6.3.2. Shared resources between optimization and simulation

When the reevaluation algorithm and simulation model are running in parallel, they use the same set of physical resources and assets such as FCs, SKUs, and inventory units. When the reevaluation algorithm begins, it takes a snapshot of available resources and uses that information throughout its execution. This information cannot be modified while the reevaluation algorithm is running. However, resource availability is impacted by the simulation model which could invalidate decisions produced by the reevaluation algorithm at the end. Therefore, the FEDMA needs to be designed to synchronize resource pooling and prevent any conflicts between the decisions produced by the reevaluation algorithm and simulation. An example of this situation is illustrated in Figure 6.3. In this example, the first customer order is placed at $t = 0$ and the fulfillment decision assigns this order to FC_1 which has adequate inventory to satisfy all order items with a minimal shipping cost. Inventory levels are updated based on this fulfillment decision. A second customer order is placed at $t = 10$ and since no FC can satisfy all items in this order, the rule-based fulfillment algorithm assigns the first two items (SKU_1, SKU_2) to FC_3 and the last item (SKU_3) to FC_2. At $t = 30$ the reevaluation algorithm is triggered to optimize assignments for both orders (O_1, O_2). This reevaluation is expected to find a decision in 10 minutes ($t = 40$). While reevaluation algorithm is being executed, a third customer order is placed at $t = 35$. If the simulation model does not consider the inventory resources that are temporarily allocated to the reevaluation, it may mistakenly allocate inventory units from all FCs (FC_1, FC_2, FC_3) to this order. However, the FEDMA model synchronizes resource pooling by prioritizing resources that are required by the reevaluation algorithm over the new order and decides to reject O_3 due to lack of inventory availability. At $t = 40$ the reevaluation algorithm finds an optimal assignment for O_1 and O_2 and its result is successfully fed back to the simulation model.

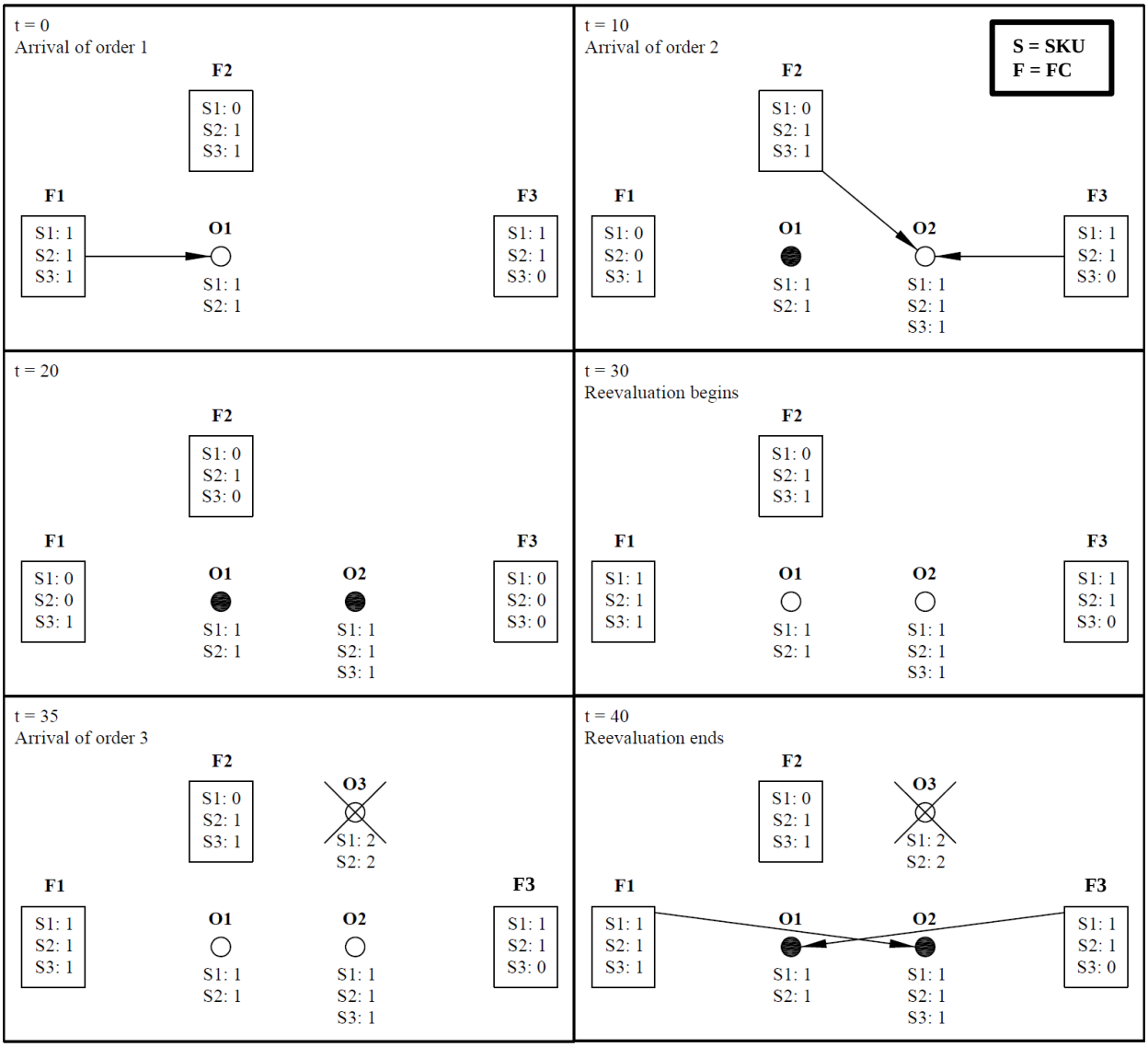

Figure 6.3: Shared resources between reevaluation algorithm and simulation model

6.3.3. Impact of shipment pick-up time

As mentioned earlier, in order to leverage economies of scale, shipment pick-up happens at specific times during a day. In this dissertation, we assume this time is fixed at 2:00 p.m. every day at every FC. Additionally, to prepare customer shipments, all fulfillment decisions must be locked an hour before shipment pick-up time. This means that if reevaluation algorithm is being executed, it must finish before 1:00 p.m.; otherwise its decisions may not be valid. Figure 6.4 illustrates this situation using the previous example. In this case, reevaluation begins at $t = 30$ and is expected to find an optimal decision at $t = 40$. All assignments must be locked by $t = 35$ for preparing customer shipments. Therefore, although the reevaluation algorithm can find better assignments, since it ends after the *Lock fulfillment decision* event, its decisions are nullified and the e-tailer must use the original fulfillment decisions for these customer orders.

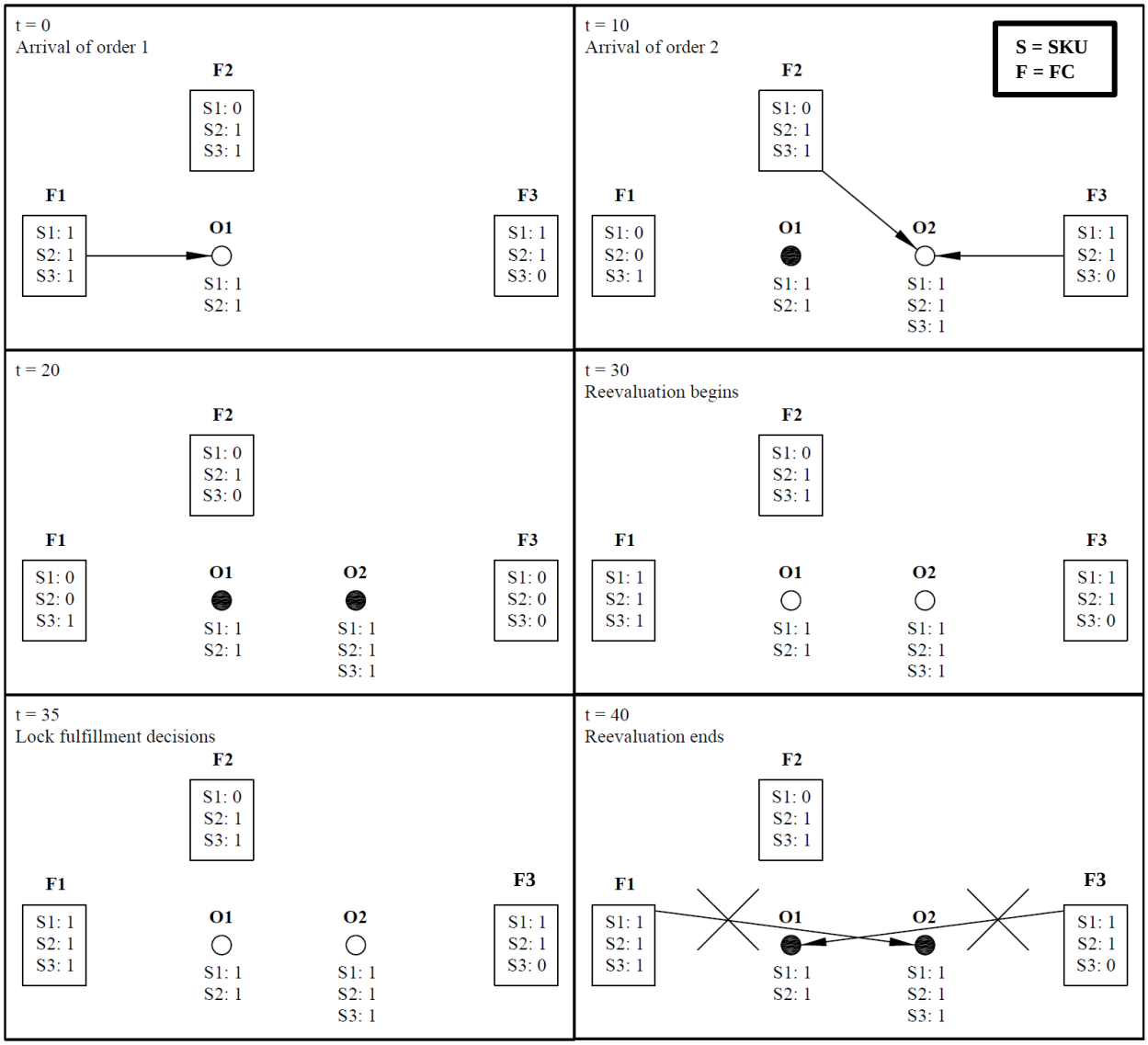

Figure 6.4: Impact of shipment pick-up time on reevaluation algorithm

6.3.4. Impact of inventory replenishment

While the reevaluation algorithm is being executed, FCs might receive inventory replenishments. Considering these replenishments could improve the decisions produced by the reevaluation algorithm. Figure 6.5 illustrates this situation using a simple example. In this example, when the reevaluation algorithm begins at $t = 30$, FC_1 holds one unit of SKU_1 and SKU_2, FC_2 holds one unit of SKU_2 and SKU_3, and FC_3 holds one unit of SKU_1 and SKU_2. On the other hand, O_1 requires one unit of SKU_1 and SKU_2 and O_2 requires one unit of SKU_1, SKU_2 and SKU_3. At $t = 35$, FC_1 receives a replenishment for SKU_3 which increases its inventory level to one unit. Without considering this replenishment, the reevaluation algorithm would assign O_1 to FC_1 and would split O_2 between FC_2 (SKU_3) and FC_3 (SKU_1, SKU_2). Considering this replenishment allows the reevaluation algorithm to find a better decision by assigning O_1 to FC_3 and O_2 to FC_1.

In designing the FEDMA, before reevaluation algorithm begins, the model detects any future replenishments that can be incorporated and makes the inventory in those replenishments available to the reevaluation algorithm. This means that any replenishment that arrives while the reevaluation is executed gets added to the inventory pool. In the next section, we explain how the reevaluation algorithm also considers replenishment orders that are placed after the reevaluation execution is finished for eligible customer orders.

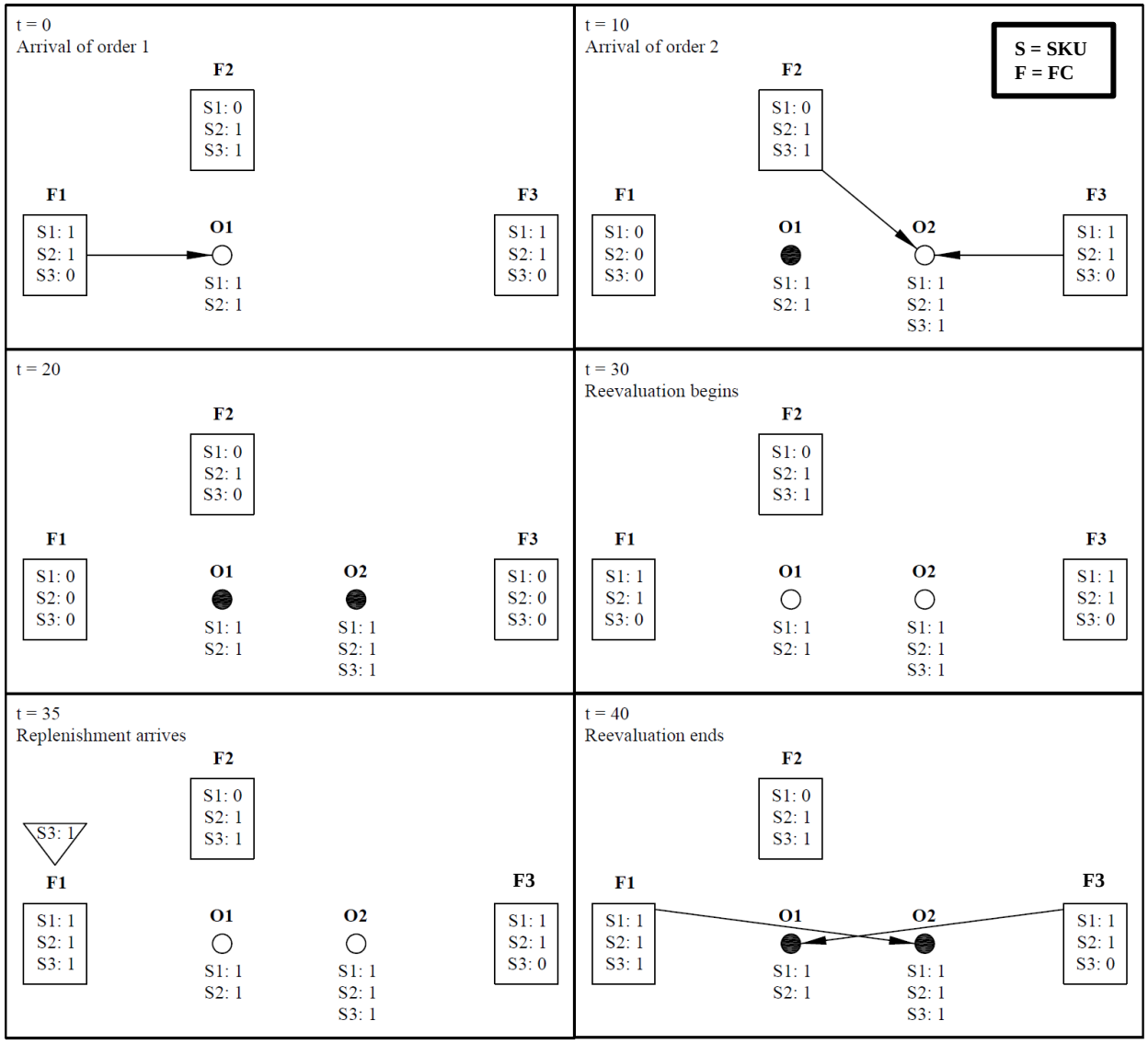

Figure 6.5: Impact of inventory replenishment on reevaluation algorithm

6.3.5. Impact of customer delivery preferences

Customers can choose how long they should wait to receive their online order through delivery preferences. As mentioned earlier, in this dissertation we consider four delivery preferences: *One Day Delivery, Two Day Delivery, Five Day Delivery* and *Seven Day Delivery*. E-tailers use different shipping methods to meet customer delivery preferences. Shipping methods that are considered in this dissertation include *Next Day Air*, *Second Day Air*, *Three Day Select* and *UPS Ground* which has an average transit time of five days. For instance, if a customer chooses *Five Day Delivery*, the e-tailer may use any of these shipping methods to satisfy that order. Since shipping methods with longer transit times are typically cheaper, the e-tailer would normally choose *UPS Ground* in this case. However, there could be a situation where the e-tailer is forced to select a more expensive shipping method due to lack of inventory availability. For instance, if the e-tailer does not have enough inventory to meet this customer order when it is placed, but it is expecting to receive a replenishment in 2 days, it can still meet this order using a *Three Day Select* shipping method.

The reevaluation algorithm must consider this relationship between customer delivery preference, shipping method, and inventory replenishment for all customer orders that are reevaluated together. A simple example is provided in Figure 6.6. In this example, when reevaluation begins at $t = 30$, FC_1 expects a replenishment for SKU_3 that is scheduled to arrive at $t = 75$. This replenishment could be used to assign all items in O_2 to FC_1. However, the reevaluation algorithm must consider the delivery preference for O_2 to determine if by waiting until $t = 75$ the customer delivery deadline can be met. Additionally, if delaying the shipment until $t = 75$ forces the e-tailer to use a more expensive shipping method, that trade-off must be carefully evaluated by the algorithm.

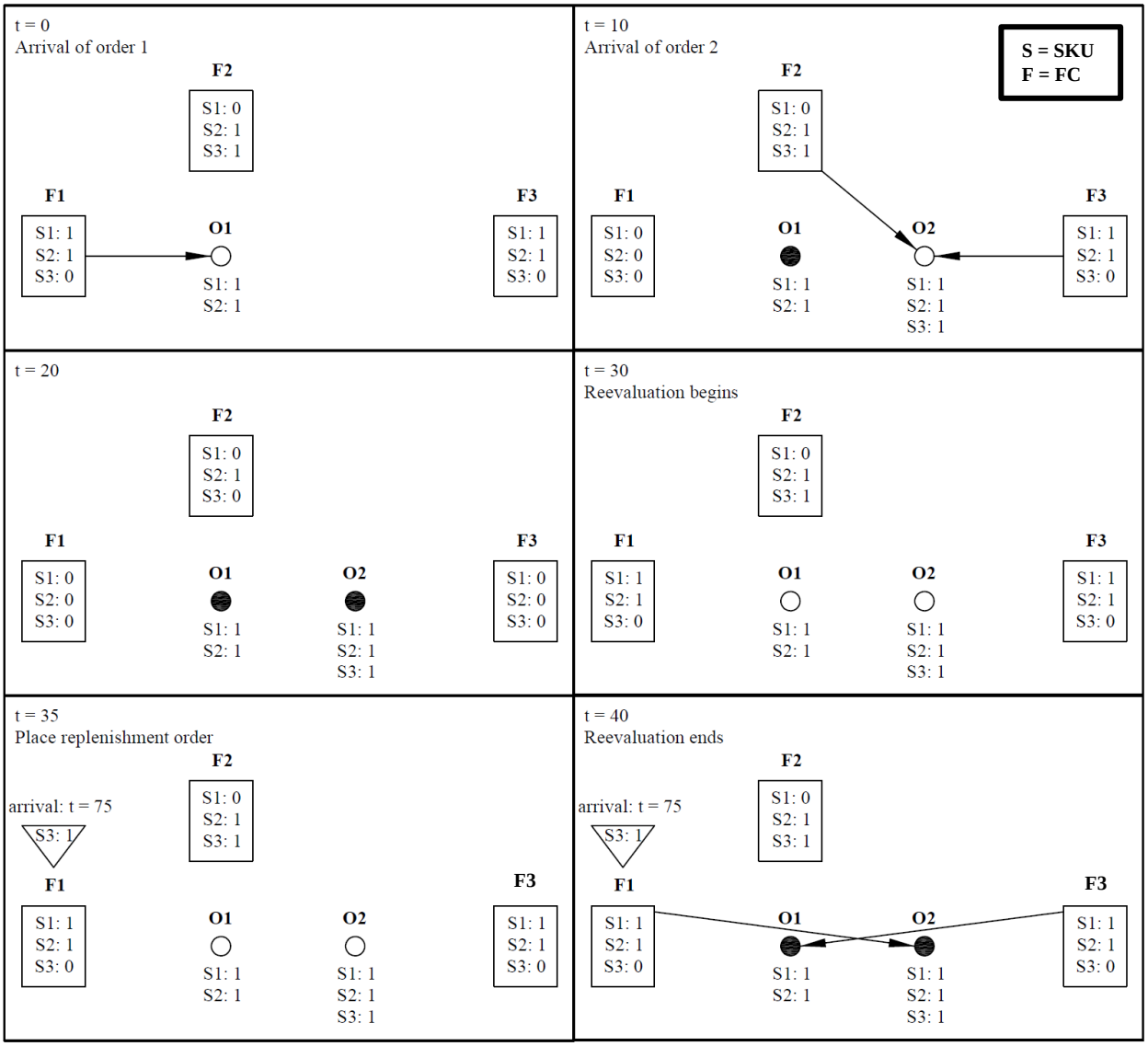

Figure 6.6: Impact of customer delivery preference on reevaluation algorithm

6.3.6. Executing multiple reevaluations in parallel

When embedding a reevaluation algorithm in a simulation model, if we choose to execute reevaluation for a group of n customer orders, there might be a situation where as reevaluation is executed for $\{O_1, O_2, \dots, O_n\}$ a second group of customer orders $\{O_{n+1}, O_{n+2}, \dots, O_{2n}\}$ are placed which triggers another instance of reevaluation algorithm. Figure 6.7 illustrates this situation for batch size of 5. As shown in this figure, while the first instance of the reevaluation algorithm is optimizing assignments for $\{O_1, O_2, O_3, O_4, O_5\}$, customers continue to place new orders. When O_{10} is placed, a second instance of the reevaluation algorithm is triggered to optimize the assignments for $\{O_6, O_7, O_8, O_9, O_{10}\}$. In a real-world e-tailer system, since each instance requires separate infrastructure and a separate optimization agent, the decision maker needs to determine how many reevaluation algorithms can be executed in parallel. In this dissertation, we assume only one reevaluation algorithm can be executed at each time. All customer orders that are placed during the execution of the reevaluation algorithm are added to the next batch. When execution of the reevaluation algorithm ends, the model checks the batch size and if it is greater than or equal to the threshold another instance of the reevaluation is triggered immediately. Otherwise, the model waits until more orders come in before triggering the next instance.

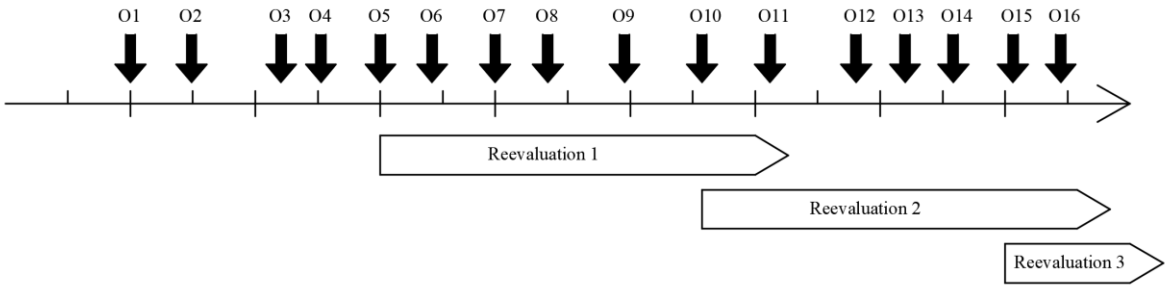

Figure 6.7: Executing multiple reevaluation algorithms in parallel

6.3.7. Locking inventory for reevaluation

When the reevaluation algorithm is triggered, in addition to the inventory units that are already assigned to the orders that are being reevaluated, a portion of unassigned inventory at each FC gets locked and is made available to the reevaluation algorithm. Since inventory is one of the primary constraints for the reevaluation, this enables the algorithm to potentially find a better optimal solution. However, while these inventory units are being used by the reevaluation algorithm, the e-tailer may not use them to fulfill other customer orders that are placed while the reevaluation is being executed. This results in a tradeoff between the quality of decisions produced by the reevaluation algorithm and the quality of the myopic decisions for other orders that are placed during the reevaluation.

When deciding how many inventory units should be reserved for the reevaluation, this tradeoff must be carefully considered so that the e-tailer's overall performance is optimized. Note that, by increasing the proportion of units reserved for reevaluation, the reevaluation decisions are improved, but the myopic decisions are degraded. On the other hand, when the proportion of units reserved for reevaluation decreases, the reevaluation decisions are degraded but the myopic decisions are improved. Therefore, the optimal proportion is a value between 0 and 1 that results in the lowest total shipping cost for all customer orders.

In this simulation, this proportion is a model parameter that can be set by the e-tailer. This allows the decision maker to test different values and find the one that works best for its specific system. Our analysis indicates that, for an e-tailer with limited inventory levels, the proportion should be set to a lower number compared to an e-tailer who has a large amount of inventory. The optimal proportion is also a function of reevaluation computation time. For a shorter reevaluation

computation time, the proportion can be set to a higher value compared to a reevaluation that takes longer to compute.

6.4. Execution cadence for reevaluation

Identifying the best execution cadence for the reevaluation algorithm is an important aspect of fully embedding the DMA within the DES model. Overall, if the reevaluation algorithm is allowed to run until termination, total cost savings will increase as the number of customer orders considered in each call to the algorithm increases. However, increasing the number of customer orders considered in each call to the algorithm also increases computation time which has a negative impact on system performance. Therefore, execution cadence for the reevaluation algorithm needs to be determined by considering this tradeoff between decision quality and computation time. As mentioned earlier, in this dissertation we consider two strategies for triggering the reevaluation algorithm. In the first strategy, the reevaluation algorithm is executed for a group of n open customer orders, where n is a model parameter. In the second strategy, the reevaluation algorithm is executed every t minutes, where t is a model parameter. Both strategies are more effective if the model parameter value is optimized according to the system characteristics.

The importance of execution cadence for the reevaluation algorithm is illustrated through an example shown in Figures 6.8 and 6.9. In this example, we compare the reevaluation decisions when the algorithm is executed for 2 customer orders versus 3 customer orders. The first scenario (Figure 6.8) considers triggering the reevaluation algorithm when there are two open customer orders in the system. In this scenario, when O_1 is placed at $t = 0$, it is assigned to FC_1 which is the closest FC that can satisfy all items in this order in a single shipment. When O_2 is placed at $t = 10$, given the updated inventory levels, its shipment is split between FC_2 for SKU_3 and FC_3 for

SKU_1 and SKU_2. At $t = 20$ the reevaluation algorithm is triggered to optimize the assignment for O_1 and O_2. While the reevaluation algorithm is searching for an optimal assignment, O_3 is placed at $t = 25$. As mentioned earlier, during the reevaluation, in addition to the inventory units that are already assigned to the orders that are reevaluated, 50 percent of the available inventory for each SKU at each FCs (rounded up) gets locked and is made available to the reevaluation algorithm. In this example, since there is only one inventory unit for each SKU at each FCs, all inventory units are locked for reevaluation and O_3 is rejected by the e-tailer. At $t = 30$ reevaluation algorithm execution ends which reduces total number of shipments by 1 by assigning $O_1$to FC_3 and O_2 to FC_1.

The second scenario (Figure 6.9) considers triggering the reevaluation algorithm when there are three open customer orders in the system. This scenario follows a similar process to make the myopic fulfillment decisions for O_1 and O_2. However, the reevaluation algorithm is not triggered before O_3 is placed at $t = 25$ and is assigned to FC_1 for SKU_3 and FC_2 for SKU_2. Instead, it is triggered at $t = 35$ and reevaluates assignments for all three customer orders together. As a result, the number of shipments is reduced from 5 to 3 by assigning O_1 to FC_3, O_2 to FC_1 and O_3 to FC_2.

As shown in this simple example, changing the execution cadence for the reevaluation algorithm not only improves service level by reducing the number of rejected orders, but also promotes cost savings by giving the optimization algorithm more flexibility and degrees of freedom. Although we were able to find the best execution cadence through observation for this example, finding the optimal execution cadence for larger problems is not trivial and requires extensive analysis and experimentation. In the next chapter we present the results of several experiments that relate to execution cadence and other important issues.

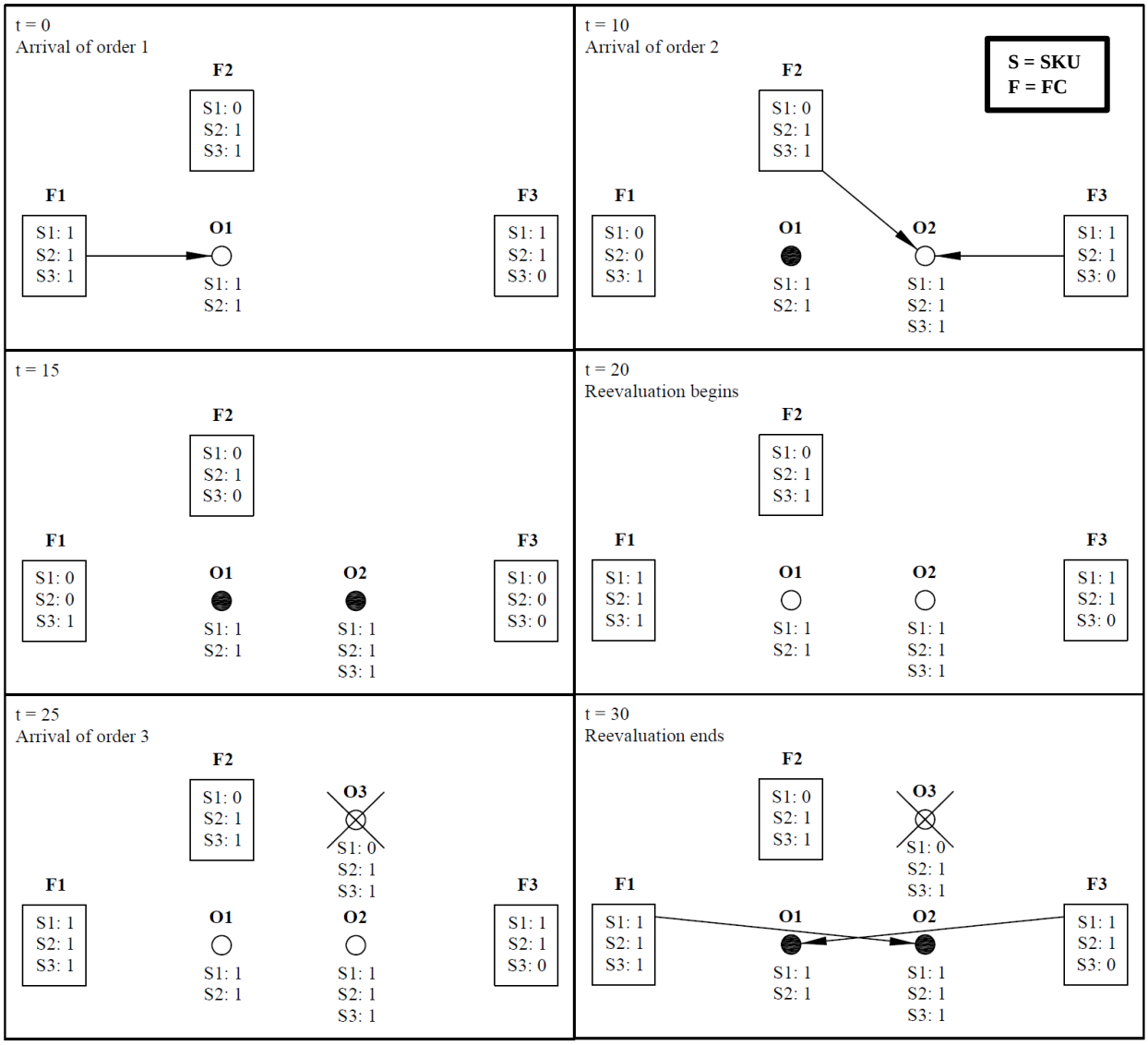

Figure 6.8: Scenario 1: executing reevaluation algorithm for a batch size of 2

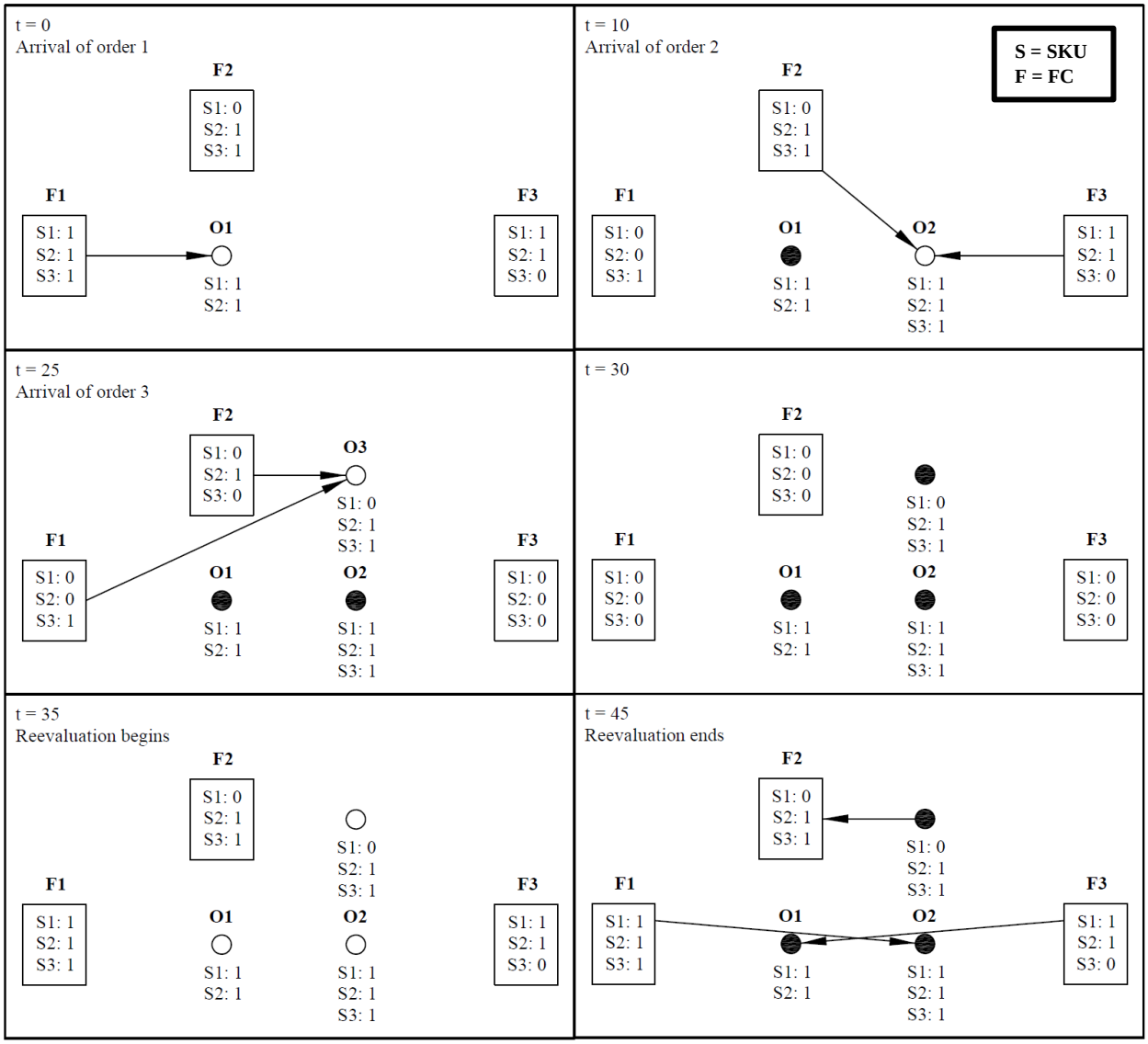

Figure 6.9: Scenario 2: executing reevaluation algorithm for a batch size of 3

Chapter 7

Experimental setup, results, and discussion

In order to analyze the DES model with the fully embedded reevaluation algorithm, an extensive set of experiments is conducted in this section. These experiments are set up to study different aspects of the e-tailer order fulfillment process and to evaluate the impact of various system parameters on the performance of the reevaluation algorithm. The first set of experiments consider the performance of the integer program reevaluation algorithm alone and measure its scalability based on system parameters such as number of customer orders, SKUs, and FCs. The second set of experiments consider the fully embedded reevaluation algorithm within the DES model to demonstrate its effectiveness and to study the impact of the reevaluation algorithm's settings on long-run system performance. All experiments are conducted within Windows 7 environment on a desktop computer with a Core i7 3.4 GHz processor and 16 GB of RAM.

7.1. Experimental setup for IP-based reevaluation algorithm

As described in Chapter 4, the primary decision variable in the integer program reevaluation algorithm is a five-dimensional integer variable x_{sdmfr}. Since in this dissertation we consider a fixed set of shipping methods and customer delivery preferences, the range of the d and m indices are fixed and therefore they do not impact the integer program's scalability. In order to analyze the impact of other three indices, we solve 27 different problem instances. As shown in Table 7.1, these problem instances are defined by the values of S, F, and R. For each of these parameters a low, medium, and high value is considered. This allows us to study the performance of the integer program for a range of problem complexities in all three dimensions. Note that each instance is given a unique ID that is used throughout this chapter to refer to that instance.

Table 7.1: Instances for integer program scalability experiments

Instance ID	# SKUs	# Orders	# FCs
IP_1	5	5	2
IP_2	5	5	10
IP_3	5	5	20
IP_4	5	20	2
IP_5	5	20	10
IP_6	5	20	20
IP_7	5	100	2
IP_8	5	100	10
IP_9	5	100	20
IP_10	20	5	2
IP_11	20	5	10
IP_12	20	5	20
IP_13	20	20	2
IP_14	20	20	10
IP_15	20	20	20
IP_16	20	100	2
IP_17	20	100	10
IP_18	20	100	20
IP_19	100	5	2
IP_20	100	5	10
IP_21	100	5	20
IP_22	100	20	2
IP_23	100	20	10
IP_24	100	20	20
IP_25	100	100	2
IP_26	100	100	10
IP_27	100	100	20

All other parameters are fixed or randomized for these instances. The value of $weight_s$ for each s is set using a uniformly distributed random number between 1 and 10 pounds. Parameter $maxBoxWeight$ is set to a constant value of 18.11 pounds for all instances. The values of parameters $cPound_{mfr}$ and $cBox_{mfr}$, which represent shipping cost per pound and per box respectively, are calculated using the UPS shipping rates that were introduced in Chapter 3. The value of $ordQty_{sdr}$ is calculated using a three-step process which is consistent throughout all instances. In the first step, a truncated exponential distribution is used to determine the number of

order lines for order r; this distribution models the fact that orders with fewer lines are more common. In the second step, another truncated exponential distribution that captures demand variability among SKUs is used to determine the SKU for each order line. Finally, the quantity of each line is calculated using a third truncated exponential distribution. The value of $delivery_r$ for each order is also randomized in a way that cheaper delivery preferences are given a higher weight. Finally, total inventory is determined based on total demand and it is randomly distributed among FCs to set the value of $inventory_{sdf}$.

7.2. Experimental setup for DES model with fully embedded reevaluation algorithm

A second set of problem instances is used to study the behavior of the fully embedded reevaluation algorithm within the DES model. These instances, which are listed in Table 7.2, are defined by three primary parameters: (i) number of SKUs, (ii) number of FCs, and (iii) inter-order-placement time distribution. Like the first set of problem instances, a unique ID is assigned to each instance in Table 7.2 which is used throughout this chapter to refer to that instance.

Table 7.2: Instances for DES model experiments

Instance ID	# SKUs	# FCs	Inter-order-placement time distribution (minutes)
DES_1	1000	10	$E(5)$
DES_2	5	3	$E(5)$
DES_3	10	3	$E(5)$
DES_4	100	3	$E(5)$
DES_5	100	5	$E(5)$
DES_6	100	10	$E(5)$

In addition to the primary parameters that are listed in Table 7.2, several other model parameters are used in this experimentation. The value of most parameters is fixed or randomized for all experiments, while some parameter values are changed to study the sensitivity of the DES

model. The e-tailer's area of operation is assumed to be a rectangular space with a length of 1000 miles and width of 500 miles. All FCs and customer orders are located within this area using a uniformly distributed probability distribution. SKU weights are uniformly distributed between 1 and 3 pounds and the maximum box weight is set to 20 pounds.

As described in Chapter 3, four delivery preferences are considered in this study (i) *One Day Delivery*, (ii) *Two Day Delivery*, (iii) *Five Day Delivery* and (iv) *Seven Day Delivery.* The delivery preference for each customer order is selected using a truncated exponential probability distribution with $\lambda = 0.85$ that gives cheaper delivery preferences a higher likelihood of getting selected. On the other hand, four shipping methods are available to the e-tailer to meet customer delivery deadlines: (i) *Next Day Air*, (ii) *Second Day Air*, (iii) *Three Day Select* and (iv) *UPS Ground*. The number of items in a customer order is a uniformly distributed integer value between 1 and 5. The SKU and quantity for each item are also uniformly distributed. The maximum order quantity for a single item is assumed to be 3.

In order to model the fact that a real-world e-tailer does not hold all SKUs at all FCs, we consider an 80% likelihood that SKU_s is available at FC_f. For each s, f combination a uniformly distributed random value between 0 and 1 is generated; if its value is less than or equal to 0.8, we assume that SKU s is available at FC f. If no FC is selected to hold SKU s through this randomized process, we assume that the last FC in the list holds that SKU.

The inventory policy for each FC and SKU combination is defined using three parameters, $maxLevel$, $reviewCycle$ and $leadTime$. To set the value of these parameters for SKU_s, first the average daily demand for SKU_s is calculated based on the distribution for the inter-order-placement time, number of order lines and quantity of each item ordered. Then that demand rate

is equally distributed between FCs that hold SKU_s. The parameter $reviewCycle$ and $leadTime$ are randomly set to either 1, 2 or 3 days and $maxLevel$ is derived by multiplying the demand rate that is assigned to FC f and the $lead\_time$. Additionally, a 10% safety stock is added to $maxLevel$ to absorb demand variability. Inventory replenishments are assumed to reach FCs one second after midnight every day. Customer shipments are picked-up from FCs at 2:00 p.m. every day and all assignments for those shipments are locked one hour before that event.

In all experiments, 4000 customer orders are simulated which represents approximately two weeks of e-tailer operations. In order to make statistical inference, each experiment is replicated 6 times with a different seed for the random number generator. The same 6 random number seeds are used across experiments to ensure cross-evaluations are accurate.

The reevaluation algorithm can either be triggered for a fixed batch size of customer orders or a fixed cycle time. The batch size and cycle time are both model parameters. The value of these parameters is adjusted throughout the experiments to analyze model sensitivity and to find their optimal value based on the e-tailer's operational characteristics.

The reevaluation computation time is controlled using two parameters, $timePerOrder$ and $adjustmentFactor$. The first parameter represents the actual time that is given to the reevaluation algorithm per order that is reevaluated. The second parameter models the fact that in a real-world e-tailer system, reevaluation computation time could be different. Both parameters are studied extensively in the following experiments.

For example, if the $timePerOrder$ is set to 1 minute, and the $adjustmentFactor$ to 10, when the reevaluation algorithm is executed for a batch of 50 customer orders, the computation time limit for the reevaluation is set to 50 minutes; however, in the simulation model, we assume that

this reevaluation takes 500 minutes. On the other hand, if $adjustmentFactor$ is set to 0.1, then in the simulation we assume the reevaluation algorithm only takes 5 minutes. Therefore, when $adjustmentFactor$ is set to a value less than 1, it allows us to model the fact that an e-tailer may have access to a more powerful CPU for executing the reevaluation algorithm than our test environment. When it is set to a value higher than 1 on the other hand, it allows us to speed up our experimentation by reducing the amount of the experiment's time that is spent on each instance of the reevaluation execution. Finally, by setting $adjustmentFactor$ to 1, we can model a scenario where the e-tailer's reevaluation computation time is equal to the actual computation time used in our experiments.

7.3. Results and discussion

7.3.1. Integer programming reevaluation algorithm scalability

In this experiment, the scalability of the integer programming reevaluation algorithm is studied by solving problem instances *IP_1* to *IP_27* using CPLEX. A fixed time limit of 10 minutes is imposed for all instances and the solution status, objective function, and gap percentage (percentage difference between the objective value of the best solution found and a lower bound on the optimal value) are reported at the end. The results of this experiment are reported in Table 7.3. As shown in this table, optimal solutions are found for 16 instances in less than 10 minutes. For the remaining 11 instances, a feasible solution is reached with an average gap percentage of 4%.

In order to analyze the relationship between problem difficulty and the number of SKUs, number of customer orders, and number of FCs, a scatter plot of the optimization gap percentage based on value of each parameter is constructed in Figure 7.1. It can be observed that the number of customer orders has a non-linear positive impact, number of FCs has a linear positive impact

and number of SKUs has a non-linear negative impact on problem difficulty. The positive impact of the number of customer orders and FCs on problem difficulty can be explained by the increasing number of decision variables and constraints in the integer program. The negative impact of the number of SKUs, on the other hand, can be attributed to the fact that, for a fixed number of customer orders, increasing the number of SKUs reduces the amount of overlap between orders which subsequently reduces the number of ways that orders can be reassigned to different FCs.

Table 7.3: Results from integer program experiments

Problem instance				Solution			
Instance ID	# SKUs	# Orders	# FCs	Solution Status	Objective Function ($)	Gap (%)	Elapsed Time (Sec)
IP_1	5	5	2	Optimal	354	0	< 1
IP_2	5	5	10	Optimal	425	0	< 1
IP_3	5	5	20	Optimal	402	0	< 1
IP_4	5	20	2	Feasible	768	0.54%	600
IP_5	5	20	10	Feasible	682	1.22%	600
IP_6	5	20	20	Optimal	808	0	91
IP_7	5	100	2	Feasible	2191	2.52%	600
IP_8	5	100	10	Feasible	3451	4.10%	600
IP_9	5	100	20	Feasible	2959	9.96%	600
IP_10	20	5	2	Optimal	498	0	< 1
IP_11	20	5	10	Optimal	542	0	< 1
IP_12	20	5	20	Optimal	361	0	< 1
IP_13	20	20	2	Optimal	714	0	21
IP_14	20	20	10	Optimal	1373	0	18
IP_15	20	20	20	Optimal	820	0	13
IP_16	20	100	2	Feasible	3692	3.32%	600
IP_17	20	100	10	Feasible	3250	7.08%	600
IP_18	20	100	20	Feasible	4112	7.55%	600
IP_19	100	5	2	Optimal	358	0	< 1
IP_20	100	5	10	Optimal	431	0	< 1
IP_21	100	5	20	Optimal	158	0	< 1
IP_22	100	20	2	Optimal	1409	0	253
IP_23	100	20	10	Optimal	1276	0	3
IP_24	100	20	20	Optimal	922	0	2
IP_25	100	100	2	Feasible	5215	2.65%	600
IP_26	100	100	10	Feasible	6521	1.43%	600
IP_27	100	100	20	Feasible	5176	1.70%	600

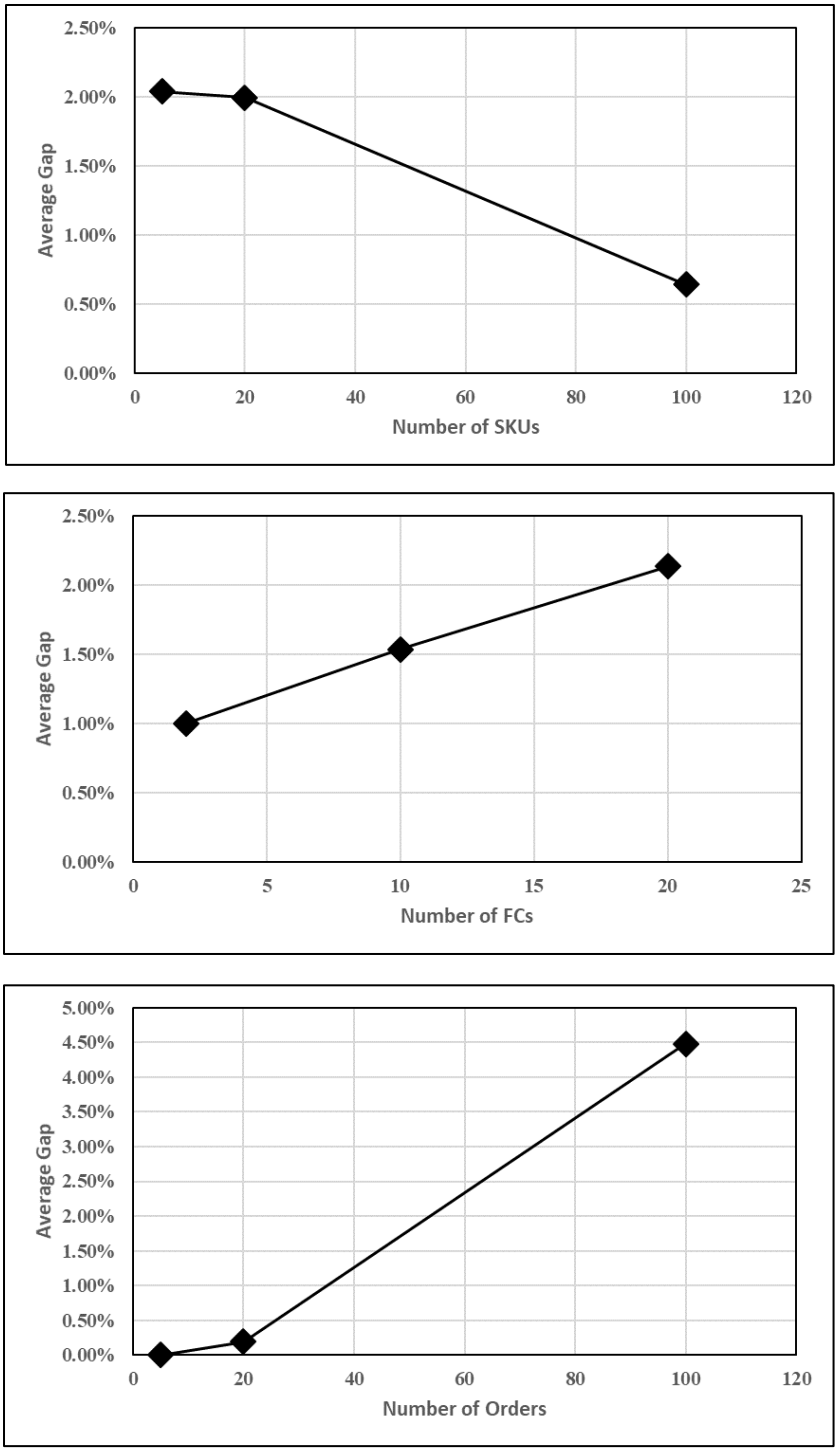

Figure 7.1: Relationship between number of SKUs, FCs and orders and optimization gap

7.3.2. Simulation model performance without running reevaluation algorithm

In order to study the simulation model and develop a baseline for the e-tailer's performance without the reevaluation algorithm, six replications of the simulation model were executed on instance *DES_1*. The result of this experiment is quantified using average shipping cost per order and service level which is illustrated in Figure 7.2. As shown in this figure, the average shipping cost per order is consistent across all replications and its mean value is $22.75. Additionally, service level, which is calculated as the percentage of customer orders that are accepted by the e-tailer, is similar for all replications and its mean value is 88%. In the following experiments the impact of the reevaluation algorithm in these KPIs is analyzed.

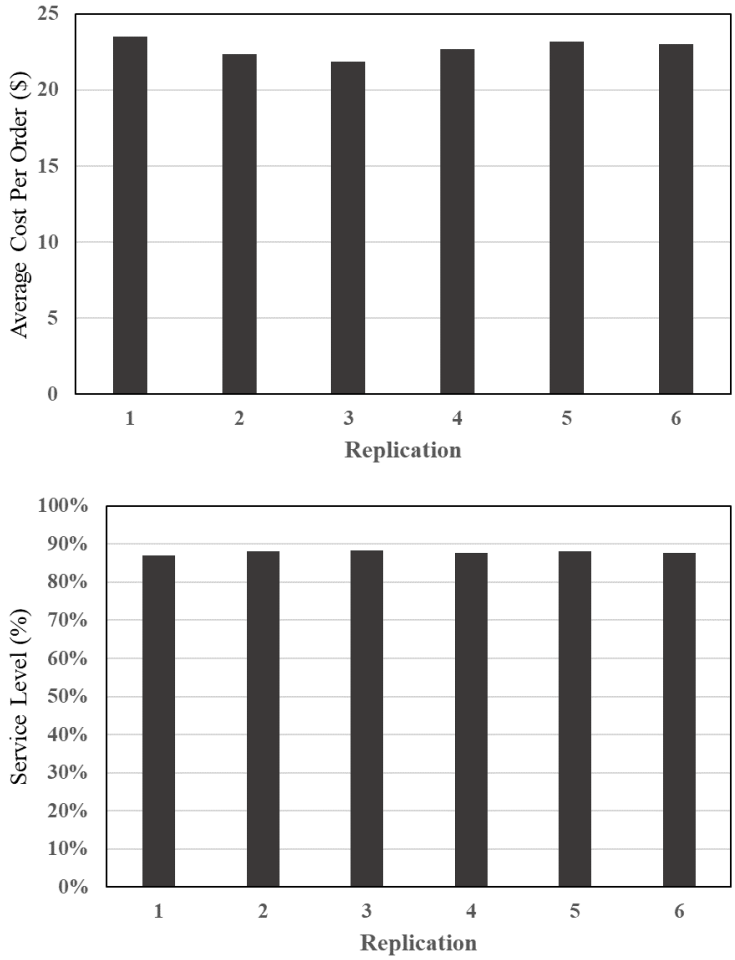

Figure 7.2: Simulation model performance without reevaluation

7.3.3. Reevaluation algorithm performance for individual customer orders

As described in Chapter 3, the simulation model utilizes a rule-based method to make fulfillment decisions for customer orders one at a time. Since the reevaluation algorithm from Chapter 4 is an integer program that uses mathematical optimization, it outperforms the rule-based method even when it is applied to one customer order at a time. To quantify the difference in performance between these two methods, two sets of experiments are conducted on instance *DES_1*. The first set is identical to the experiments that are outlined in Section 7.3.2 where the simulation model is executed without considering reevaluation. For the second set of experiments, the reevaluation algorithm is triggered for a batch size of 1. The $timePerOrder$ parameter is set to 1 second and $adjustmentFactor$ is set to 10.

As displayed in Figure 7.3, running the reevaluation algorithm for individual orders reduces average shipping cost per order from $22.75 to $22.23 which amounts to a savings of approximately 2.3%. Additionally, the service level is not negatively impacted by reevaluation, and its mean value remains at 88%.

In this experiment, the inter-order-placement time is significantly higher than the allocated time for each reevaluation. In particular, customer orders are placed every 5 minutes and it takes the reevaluation algorithm only 10 seconds ($timePerOrder\ X\ adjustmentfactor$) to find an optimal fulfillment decision for them. Therefore the e-tailer can replace the rule-based method with the integer program reevaluation algorithm and save 2.3% in shipping costs. However, for an e-tailer with a shorter inter-order-placement time and more FCs, less computation time will be available and each problem instance would be more difficult to solve, so it may not be practical to replace the rule-based method with an integer programming approach. Additionally, reevaluating

fulfillment decisions for a group of orders enables an e-tailer to shuffle the assignments holistically and minimize the overall shipping cost for that group. Therefore, in the next experiment we study the strategy of reevaluating fulfillment decisions for a batch of customer orders.

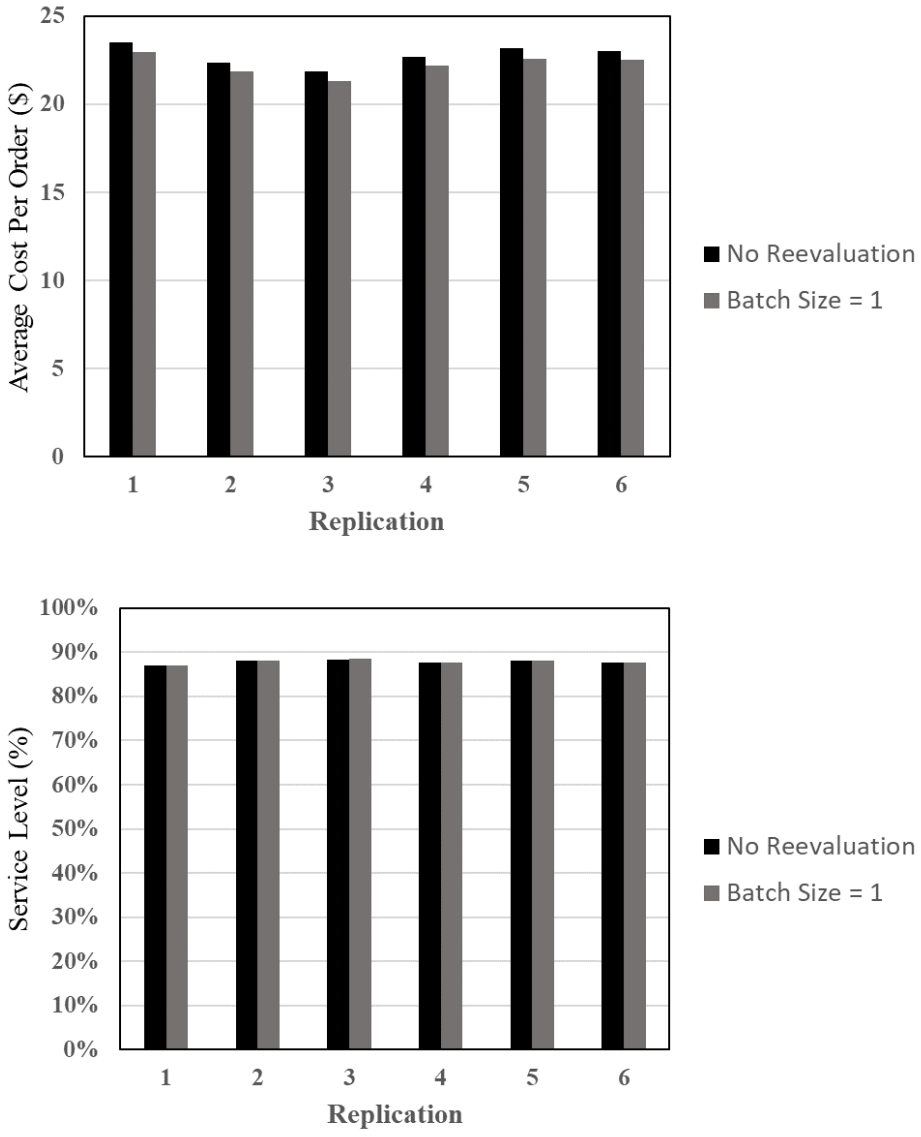

Figure 7.3: System performance for reevaluating orders one at a time

7.3.4. Triggering reevaluation for a batch of customer orders

In order to analyze the value of reevaluating fulfillment decisions for a group of customer orders together, a set of experiments are executed using *DES_1* and a reevaluation batch size of 20. The result of this experiment in comparison with the baseline (no reevaluation) as well as a reevaluation batch size of 1 is illustrated in Figure 7.4. Note that $timePerOrder$ and $adjustmentFactor$ parameters are set to 1 second and 10 respectively. As shown in this figure, setting the reevaluation batch size to 20 reduces average shipping cost per order in all replications resulting in mean value of $22.11 which is $0.12 less than batch size of 1. Additionally, this strategy does not influence service level. However, although for a batch size of 1 all customer orders could be reevaluated for this instance, by increasing the batch size to 20, about 2.3% of customer orders are not reevaluated (Figure 7.5). Those are the orders with a tight customer delivery deadline.

The result of this experiment confirms that reevaluating a batch of customer orders reduces average shipping cost. In the next experiment we find the optimal reevaluation batch size for the instance *DES_1* and develop a framework that can be replicated to find the optimal value for e-tailers with various operational characteristics.

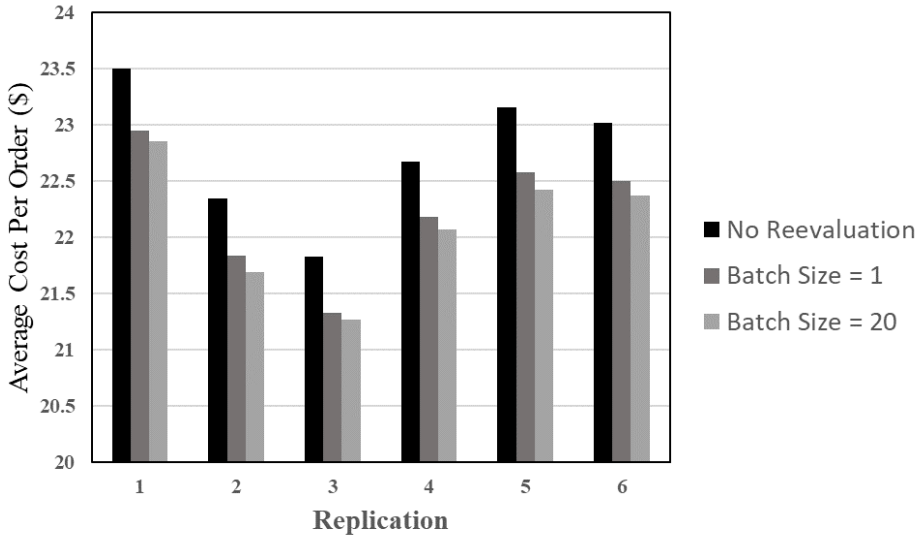

Figure 7.4: Impact of reevaluating a batch of orders on average shipping cost

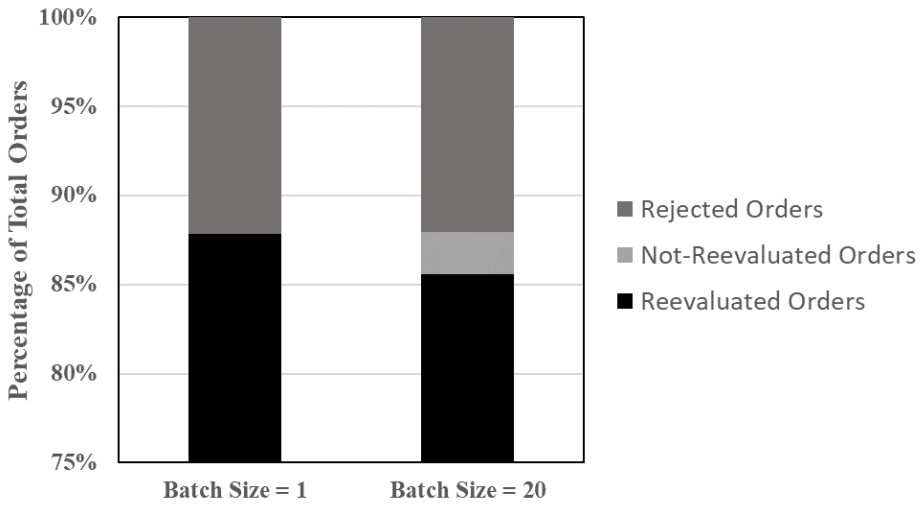

Figure 7.5: Impact of reevaluating a batch of orders on number of orders reevaluated and service level

7.3.5. Identifying the optimal batch size for reevaluation

In the previous experiment, we illustrated the value of reevaluating fulfillment decisions for a group of customer orders. Although increasing the batch size enables the reevaluation algorithm to make better reassignments that result in more cost reduction, it also increases problem complexity and computation time that have a potential negative impact on system performance. Therefore, by using a very large batch size, the overall system performance might be degraded which subsequently impacts average shipping cost. In other words, there must be an optimal value for the batch size that best trades off the decision quality and computation time of the reevaluation algorithm.

To find the optimal batch size for instance *DES_1*, a set of experiments are conducted with different values for this parameter. Note that values of all other model parameters including $timePerOrder$ and $adjustmentFactor$ are fixed. Figure 7.6 shows the result of this experiment. According to the results, a batch size of 50 provides the best outcome by reducing the average cost per order to $21.79 and maintaining an 88% service level.

As shown in Figure 7.6, by increasing the batch size from 50 to a larger number, the reevaluation algorithm is not able to find an optimal decision for all customer orders and the optimization gap percentage grows. Additionally, the number of customer orders that are not reevaluated increases. This is because the customer orders that have a tight delivery deadline may need to be shipped before a batch of 50 customer orders accumulates in the system to trigger the reevaluation. The combination of these two phenomena results in the system performance degradation which increases the average cost per order.

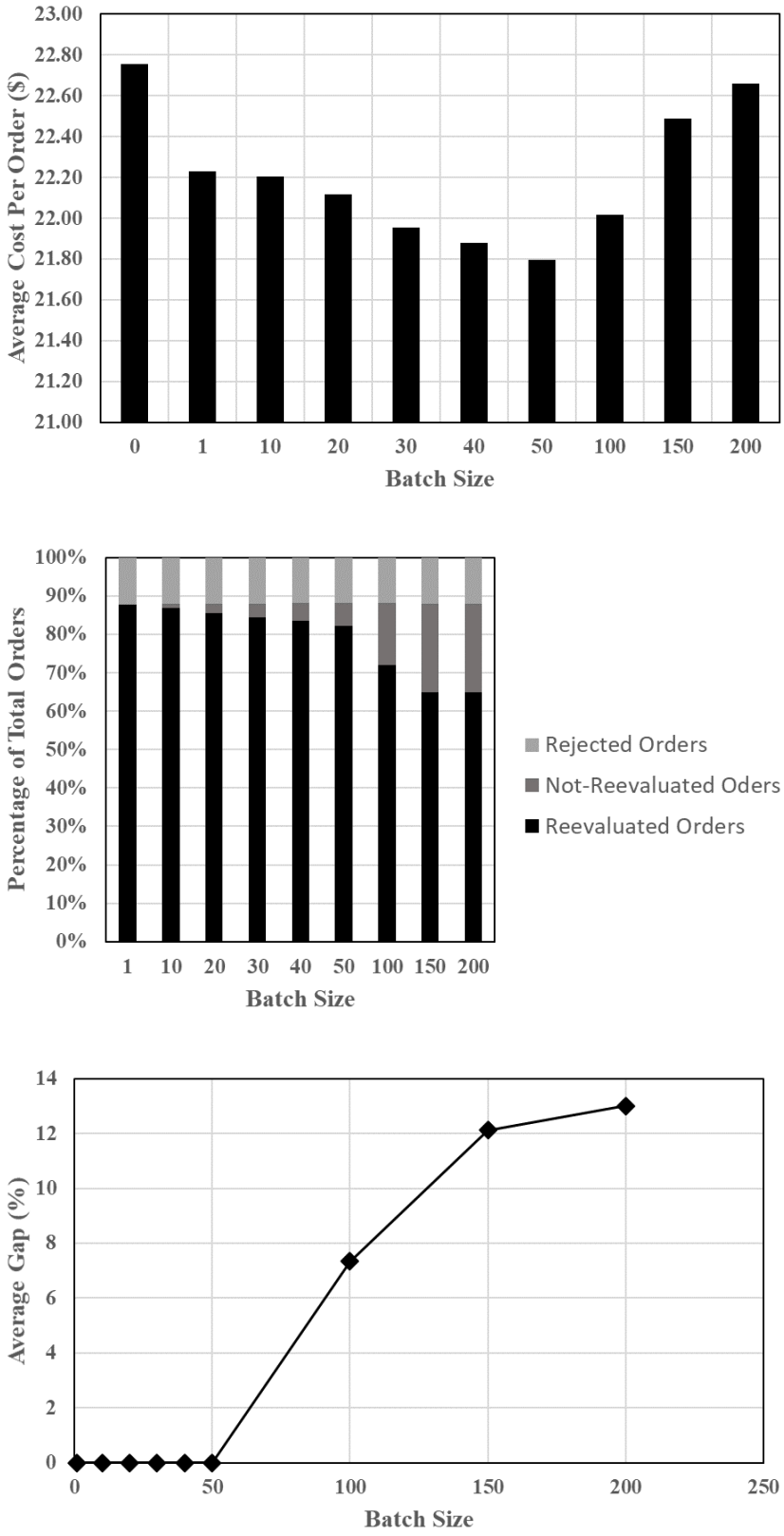

Figure 7.6: Identifying the optimal batch size for reevaluating orders

7.3.6. Impact of reevaluation time per order

The $timePerOrder$ parameter is one of the key model parameters that specifies the computation time limit for the reevaluation algorithm. In order to study the impact of this parameter on system performance a set of experiments are conducted using instance *DES_1*. As shown in the previous section, by setting the reevaluation batch size to 100 and $timePerOrder$ to 1 second, the reevaluation algorithm is not able to find an optimal decision for all customer orders. In this experiment, we test four different values for $timePerOrder$ and analyze the results. The values that are considered are 1, 2, 5 and 10 seconds. Note that $adjustmentFactor$ and the batch size are set to 10 and 100 respectively.

Figure 7.7 illustrates the result of this experiment. This result shows that by increasing the value of $timePerOrder$ from 1 second to 2 seconds, system performance is improved and the average cost per order decreases from \$22.02 to \$21.69. However, when $timePerOrder$ is further increased to 5 and 10 seconds, average cost per order increases slightly.

This behavior can be explained by observing the gap percentage and percentage of orders that are reevaluated in Figure 7.7. As shown in this figure, by increasing $timePerOrder$ from 1 second to 2 seconds, the optimization gap percentage is significantly reduced while the percentage of orders that are reevaluated remains intact. This results in a major improvement in average cost per order. However, increasing $timePerOrder$ to 5 and 10 seconds does not significantly reduce the gap percentage and on the other hand decreases the percentage of orders that are reevaluated. The combined effect of these two events results in a slight degradation in the system performance.

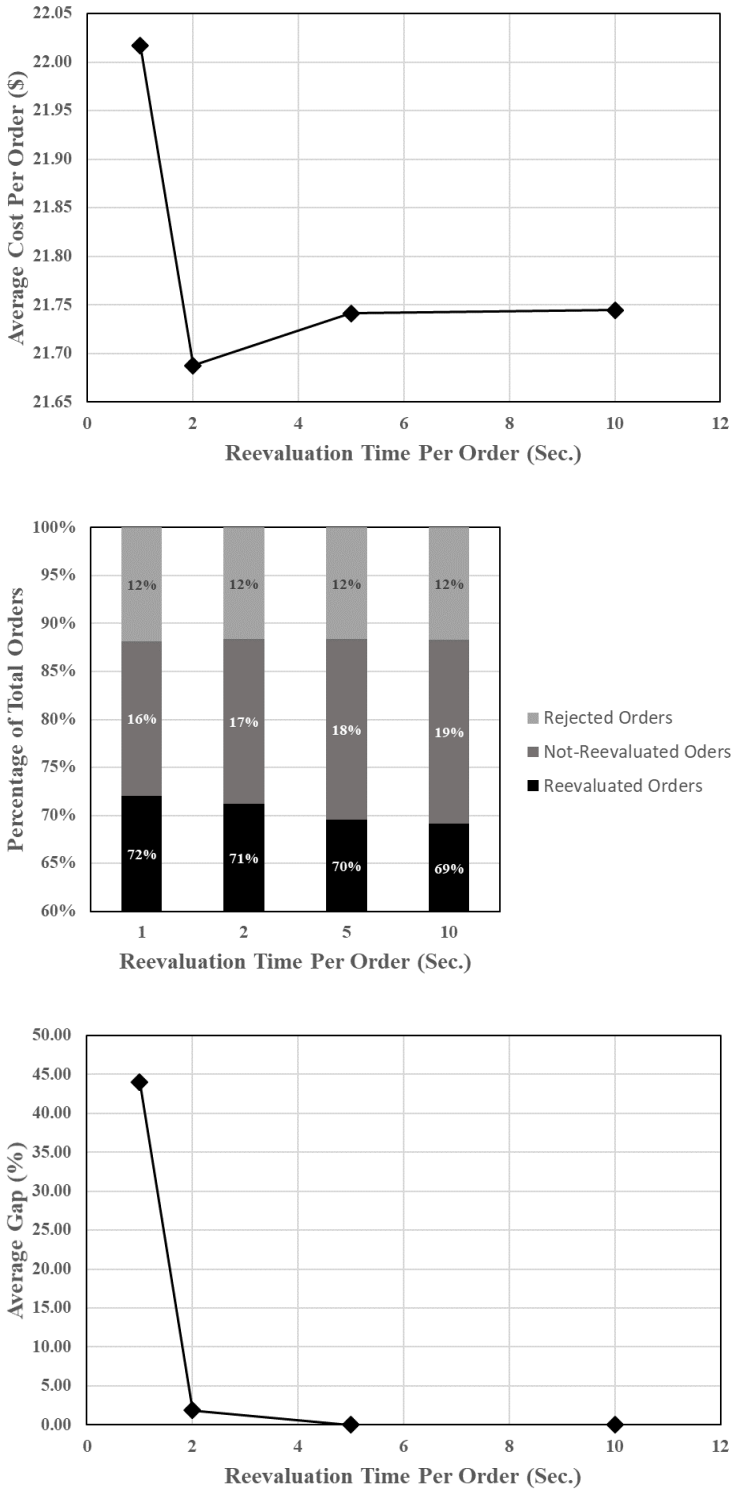

Figure 7.7: Impact of reevaluation time per order on system performance

7.3.7. Impact of adjustment factor

Another important parameter in this simulation is $adjustmentFactor$. As described earlier in this chapter, this parameter models the fact that in a real-world e-tailer system, reevaluation computation time might be different from what is considered in the experiments. We test the impact of this parameter through a set of experiments in this section.

For this experiment, four different values of $adjustmentFactor$ parameter are tested for the instance *DES_1*. These values range from 1 to 200. The reevaluation batch size is set to 50, $timePerOrder$ to 1 second, and other model parameters are fixed as in the previous experiments. The results are summarized in Figure 7.8, which illustrates that increasing the adjustment factor negatively impacts system performance and increases average cost per order.

Note that since the value of $timePerOrder$ parameter is fixed, the optimization gap percentage does not change by increasing the value of $adjustmentFactor$. However, as shown in Figure 7.8, the number of orders that are reevaluated tend to be lower for larger values of $adjustmentFactor$. This is because increasing the value of $adjustmentFactor$ increases the computation time of each reevaluation algorithm run within the simulation model. Subsequently, the e-tailer may not get the opportunity to reevaluate customer orders with a tight delivery deadline. This negatively impacts system performance and results in a higher average cost per order.

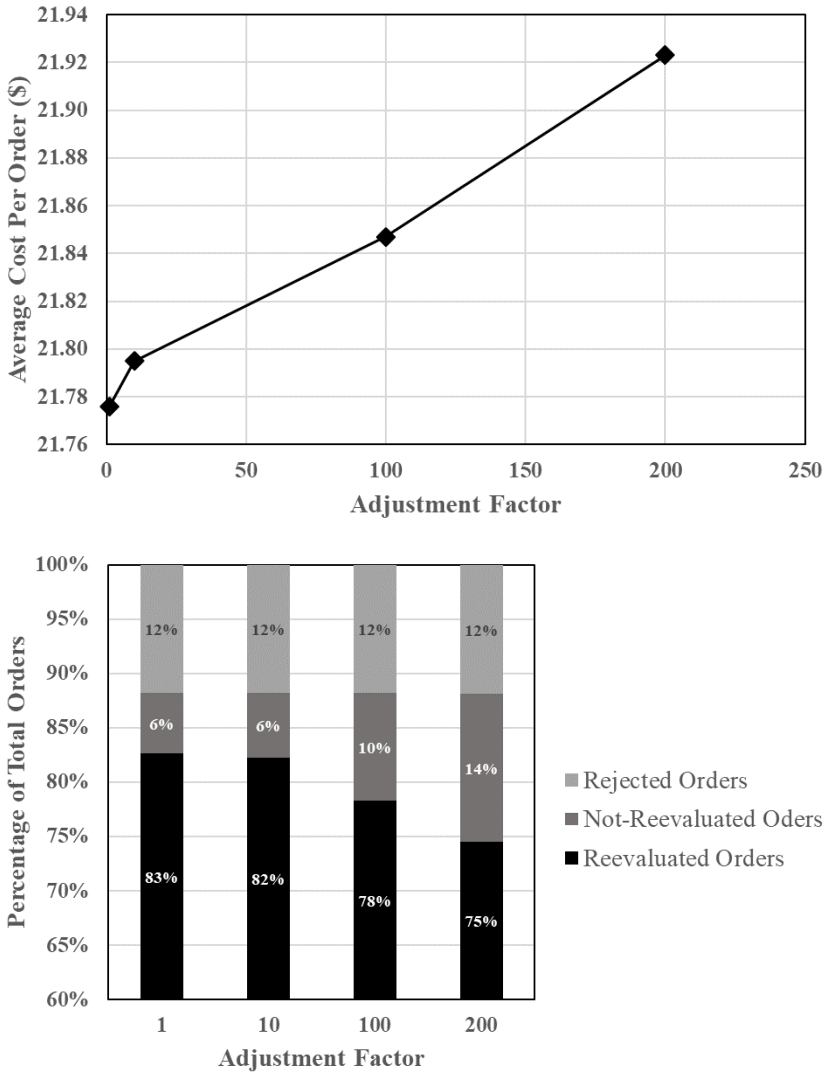

Figure 7.8: Impact of adjustment factor on system performance

7.3.8. Triggering reevaluation in fixed time intervals

Instead of triggering the reevaluation for a predetermined number of customer orders, the e-tailer may choose to execute the reevaluation in fixed time intervals. In this section we study this strategy by running a set of experiments on instance *DES_1*. For these experiments, the value of $timePerOrder$ and $adjustmentFactor$ parameters are set to 1 second and 10 respectively and the simulation is executed with different reevaluation cycle times. Results are presented in Figure 7.9.

As shown in this figure, the average cost per order demonstrates a very similar pattern to batch size execution results. Note that since the inter-order-placement time follows an exponential distribution with a mean of 5 minutes, a cycle time of 250 minutes is approximately equivalent to a batch size of 50. Therefore, according to this experiment, both strategies provide a very similar outcome for instance *DES_1*. In other words, the e-tailer can minimize shipping costs by choosing to trigger the reevaluation algorithm either every 250 minutes or each time 50 customer orders accumulate.

In this experiment, both strategies provide a very similar outcome, but this may not be necessarily the case for all e-tailers. For example, if the inter-order-placement time does not follow an exponential distribution, a batch size of 50 might not be comparable to a 250-minute cycle time. Therefore, our recommendation is to test both strategies based on the e-tailer's operational characteristics to identify the best option that provides the minimum shipping cost.

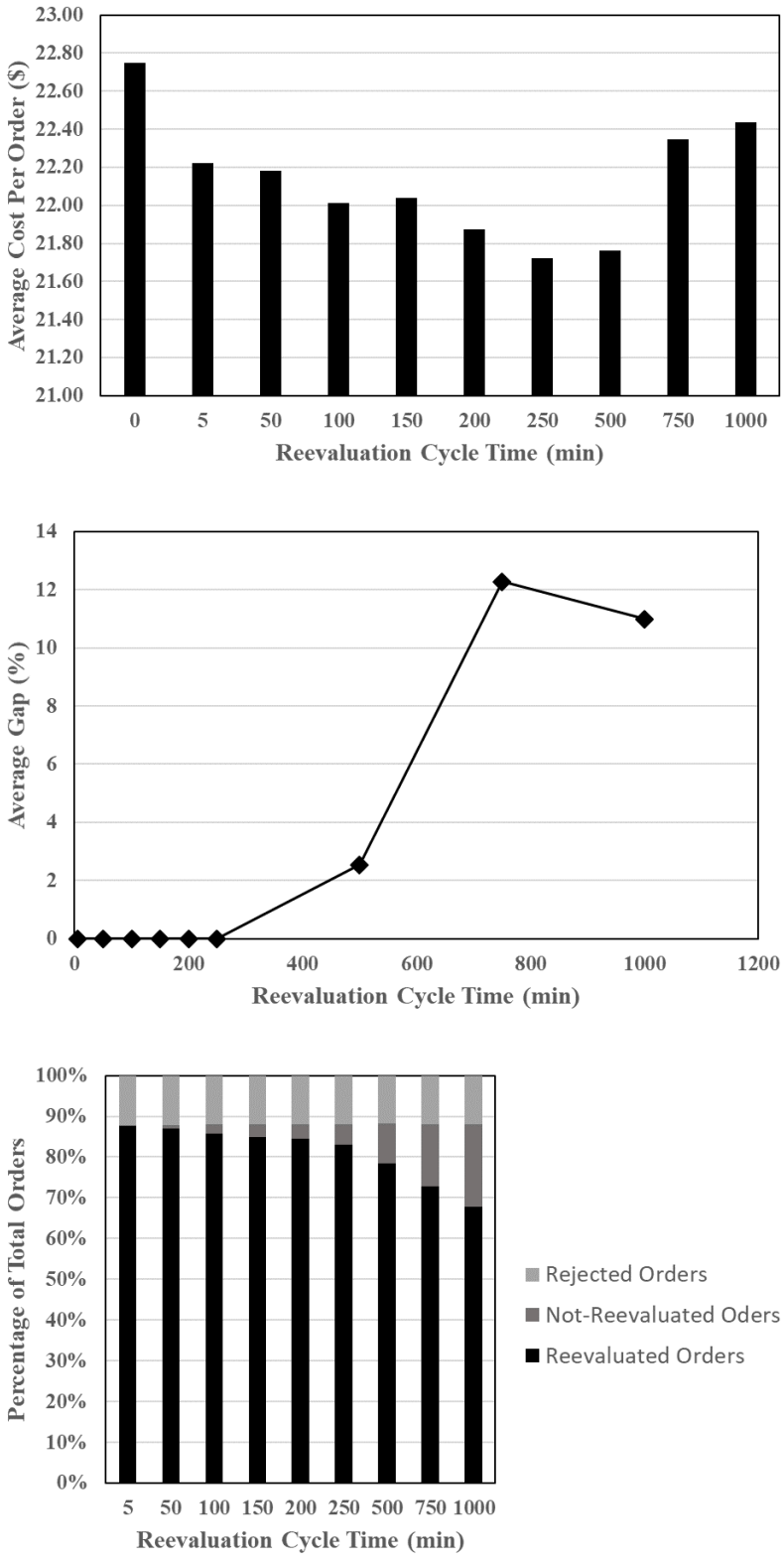

Figure 7.9: Triggering reevaluation in fixed time intervals

7.3.9. Impact of number of FCs

In Section 7.3.1 we studied the impact of the number of FCs on the performance of the IP-based reevaluation algorithm. However, we did not consider the simulation model in that experiment; the reevaluation algorithm was analyzed in isolation. This section extends that analysis by considering the impact of the number of FCs when the reevaluation algorithm is fully embedded within the DES model. For this analysis, a set of experiments are conducted on instances *DES_4*, *DES_5* and *DES_6*. These instances have a very similar configuration except for the number of FCs. There are 3 FCs in *DES_4,* 5 FCs in *DES_5* and 10 FCs in *DES_6.* For these experiments, $timePerOrder$ and $adjustmentFactor$ are set to 1 second and 10 respectively and reevaluation is triggered for a batch size of 50 customer orders.

Figure 7.10 illustrates the results of this analysis. As shown in this figure, by increasing the number of FCs, the average shipping cost per order decreases. However, this is not because the reevaluation algorithm performs better for instances with more FCs. Increasing the number of FCs reduces the average distance between FCs and customer orders which subsequently reduces average shipping cost per order. Based on the optimization gap percentage that is reported for these instances, the reevaluation decisions tend to be negatively impacted by increasing number of FCs. In other words, this experiment confirms that an e-tailer with a more sophisticated supply chain network requires more processing power to reevaluate its customer orders.

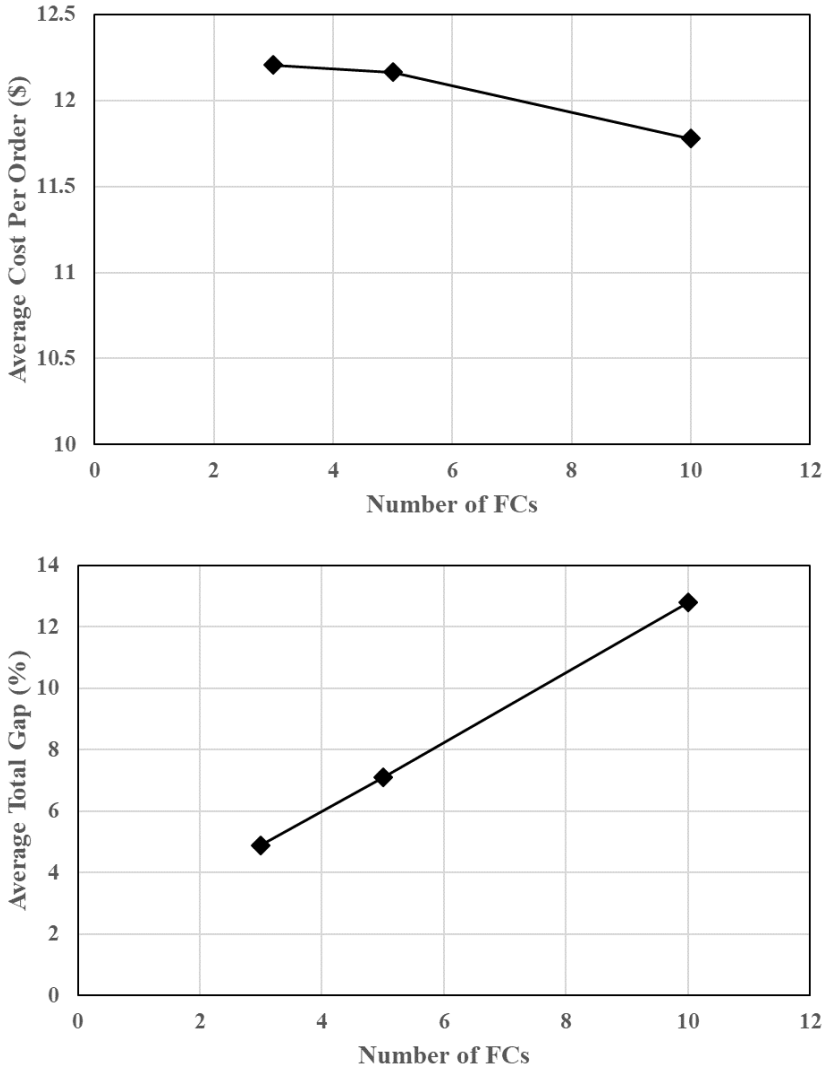

Figure 7.10: Impact of number of FCs on system performance

7.3.10. Impact of number of SKUs

In this experiment, we extend the analysis that was presented in Section 7.3.1 where we studied the impact of the number of SKUs on the performance of the IP-based reevaluation algorithm performance. In order to quantify how e-tailer system performance is influenced by the number of SKUs, a set of experiments are conducted with the DES model using three problem instances: *DES_2*, *DES_3* and *DES_4*. As shown in Table 7.2, these instances have a similar configuration except for the number of SKUs. The number of SKUs for *DES_2*, *DES_3* and *DES_4* is 5, 10, and 100 respectively.

All DES model parameters are fixed throughout this experiment. The reevaluation algorithm is triggered for a batch size of 50 orders and the values of $timePerOrder$ and $adjustmentFactor$ are set to 1 second and 10 respectively. The results are summarized in Figure 7.11. According to this figure, average cost per order does not demonstrate a correlation with the number of SKUs. This can be explained by the fact that in this simulation the number of lines in a customer order is independent of number of SKUs in e-tailer's product catalog. Therefore, the average shipping cost per order and number of SKUs are not closely related.

Note that in each execution of the reevaluation algorithm, only the SKUs in the customer orders that are reevaluated are considered by the integer program. In other words, if number of SKUs is 100, but only 30 of those SKUs are included in any of the 50 orders that are reevaluated together, the integer program for that reevaluation is constructed with the smaller subset of SKUs. In Section 7.3.1, we considered an exponential distribution to identify the SKUs that are ordered by each customer while in this section we use a uniform distribution. Therefore, the number of SKUs that are in the subset for this case is larger than what was presented previously. Hence, the optimization gap percentage in this experiment has a positive correlation with the number of SKUs.

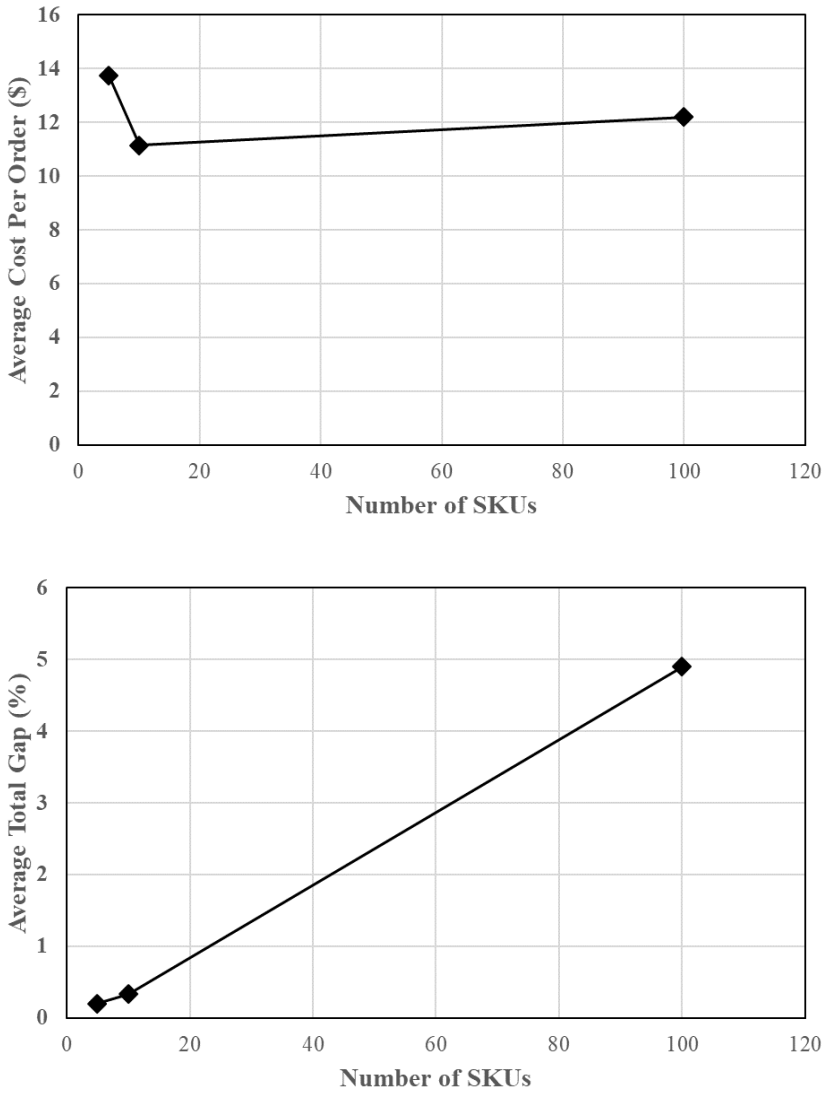

Figure 7.11: Impact of number of SKUs on system performance

7.3.11. Heuristic vs. IP-based reevaluation

Although the IP-based reevaluation algorithm is effective in reducing shipping costs, it has some limitations for larger problem instances. As shown in previous experiments, parameters such as the number of FCs, SKUs, and reevaluation batch size add to the complexity of the integer program and increase the time needed to find an optimal decision. To solve larger problem instances, a heuristic reevaluation algorithm is presented that reduces average shipping cost for the e-tailer and can be triggered using the same mechanisms that were described for the IP-based reevaluation algorithm. In this section, experiments are conducted to analyze the performance of this heuristic algorithm in comparison with the integer program.

Figure 7.12 illustrates the results of these experiments. According to this figure, although the integer program performs better for smaller batch sizes, its performance degrades as the batch size increases. The heuristic algorithm, on the other hand, demonstrates more consistent performance, and although it does not reduce the shipping cost to the same level as the integer program, it can reevaluate a larger batch of customer orders. On the other hand, since the heuristic algorithm is triggered like the integer program and its computation time is controlled with the same approach, it does not impact the service level.

This experiment can be replicated for other parameters that increase the complexity of the problem such as the number of SKUs and FCs. In general, for more complex reevaluation problems, the heuristic algorithm may be a better alternative to consider that can reduce shipping costs within a feasible timeframe.

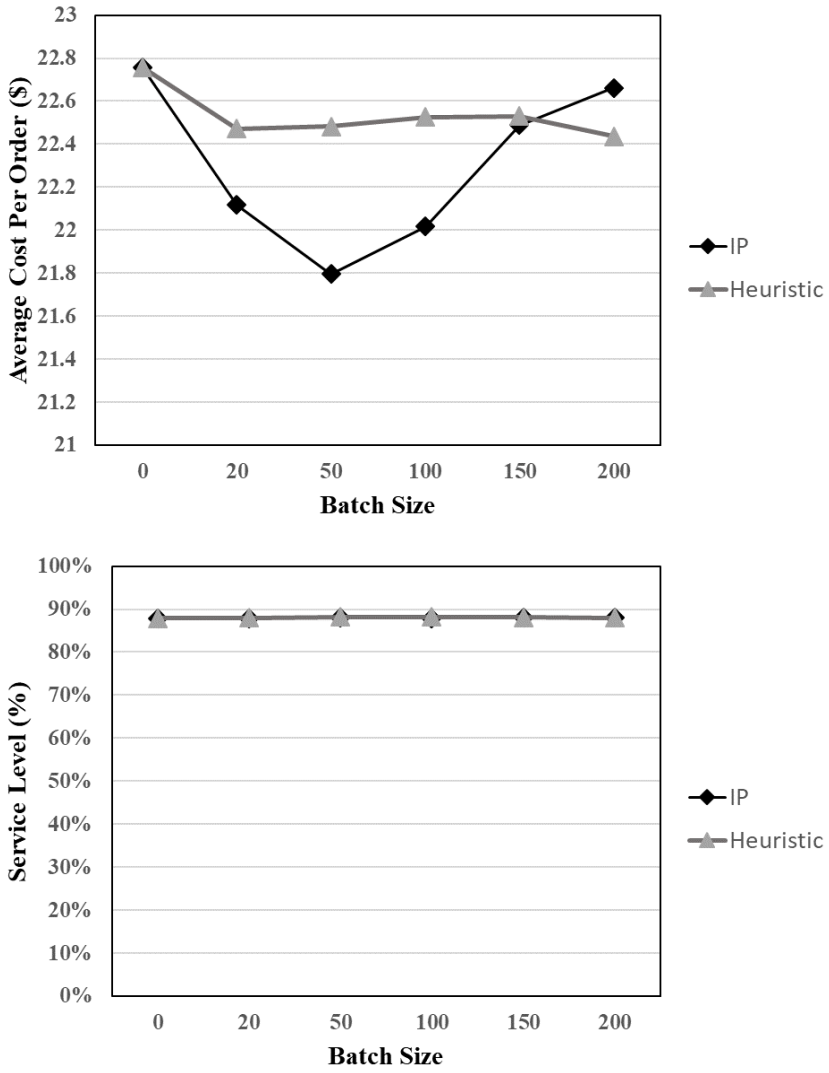

Figure 7.12: Heuristic vs. IP-based reevaluation algorithm performance comparison

Chapter 8

Conclusions and future work

In this dissertation we have fully embedded two decision making algorithms within a DES model of a general e-tailer order fulfillment process. This is the first study in the literature to integrate integer programming and discrete event simulation in a novel way by feeding both the decisions produced and the computation time used by the integer program to the DES model to improve an e-tailer's order fulfillment decisions. The DES model simulates daily operations of an e-tailer by considering important processes such as orders being placed, order fulfillment, shipment, and inventory replenishment. Order fulfillment decisions in the DES model are made when a customer places an order an using a rule-based method that assigns each customer order to one or more FCs that can fulfill them with a minimum number of shipments. This is a common practice that allows e-tailers to maintain an updated available-to-promise inventory record for all FCs and to provide an estimated delivery window to their customers.

We demonstrated that although making fulfillment decisions on the fly is critical for maintaining the e-tailer's operations, these decisions are made solely based on the available information at the time an order is placed and with the objective of minimizing the shipping cost for that individual order. In other words, they lead to a series of myopic fulfillment decisions that are locally optimized for each customer order, but when considered holistically, they do not globally minimize the e-tailer's total shipping cost. In order to solve this problem, we presented an IP-based reevaluation algorithm and a heuristic algorithm that simultaneously consider the fulfillment decisions for a group of customer orders by shuffling the assignments that are made by the myopic decisions.

E-tailers receive orders around the clock, 24 hours a day and 7 days a week. Reevaluation needs to occur regularly during normal operations and without impacting important business KPIs such as customer service level. During reevaluation, in addition to the inventory units that are assigned to customer orders by myopic decisions, a portion of the un-assigned inventory at each FC gets reserved and is made available to the reevaluation algorithm to improve its decisions. That inventory may not be used to fulfill other customer orders that are placed while reevaluation is being executed. Note that other processes may rely on or be impacted by the reevaluation. For example, if customer shipments are sent from each FC at a fixed time every day, the fulfillment decision for orders must be locked before that time. Therefore, reevaluation computation time must be considered when designing a reevaluation strategy for an e-tailer.

Since both reevaluation decisions and computation time must be studied within the context of e-tail operations and by considering the dependencies and relationships among multiple events, we used a novel technique to fully embed our two reevaluation algorithms within a discrete event simulation model. This framework enables us to design a reevaluation strategy that fits the e-tailer's operational characteristics. This also allows e-tailers to test multiple reevaluation scenarios with different configurations in a simulated environment before selecting and deploying the desired strategy to their fulfillment system.

The IP-based reevaluation algorithm is a complex problem which requires a nontrivial amount of computation time. On one hand, reevaluating the fulfillment decisions for a larger group of customer orders (with a larger reevaluation batch size) allows the integer program to generate better decisions. On the other hand, a higher batch size requires more computation time which has a negative impact on system performance. Because of this tradeoff, selecting the best strategy for triggering the reevaluation algorithm is a nontrivial problem. We proposed two different methods

for doing this: (i) reevaluating for fixed batch sizes and (ii) reevaluating at a fixed time interval. Additionally, we developed a framework that allows e-tailers to find the best value for the batch size or time interval that minimizes total shipping cost.

The computation time for the IP-based reevaluation algorithm is dependent on several model parameters including the number of FCs, SKUs, customer orders, and order lines. For an e-tailer with a complex supply chain which contains millions of SKUs and hundreds of FCs, the integer program may not find an optimal solution in a reasonable amount of time. Therefore, we developed a heuristic reevaluation algorithm as an alternative. The heuristic algorithm's computation time can be controlled by the e-tailer. According to our experimental results, the heuristic algorithm does not decrease total shipping cost as much as the integer program, but it shows a more consistent performance for large problem instances.

The experimental results yield several managerial insights. First, our ability to significantly reduce total shipping cost for customer orders using reevaluation demonstrates the effectiveness of this approach. Second, in order to asses a reevaluation algorithm for making order fulfillment decisions, both the decisions generated and computation time used by the algorithm need to be considered. A reevaluation algorithm that generates high-quality decisions but requires a long computation time may not be the best fit for a fast-paced e-tailer that receives hundreds of customer orders in an hour. Third, there is a close relationship between the decisions generated by the reevaluation algorithm, the computation time it uses, and the method by which it is triggered. Increasing the frequency of reevaluation reduces the ability of the reevaluation algorithm to shuffle assignments among a larger group of orders. On the other hand, it also decreases the reevaluation computation time and reduces the optimization gap. A successful reevaluation strategy is therefore a combination of an effective algorithm and a triggering method. Since these two are interrelated,

the best approach to find a successful strategy is experimenting with multiple scenarios. Our DES model provides a framework for such analysis.

Future work on this problem might proceed in several directions. First, the DES model can be extended to consider other processes in the e-tailer's operations. For example, returns and reverse logistics is an important aspect of an e-tailer supply chain. Some inventory units that are returned to the e-tailer could be assigned to new customer orders. Our proposed DES model has a modular design which is based on object-oriented programming and supports the addition of new events and processes for further studies. Second, the objective function of the reevaluation integer program can be extended to include other cost elements such as inventory holding cost and order processing cost. Although outbound transportation cost (i.e. order shipping cost) accounts for a significant portion of an e-tailer's overall operating cost, including other cost elements increases the effectiveness of the reevaluation algorithm. Third, there are research opportunities for improving the reevaluation triggering methods. In this dissertation, we proposed two methods based on a fixed batch size and fixed cycle time. We believe this could be augmented with other techniques that consider triggering the reevaluation algorithm based on the characteristics of customer orders as well as inventory levels at FCs. In other words, reevaluation can be triggered according to the level of SKU overlap within a set of customer orders as opposed to the size of that set. Fourth, there are two directions in which the proposed heuristic algorithm could be improved. Instead of splitting the batch of customer orders to equal size subsets, the proportion could be a model parameter that is customizable for each e-tailer according to their specific business requirements. Additionally, although in this dissertation, we split customer orders only once and apply the decomposed integer program to one of the subsets, this procedure could be replicated multiple times with different randomized subsets to make a better overall decision. Fifth,

extending this model to analyze a retailer with an omni-channel supply chain network is a worthwhile direction for future work. Finally, this model can be used to study the impact of a crisis such as COVID-19 on an e-tailer order fulfillment process. COVID-19 has impacted e-tailers in many ways by causing demand volatility and supply shortages, and by increasing replenishment lead time, and creating logistics and transportation challenges. Since our proposed model is designed in a modular way and it is highly parameterized, decision makers can study the impact of many of these disruptions by either adjusting the model parameters or adding additional events to the simulation model. This can be extended to any future crisis and disruptions to allow e-tailers to proactively prepare their supply chains for those events.

Appendix A

Linear regression models for UPS shipping rates

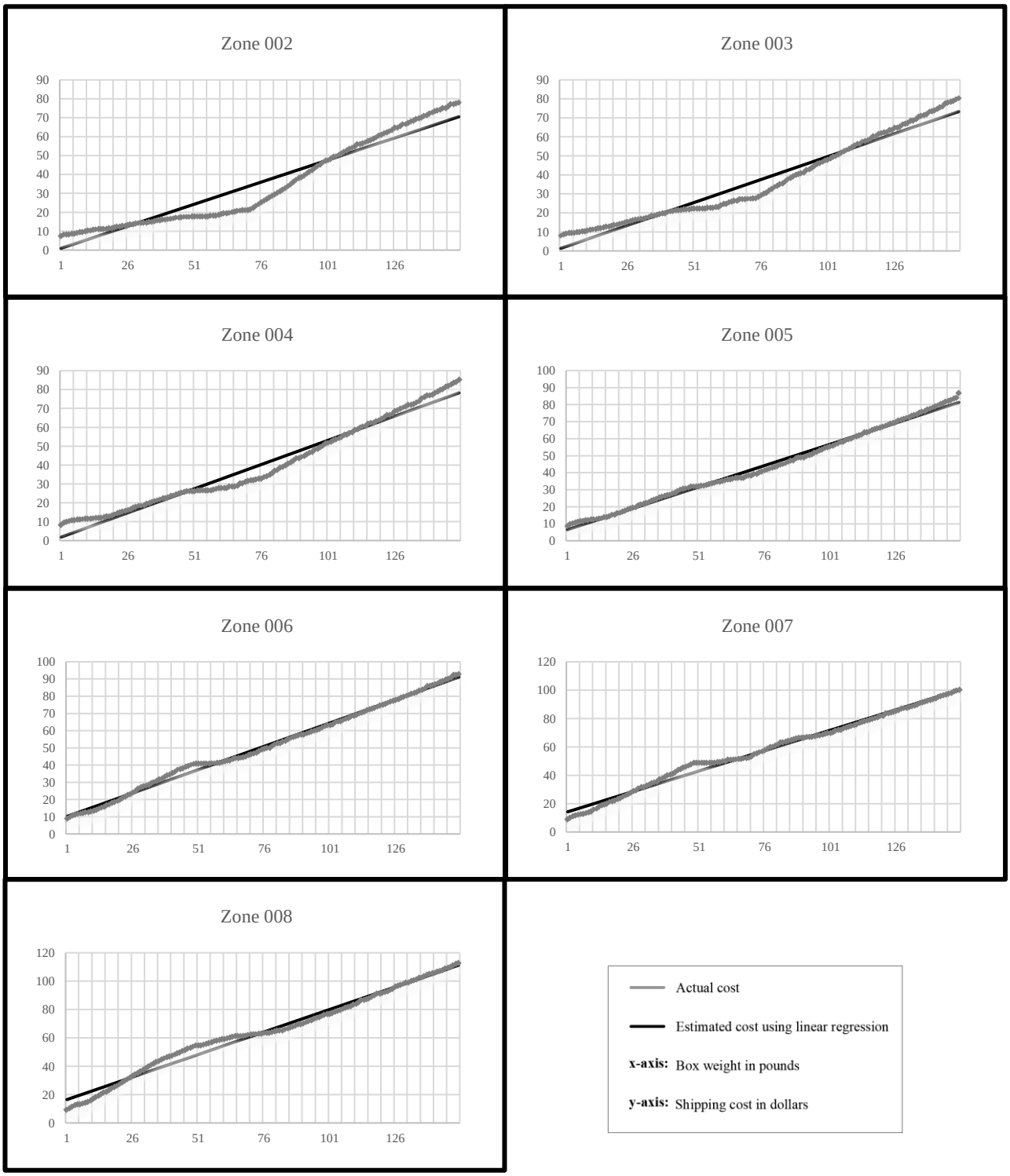

Figure 1: Fitted linear regression models for shipping method 1

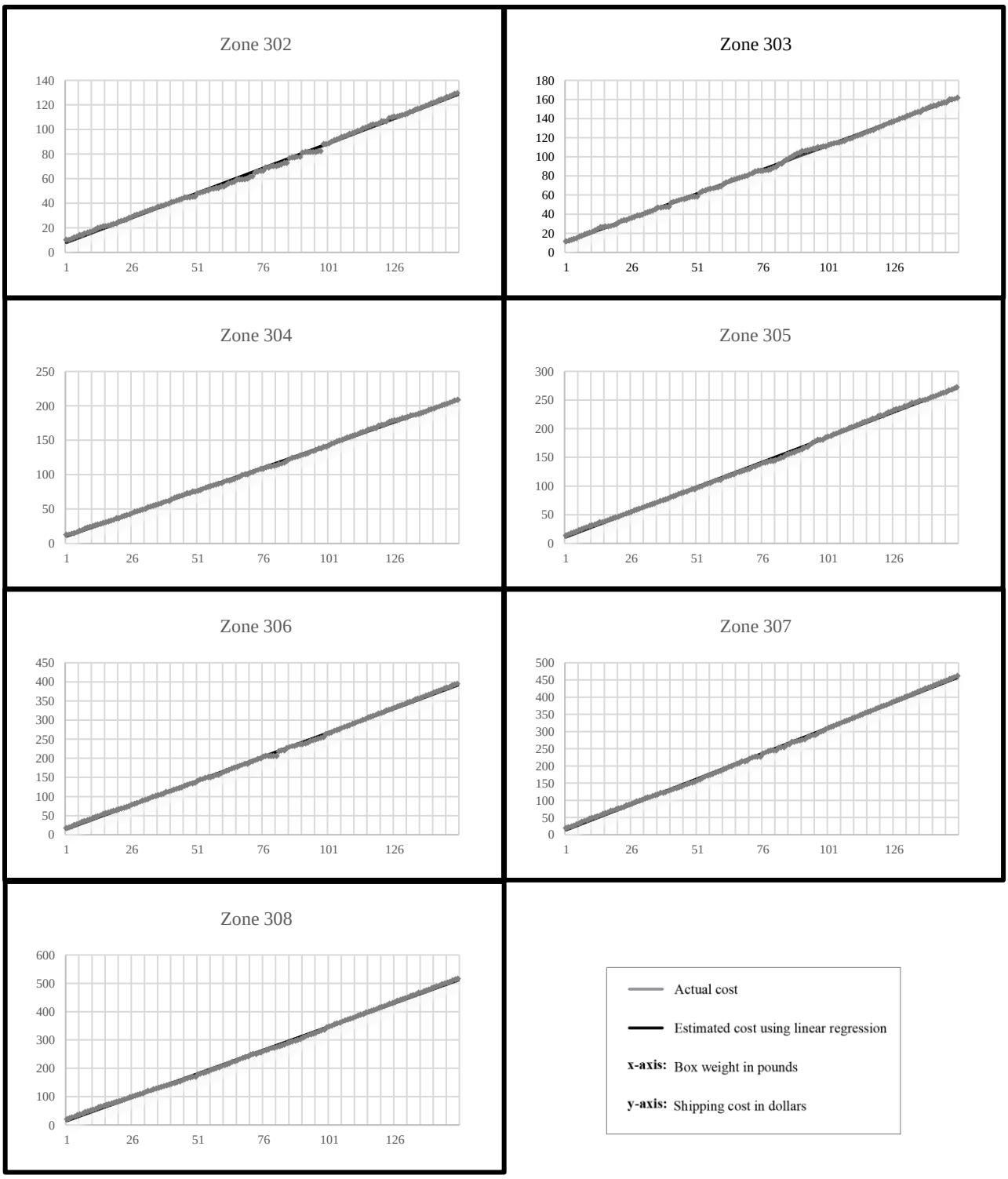

Figure 2: Fitted linear regression models for shipping method 2

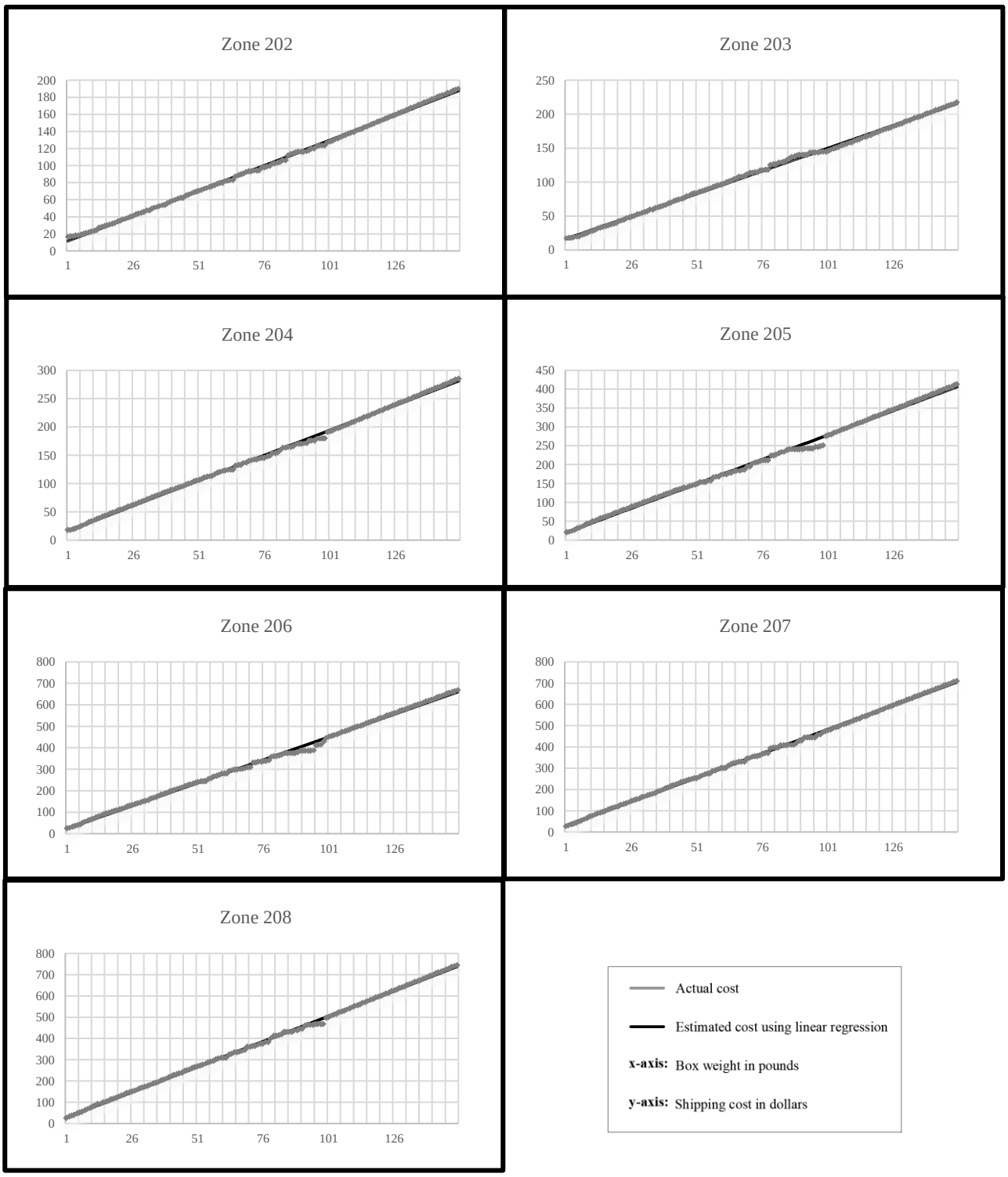

Figure 3: Fitted linear regression models for shipping method 3

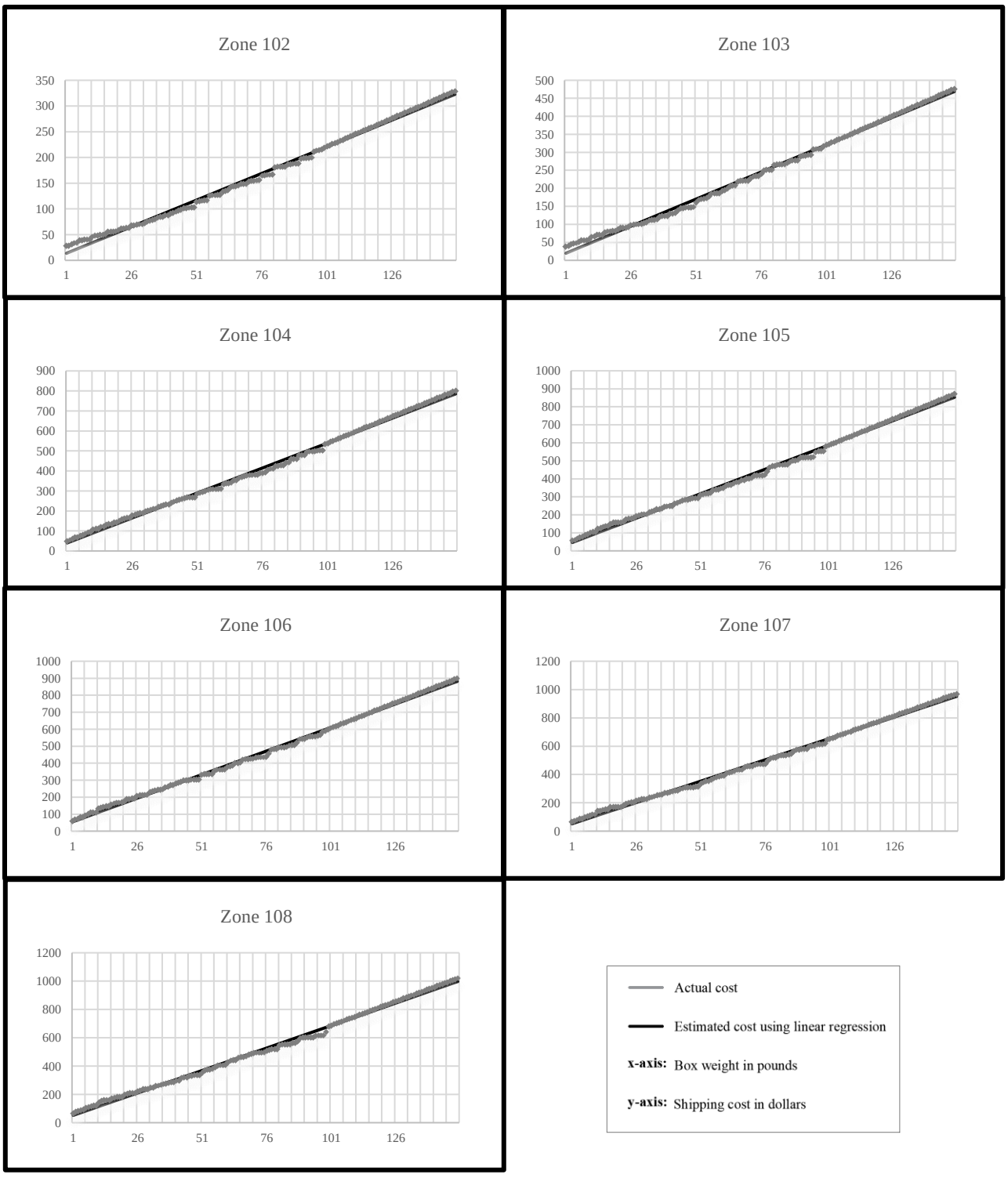

Figure 4: Fitted linear regression models for shipping method 4

Appendix B

Simulation model pseudocodes

Table 1: Zone calculator pseudocode

Procedure: *Zone*
Input: $orgLoc, destLoc, shippingMehod$

```
shipDistance ← distance(orgLoc, destLoc)
if shippingMethod = 1
    if shipDistance ≤ 165
        zoneNum ← 2
    else if shipDistance ≤ 308
        zoneNum ← 3
    else if shipDistance ≤ 607
        zoneNum ← 4
    else if shipDistance ≤ 1020
        zoneNum ← 5
    else if shipDistance ≤ 1440
        zoneNum ← 6
    else
        zoneNum ← 7
    endif
else if shippingMethod = 2
    if shipDistance ≤ 165
        zoneNum ← 302
    else if shipDistance ≤ 308
        zoneNum ← 303
    else if shipDistance ≤ 607
        zoneNum ← 304
    else if shipDistance ≤ 1020
        zoneNum ← 305
    else if shipDistance ≤ 1440
        zoneNum ← 306
    else
        zoneNum ← 307
    endif
else if shippingMethod = 3
    if shipDistance ≤ 165
        zoneNum ← 202
    else if shipDistance ≤ 308
        zoneNum ← 203
    else if shipDistance ≤ 607
        zoneNum ← 204
    else if shipDistance ≤ 1020
        zoneNum ← 205
    else if shipDistance ≤ 1440
        zoneNum ← 206
    else
        zoneNum ← 207
    endif
else if shippingMethod = 4
    if shipDistance ≤ 165
        zoneNum ← 102
    else if shipDistance ≤ 308
        zoneNum ← 103
    else if shipDistance ≤ 607
        zoneNum ← 104
    else if shipDistance ≤ 1020
        zoneNum ← 105
    else if shipDistance ≤ 1440
        zoneNum ← 106
    else
        zoneNum ← 107
    endif
endif
return (zoneNum)
```

Table 2: Shipping cost per box calculator pseudocode

Procedure: *ShipCostPerBoxCalc*
Input: *shippingZone, shippingMehod*

```
if shippingMethod = 1
    if shippingZone = 2
        costPerBox ← 0.50
    else if shippingZone = 3
        costPerBox ← 0.90
    else if shippingZone = 4
        costPerBox ← 1.36
    else if shippingZone = 5
        costPerBox ← 6.01
    else if shippingZone = 6
        costPerBox ← 9.70
    else
        costPerBox ← 13.62
    endif
else if shippingMethod = 2
    if shippingZone = 302
        costPerBox ← 7.23
    else if shippingZone = 303
        costPerBox ← 9.87
    else if shippingZone = 304
        costPerBox ← 9.28
    else if shippingZone = 305
        costPerBox ← 9.30
    else if shippingZone = 306
        costPerBox ← 12.31
    else
        costPerBox ← 11.16
    endif
else if shippingMethod = 3
    if shippingZone = 202
        costPerBox ← 10.35
    else if shippingZone = 203
        costPerBox ← 15.01
    else if shippingZone = 204
        costPerBox ← 15.22
    else if shippingZone = 205
        costPerBox ← 17.50
    else if shippingZone = 206
        costPerBox ← 19.85
    else
        costPerBox ← 25.64
    endif
else if shippingMethod = 4
    if shippingZone = 102
        costPerBox ← 11.53
    else if shippingZone = 103
        costPerBox ← 16.57
    else if shippingZone = 104
        costPerBox ← 35.21
    else if shippingZone = 105
        costPerBox ← 41.15
    else if shippingZone = 106
        costPerBox ← 43.56
    else
        costPerBox ← 45.37
    endif
endif
return(costPerBox)
```

Table 3: Shipping cost per pound calculator pseudocode

```
Procedure: ShipCostPerPoundCalc
Input: shippingZone, shippingMehod
if shippingMethod = 1
    if shippingZone = 2
        costPerPound ← 0.47
    else if shippingZone = 3
        costPerPound ← 0.48
    else if shippingZone = 4
        costPerPound ← 0.51
    else if shippingZone = 5
        costPerPound ← 0.50
    else if shippingZone = 6
        costPerPound ← 0.54
    else
        costPerPound ← 0.54
    endif
else if shippingMethod = 2
    if shippingZone = 302
        costPerPound ← 0.81
    else if shippingZone = 303
        costPerPound ← 1.01
    else if shippingZone = 304
        costPerPound ← 1.33
    else if shippingZone = 305
        costPerPound ← 1.75
    else if shippingZone = 306
        costPerPound ← 2.52
    else
        costPerPound ← 2.97
    endif
else if shippingMethod = 3
    if shippingZone = 202
        costPerPound ← 1.18
    else if shippingZone = 203
        costPerPound ← 1.34
    else if shippingZone = 204
        costPerPound ← 1.34
    else if shippingZone = 205
        costPerPound ← 2.59
    else if shippingZone = 206
        costPerPound ← 4.27
    else
        costPerPound ← 4.52
    endif
else if shippingMethod = 4
    if shippingZone = 102
        costPerPound ← 2.07
    else if shippingZone = 103
        costPerPound ← 3.01
    else if shippingZone = 104
        costPerPound ← 5.00
    else if shippingZone = 105
        costPerPound ← 5.41
    else if shippingZone = 106
        costPerPound ← 5.54
    else
        costPerPound ← 5.54
    endif
endif
return(costPerPound)
```

Table 4: Customer order placement event pseudocode

Procedure: $CustomerOrderPlacement$
Input: $sysState, currTime$

```
increase numOrd by 1
initialize nOrd as an empty instance of Order
nOrd.ordNum ← numOrd
nOrd.ordTime ← currTime
nOrd.ordHour ← Hour(currTime)
nOrd.ordDay ← Day(currTime)
nOrd.ordLoc.xCoord ← unifDist(0, Length)
nOrd.ordLoc.yCoord ← unifDist(0, Width)
rand1 ← unifDist(0,1)
rand2 ← unifDist(0,1)

/* Specify new order's delivery preference based on cumulative probability distribution */
if rand1 ≤ DelProb[1]
    nOrd.delPref ← 1
    if nOrd.ordHour < DailyShipHour
        nOrd.estDel ← (nOrd.ordDay + DeliveryDays[1]) * 1440 + unifDist(480,1140)
    else
        nOrd.estDel ← (nOrd.ordDay + DeliveryDays[1] + 1) * 1440 + unifDist(480,1140)
    endif
else if rand1 ≤ DelProb[2]
    nOrd.delPref ← 2
    if nOrd.ordHour < DailyShipHour
        nOrd.estDel ← (nOrd.ordDay + DeliveryDays[2]) * 1440 + unifDist(480,1140)
    else
        nOrd.estDel ← (nOrd.ordDay + DeliveryDays[2] + 1) * 1440 + unifDist(480,1140)
    endif
else if rand1 ≤ DelProb[3]
    nOrd.delPref ← 3
    if nOrd.ordHour < DailyShipHour
        nOrd.estDel ← (nOrd.ordDay + DeliveryDays[3]) * 1440 + unifDist(720,1140)
    else
        nOrd.estDel ← (nOrd.ordDay + DeliveryDays[3] + 1) * 1440 + unifDist(720,1140)
    endif
else
    nOrd.delPref ← 4
    if nOrd.ordHour < DailyShipHour
        nOrd.estDel ← (nOrd.ordDay + DeliveryDays[4]) * 1440 + unifDist(900,1140)
    else
        nOrd.estDel ← (nOrd.orderDay + DeliveryDays[4] + 1) * 1440 + unifDist(900,1140)
    endif
endif

/* Specify when this order must be locked based on its arrival time */
nOrd.mustLockTime ← (Day(nOrd.estDel) − 1) * 1440 + 780

/* Determine number of line items in the order based on cumulative probability distribution */
for i = 1 to MaxOrdLines
    if rand2 ≤ OrdLinesProb[1]
        nOrd.numItems ← i
        exit for loop
    endif
endfor

/* Specify SKUs for each order line based on cumulative probability distribution */
check ← false
for i = 1 to nOrd.numItems
    while check is false
        check ← true
            rand3 ← unifDist(0,1)
            for s = 1 to NumSkus
                if rand3 ≤ SkuProb[s]
                    candidateSku ← s
                    exit for loop
                endif
            endfor
```

```
            for j = 1 to i
                if nOrd.item[j].skuNum = candidateSku
                    check ← false
                    exit for loop
                endif
            endfor
        endwhile
        nOrd.item[i].skuNum ← candidateSku
        check ← false
    endfor

    /* Specify order quantity for each order line based on cumulative probability distribution */
    for i = 1 to nOrd.numItems
        rand4 ← unifDist(0,1)
        for q = 1 to MaxQuantity
            if rand4 ≤ QtyProb[q]
                nOrd.item[i].qty ← q
                exit for loop
            endif
        endfor
    endfor

    /* Update statistical accumulators */
    for i = 1 to nOrd.numItems
        increase TotalDemand[nOrd.item[i].skuNum] by nOrd.item[i].qty
    endfor

    /* Update system state */
    sysState.timeOfMostRecentEvent ← currTime
    push nOrd into sysState.orderQueue
    push nOrd.ordNum into sysState.openOrderNumbers
    sysState.numfOrders ← nOrd.ordNum
    increase sysState.numOfOpenOrders by 1

    /* Update event calendar and call other events */
    call CheckInventoryAvailability() event
    put the next customer order arrival event in the calendar
```

Table 5: Check inventory availability pseudocode

Procedure: *CheckInventoryAvailability*
Input: $custOrd, sysState, currTime$

$acceptOder \leftarrow true$

/* For each order line, check inventory availability at all FCs and determine if the customer order can be satisfied */
for $i = 1$ to $custOrd.numItems$
    o$nHandInv \leftarrow 0$
    $inOrderInv \leftarrow 0$
    $sNum \leftarrow custOrd.item[i].skuNum$
    for $f = 1$ to $NumFcs$
        increase o$nHandInv$ by $sysState.inv[sNum][f].onHandUnassigned$
        if $sysState.inv[sNum][f].repTime \leq custOrd.mustLockTime$
            increase $inOrderInv$ by $sysState.inv[sNum][f].inOrderUnassigned$
        endif
    endfor
    if o$nHandInv + inOrderInv < custOrd.item[i].qty$
        $acceptOrder \leftarrow false$
        exit for loop
    endif
endfor

/* Update event calendar and call other events */
if $acceptOrder = true$
    Call $AcceptCustomerOrder()$ event
else
    Call $RejectCustomerOrder()$ event
endif

Table 6: Accept customer order pseudocode

Procedure: *AcceptCustomerOrder*
Input: $custOrd, sysState, currTime$

/* Update statistical accumulators */
increase $NumOrdersAccepted$ by 1

/* Update event calendar and call other events */
call $MakeOrderFulfillmentDecision()$ event

Table 7: Reject customer order pseudocode

Procedure: *RejectCustomerOrder*
Input: $custOrd, sysState, currTime$

/* Update statistical accumulators */
increase $NumOrdersRejected$ by 1

/* Update system state */
decrease $sysState.numberOfOpenOrders$ by 1
increase $sysState.numberOfClosedOrders$ by 1
push $custOrd.ordNum$ into $sysState.closedOrderNumbers$
remove $custOrd.ordNum$ from $sysState.openOrderNumbers$
remove $custOrd$ from $sysState.orderQueue$

Table 8: Make order fulfillment decision pseudocode

Procedure: $MakeOrderFulfillmentDecision$
Input: $custOrd, sysState, currTime$

```
/* For each order item, specify number of units of on-hand and in-order inventory that are eligible to satisfy customer order */
for i = 1 to custOrd.numItems
    sNum ← cusOrd.item[i].skuNum
    for f = 1 to NumFcs
        eligibleOH[i][f] ← sysState.inv[sNum][f].onHandUnassigned
        if sysState.inv[sNum][f].repTime ≤ custOrd.mustLockTime
            eligibleIO[i][f] ← sysState.inv[sNum][f].inOrderUnassigned
        else
            eligibleIO[i][f] ← 0
        endif
    endfor
endfor

/* Specify which order items can be fulfilled by each FC. Additionally, calculate total number of order items each FC can satisfy*/
for f = 1 to NumFcs
    numLinesFcCanFulfill[f] ← 0
    for i = 1 to custOrd.numItems
        if eligibleOH[i][f] + eligibleIO[i][f] ≥ custOrd.item[i].qty
            increase numLinesFcCanFulfill[f] by 1
            canFulfill[i][f] ← "true"
        else
            canFulfill[i][f] ← "false"
        endif
    endfor
endfor

/* Calculate a weight factor for each FC based on number of items it can satisfy and its distance to customer location */
for f = 1 to NumFcs
    fcWeight[f] ← numLinesFcCanFulfill[f] * 1000 − distance(fc[f].location, custOrd.location)
endfor

/* Rank FCs based on the calculated weight */
for f = 1 to NumFcs
    rFcs[f] ← f
endfor
j ← 1
swapped ← "true"
while swapped = "true"
    swapped ← "false"
    for f = 1 to NumFcs − j
        if fcWeight[f] < fcWeight[f + 1]
            temp1 ← fcWeight[f]
            temp2 ← rFcs[f]
            fcWeight[f] ← fcWeight[f + 1]
            rFcs[f] ← rFcs[f + 1]
            fcWeight[f + 1] ← temp1
            rFcs[f] ← temp2
            swapped ← "true"
        endif
    endfor
    increase j by 1
endwhile

/* Assign each order items to a FC. When assigning items to FCs, this algorithm first checks the FC that can fulfill maximum number of order items
and assigns the item to it if there is enough inventory. Otherwise it checks the second FC in the ranked list and repeats the logic until all items are
assigned. If no FC has enough inventory to fulfill an order item, the algorithm splits that item into multiple assignments. */
for i = 1 to custOrd.numItems
    found ← "false"
    sNum ← custOrd.item[i].skuNum
    for f = 1 to NumFcs
        if canFulfill[i][rFcs[f]] = "true"
            push rFcs[f] into fcForAsg[i]
            found ← "true"
            if eligibleOH[i][rFcs[f]] ≥ custOrd.item[i].qty
                push custOrd.item[i].qty into qtyFromOH[i]
                push 0 into qtyFromIO[i]
```

```
                shMthd ← ShipMethodCalc(custOrd.delPref, 0, hour(currTime)
                shTime ← ShipTimeCalc(Day(currTime), Hour(currTime), custOrd.delPref, shMthd)
                lockTime ← shTime − 60
                push shMthd into asgShMthd[i]
                push shTime into asgShTime[i]
                push lockTime into asgLockTime[i]
            else
                push eligibleOH[i][rFcs[f]] into qtyFromOH[i]
                push (custOrd.item[i].qty − eligibleOH[i][rFcs[f]]) into qtyFromIO[i]
                shMthd ← ShipMethodCalc(custOrd.delPref,
                Day(sysState.inv[sNum][rFcs[f].repTime) − Day(currTime), Hour(currTime))
                shTime ← ShipTimeCalc(Day(currTime), Hour(currTime), custOrd.delPref, shMthd)
                lockTime ← shTime − 60
                push shMthd into asgShMthd[i]
                push shTime into asgShTime[i]
                push lockTime into asgLockTime[i]
            endif
        endif
    endfor
    if found = "false"
        remQty ← custOrd.item[i].qty
        f ← 1
        while remQty > 0
            if eligibleOH[i][rFcs[f]]+ eligibleIO[i][rFcs[f]] > 0
                push rFcs[f] into fcForAsg[i]
                if remQty > (eligibleOH[i][rFcs[f]]+ eligibleIO[i][rFcs[f]])
                    push eligibleOH[i][rFcs[f]] into qtyFromOH[i]
                    push eligibleIO[i][rFcs[f]] into qtyFromIO[i]
                    if eligibleIO[i][rFcs[f]] > 0
                        shMthd ← ShipMethodCalc(custOrd.delPref,
                        Day(sysState.inv[sNum][rFcs[f].repTime) − Day(currTime), Hour(currTime))
                        shTime ← ShipTimeCalc(Day(currTime), Hour(currTime), custOrd.delPref, shMthd)
                        lockTime ← shTime − 60
                        push shMthd into asgShMthd[i]
                        push shTime into asgShTime[i]
                        push lockTime into asgLockTime[i]
                    else
                        shMthd ← ShipMethodCalc(custOrd.delPref, 0, Hour(currTime))
                        shTime ← ShipTimeCalc(Day(currTime), Hour(currTime), custOrd.delPref, shMthd)
                        lockTime ← shTime − 60
                        push shMthd into asgShMthd[i]
                        push shTime into asgShTime[i]
                        push lockTime into asgLockTime[i]
                    endif
                else if eligibleOH[i][rFcs[f]] ≥ unfulfilledQuantity
                    push remQty into qtyFromOH[i]
                    push 0 into qtyFromIO[i]
                    shMthd ← ShipMethodCalc(custOrd.delPref, 0, Hour(currTime))
                    shTime ← ShipTimeCalc(Day(currTime), Hour(currTime), custOrd.delPref, shMthd)
                    lockTime ← shTime − 60
                    push shMthd into asgShMthd[i]
                    push shTime into asgShTime[i]
                    push lockTime into asgLockTime[i]
                else
                    push eligibleOH[i][rFcs[f]] into qtyFromOH[i]
                    if eligibleIO[i][rFcs[f]] > 0
                        push remQty − eligibleOH[i][rFcs[f]] into qtyFromIO[i]
                        shMthd ← ShipMethodCalc(custOrd.delPref,
                        Day(sysState.inv[sNum][rFcs[f].repTime) − Day(currTime), Hour(currTime))
                        shTime ← ShipTimeCalc(Day(currTime), Hour(currTime), custOrd.delPref, shMthd)
                        lockTime ← shTime − 60
                        push shMthd into asgShMthd[i]
                        push shTime into asgShTime[i]
                        push lockTime into asgLockTime[i]
                    else
                        push 0 into qtyFromIO[i]
```

```
                        shMthd ← ShipMethodCalc(custOrd.delPref, 0, Hour(currTime))
                        shTime ← ShipTimeCalc(Day(currTime), Hour(currTime), custOrd.delPref, shMthd)
                        lockTime ← shTime − 60
                        push shMthd into asgShMthd[i]
                        push shTime into asgShTime[i]
                        push lockTime into asgLockTime[i]
                    endif
                endif
                decrease remQty by eligibleOH[i][rFcs[f]] + eligibleIO[i][rFcs[f]]
            endif
            f ← f + 1
        endwhile
    endif
    found ← "true"
endfor

/* Update system state with information about fulfillment decision */
for i = 1 to custOrd.numItems
    for f = 1 to fcForAsg[i].size()
        initialize assignment to be an empty instance of Assignment
        assignment.ordNum ← custOrd.orderNumber
        assignment.skuNum ← custOrd.item_i.skuNum
        assignment.fcNum ← fcForAsg[i][f]
        assignment.qtyFromOnHand ← qtyFromOH[i][f]
        assignment.qtyFromInOrder ← qtyFromIO[i][f]
        assignment.shippingMethod ← asgShMthd[i][f]
        assignment.shippingTime ← asgShTime[i][f]
        assignment.lockingTime ← asgLockTime[i][f]
        assignment.locked ← "false"
        push assignment into sysState.assignments
        decrease sysState.inv[assignment.skuNum][assignment.fcNum].onHandUnassigned by qtyFromOH[i][f]
        decrease sysState.inv[assignment.skuNum][assignment.fcNum].inOrderUnassigned by qtyFromIO[i][f]
        increase sysState.inv[assignment.skuNum][assignment.fcNum].onHandAssignedOpen by qtyFromOH[i][f]
        increase sysState.inv[assignment.skuNum][assignment.fcNum].onOrderAssignedOpen by qtyFromIO[i][f]
    endfor
endfor
sysState.timeOfMostRecentEvent ← currTime
```

Table 9: Shipping method calculator pseudocode

```
Procedure: ShipMethodCalc
Input: deliveryOption, invAvailDay, orderHour
if orderHour < 13
    if deliveryOption = 1
        if invAvailDay ≤ 2
            shippingMethod ← 1
        else if invAvailDay ≤ 4
            shippingMethod ← 2
        else if invAvailDay = 5
            shippingMethod ← 3
        else if invAvailDay = 6
            shippingMethod ← 4
        endif
    else if deliveryOption = 2
        if invAvailDay = 0
            shippingMethod ← 1
        else if invAvailDay ≤ 2
            shippingMethod ← 2
        else if invAvailDay = 3
            shippingMethod ← 3
        else if invAvailDay = 4
            shippingMethod ← 4
        endif
    else if deliveryOption = 3
        if invAvailDay = 0
            shippingMethod ← 3
        else if invAvailDay = 1
            shippingMethod ← 4
        endif
    else if deliveryOption = 4
        if invAvailDay = 0
            shippingMethod ← 4
        endif
    endif
else
    if deliveryOption = 1
        if invAvailDay ≤ 3
            shippingMethod ← 1
        else if invAvailDay ≤ 5
            shippingMethod ← 2
        else if invAvailDay = 6
            shippingMethod ← 3
        else if invAvailDay = 7
            shippingMethod ← 4
        endif
    else if deliveryOption = 2
        if invAvailDay ≤ 1
            shippingMethod ← 1
        else if invAvailDay ≤ 3
            shippingMethod ← 2
        else if invAvailDay = 4
            shippingMethod ← 3
        else if invAvailDay = 5
            shippingMethod ← 4
        endif
    else if deliveryOption = 3
        if invAvailDay ≤ 1
            shippingMethod ← 3
        else if invAvailDay = 2
            shippingMethod ← 4
        endif
    else if deliveryOption = 4
        if invAvailDay ≤ 1
            shippingMethod ← 4
        endif
    endif
endif
return(shippingMethod)
```

Table 10: Shipping time calculator pseudocode

Procedure: $ShipTimeCalc$
$Input: orderDay, orderHour, deliveryOption, shippingMethod$

```
if orderHour ≥ 13
    additionalDay ← 1
else
    additionalDay ← 0
endif
if deliveryOption = 1
    if shippingMethod = 1
        shippingTime ← (orderDay + additionalDay + 2) * 1440 + 840
    else if shippingMethod = 2
        shippingTime ← (orderDay + additionalDay + 4) * 1440 + 840
    else if shippingMethod = 3
        sihppingTime ← (orderDay + additionalDay + 5) * 1440 + 840
    else if shippingMethod = 4
        sihppingTime ← (orderDay + additionalDay + 6) * 1440 + 840
    endif
else if deliveryOption = 2
    if shippingMethod = 1
        shippingTime ← (orderDay + additionalDay) * 1440 + 840
    else if shippingMethod = 2
        shippingTime ← (orderDay + additionalDay + 2) * 1440 + 840
    else if shippingMethod = 3
        sihppingTime ← (orderDay + additionalDay + 3) * 1440 + 840
    else if shippingMethod = 4
        sihppingTime ← (orderDay + additionalDay + 4) * 1440 + 840
    endif
else if deliveryOption = 3
    if shippingMethod = 3
        sihppingTime ← (orderDay + additionalDay) * 1440 + 840
    else if shippingMethod = 4
        sihppingTime ← (orderDay + additionalDay + 1) * 1440 + 840
    endif
else if deliveryOption = 3
    if shippingMethod = 4
        sihppingTime ← (orderDay + additionalDay) * 1440 + 840
    endif
endif
return(shippingTime)
```

Table 11: Lock fulfillment decision pseudocode

Procedure: $LockFulfillmentDecision$
Input: $sysState, currTime$

```
/* Find all assignments that need to be locked and update system state accordingly */
sysState.timeOfMostRecentEvent ← currTime
For a = 1 to sysState.numAssignments
    if sysState.assignments[a].lockingTime = currTime
        fNum ← sysState.assignments[a].fcNum
        sNum ← sysState.assignments[a].skuNum
        qtyOH ← sysState.assignments[a].qtyFromOnHand
        qtyIO ← sysState.assignments[a].qtyFromInOrder
        increase sysState.inv[sNum][fNum].onHandAssignedLocked by qtyOH
        decrease sysState.inv[sNum][fNum].onHandAssignedOpen by qtyOH
        increase sysState.inv[sNum][fNum].onOrderAssignedLocked by qtyIO
        decrease sysState.inv[sNum][fNum].onOrderAssignedOpen by qtyIO
        sysState.assignments[a].locked ← true
    endif
endfor
/* Update event calendar and call other events */
put customer order shipment event in calendar
```

Table 12: Order shipment pseudocode

Procedure: $OrderShipment$
Input: $sysState, currTime$

```
/* Find all assignments that need to be shipped and remove them from system state */
for a = 1 to sysState.numAssignments
    if sysState.assignments[a].locked = "true"
        temp ← assignments[a]
        remove assignments[a] from list of current assignments in system state
        push temp into assignmentsToShip
    endif
endfor
numAssignmenetsToShip ← assignmentsToShip.size()

/* Update system state inventory information */
for a = 1 to numAssignmentsToShip
    fcN ← assignmenetsToShip[a].fcNum
    skuN ← assignmentsToShip[a].skuNum
    qtyOH ← assignmentsToShip[a].qtyFromOnHand
    qtyIO ← assignmentsToShip[a].qtyFromInOrder
    decrease sysState.inv[fcN][skuN].onHandAssignedLocked by qtyOH
    decrease sysState.inv[fcN][skuN].inOrderAssignedLocked by qtyOH
endfor

/* Find list of all customer orders that will be shipped at this time */
for r = 1 to sysState.numOrders
    for a = 1 to numAssignmentsToShip
        if assignmentsToShip[a].ordNum = r
            push r into ordersWithShipment
            exit for loop
        endif
    endfor
endfor
numOfOrdersWithShipment ← ordersWithShipment.size()

/* Find list of all FCs that will ship boxes to customers at this time */
for f = 1 to NumFcs
    for a = 1 to numAssignmentsToShip
        if assignmentsToShip[a].fcNumber = f
            push f into fcsWithShipment
            exit for loop
        endif
    endfor
endfor
numOfFcsWithShipment ← fcsWithShipment.size()

/* Calculate total weight and total number of boxes that will be shipped from each FC to each order using each shipping method */
for m = 1 to NumShipMethods
    for f = 1 to numOfFcsWithShipment
        for r = 1 to numOfOrdersWithShipment
            weightShipped[m][f][r] ← 0
            boxShipped[m][f][r] ← 0
            for a = 1 to numAssignmentsToShip
                mi ← assignmentsToShip[a].shippingMethod
                fi ← assignmentsToShip[a].fcNum
                ri ← assignmentsToShip[a].ordNum
                si ← assignmenetToShip[a].skuNumber
                qtyOH ← assignmenetsToShip[a].qtyFromOnHand
                qtyIO ← assignmenetsToShip[a].qtyFromInOrder
                increase weightShipped[mi][fi][ri] by sku[si].weight * (qtyOH + qtyIO)
                increase boxShipped[mi][fi][ri] by ceil(weightShipped[mi][fi][ri]/MaxBoxWeight)
            endfor
        endfor
    endfor
endfor

/*Calculate cost per mile and cost per pound for each order/FC/shipping method combination */
for m = 1 to NumShipMethods
    for f = 1 to numOfFcsWithShipment
        for r = 1 to numOfOrdersWithShipment
```

```
            for r = 1 to numOfOrdersWithShipment
                fi ← fcsWithShipment[f]
                ri ← ordersWithShipment[r]
                mi ← m
                shipZone ← Zone(fc[fi].fcLoc, sysState.orderQueue[ri].ordLoc, mi)
                costPerBox[m][f][r] ← ShipCostPerBoxCalc(mi, shipZone)
                costPerPound[m][f][r] ← ShipCostPerPoundCalc(mi, shipZone)
            endfor
        endfor
    endfor

    /*Calculate total cost of this shipment */
    costOfThisShipment ← 0
    for m = 1 to NumShipMethods
        for f = 1 to numOfFcsWithShipment
            for r = 1 to numOfOrdersWithShipment
                increase costOfThisShipment by costPerBox[m][f][r] * boxShipped[m][f][r]
                increase costOfThisShipment by costPerPound[m][f][r] * weightShipped[m][f][r]
            endfor
        endfor
    endfor

    /* Update statistical accumulators */
    increase totalShippingCost by costOfThisShipment
```

Table 13: FC inventory replenishment pseudocode

Procedure: *FCInventoryReplenishment*
Input: $targetFc, revCyc, currTime$

```
/* Find all SKUs in target FC with similar review cycle and calculate the replenishment quantity for each based on their max inventory level
and current available inventory */
for s = 1 to NumSkus
    if targetFc.invInfo[s].reviewCycle = revCyc
        q ← targetFc.invInfo[s].maxLevel − sysState.inv[targetFc.fcNumber][s].onHandUnAssigned
        t ← currTime + targetFc.invInfo[s].leadTime
        push s into skusToReplenish
        push q into qty
        push t into recTime
    endif
endfor

/* Update statistical accumulators */
for s = 1 to skusToReplenish.size()
    increase NumOfReplenishments[skusToReplenish[s]][targetFc.fcNumber] by 1
endfor

/* Update system state with information from this inventory replenishment */
sysState.timeOfMostRecentEvent ← currTime
for s = 1 to skusToReplenish.size()
    increase sysState.inv[targetFc.fcNumber][skusToReplenish[s]].inOrderUnAssigned by qty[s]
    sysState.inv[targetFc.fcNumber][skusToReplenish[s]].repTime ← recTime[s]
endfor

/* Update event calendar and call other events */
put the receiving inventory replenishment event in the calendar
put event for next replenishment order in the calendar
```

Table 14: Receive replenishment pseudocode

Procedure: *ReceiveReplenishment*
Input: $fcNum, skuNum, currTime$

```
/* Update system state inventory and assignments information based on the arriving replenishment */
sysState.timeOfMostRecentEvent ← currTime
increase sysState.inv[skuNum][fcNum].onHandUnassigned by sysState.inv[skuNum][fcNum].inOrderUnassigned
increase sysState.inv[skuNum][fcNum].onHandAssignedOpen by sysState.inv[skuNum][fcNum].inOrderAssignedOpen
increase sysState.inv[skuNum][fcNum].onHandAssignedLocked by sysState.inv[skuNum][fcNum].inOrderAssignedLocked
sysState.inv[targetSku][targetFc].onOrderUnassigned ← 0
sysState.inv[targetSku][targetFc].onOrderAssignedOpen ← 0
sysState.inv[targetSku][targetFc].onOrderAssignedLocked ← 0
sysState.inv[targetSku][targetFc].repTime ← 0
for a = 1 to sysState.numAssignments
    if sysState.assignments[a].fcNum = targetFC AND sysState.assignments[a].skuNum = targetSku
        increase sysState.assignments[a].qtyFromOnHand by sysState.assignments[a].qtyFromInOrder
        sysState.assignments[a].qtyFromInOrder ← 0
    endif
endfor
```

Table 15: Day calculator function pseudocode

$\boldsymbol{Procedure}$: Day $\boldsymbol{Input}$: $timeInMinute$
$day \leftarrow timeInMinute/1440$ return(day)

Table 16: Hour calculator function pseudocode

$\boldsymbol{Procedure}$: $Hour$ $\boldsymbol{Input}$: $timeInMinute$
$timeInHour \leftarrow (timeInMinute - 1440 * Day(timeInMinute))/60$ return$(timeInHour)$

Table 17: Other functions

Functions	
$unifDist(i, j)$	Returns a uniformly distributed random real value between i and j
$distance(origin, destination)$	Returns the Euclidean distance between $origin$ and $destination$
$ceil(number)$	Rounds $number$ to closest higher integer value

References

1. Acimovic, J., & Graves, S. C. (2015). Making better fulfillment decisions on the fly in an online retail environment. *Manufacturing & Service Operations Management, 17*(1), 34-51.

2. Acimovic, J., & Graves, S. C. (2017). Mitigating spillover in online retailing via replenishment. *Manufacturing & Service Operations Management, 19*(3), 419-436.

3. Agatz, N. A., Fleischmann, M., & Van Nunen, J. A. (2008). E-fulfillment and multi-channel distribution–A review. *European Journal of Operational Research, 187*(2), 339-356.

4. Akturk, M. S., Ketzenberg, M., & Heim, G. R. (2018). Assessing impacts of introducing ship-to-store service on sales and returns in omnichannel retailing: A data analytics study. *Journal of Operations Management, 61*, 15-45.

5. Alimeling, J., & Hammer, W. P. (1999). PLECS-piece-wise linear electrical circuit simulation for Simulink. *Paper presented at the Proceedings of the IEEE 1999 International Conference on Power Electronics and Drive Systems*. PEDS'99 (Cat. No. 99TH8475).

6. Ansaripour, A., & Trafalis, T. B. (2013). A Robust multicriteria optimization model for city logistic terminal locations. *In IIE Annual Conference. Proceedings (p. 2801). Institute of Industrial and Systems Engineers.*

7. Ardjmand, E., Bajgiran, O. S., Rahman, S., Weckman, G. R., & Young II, W. A. (2018). A multi-objective model for order cartonization and fulfillment center assignment in the e-tail/retail industry. *Transportation Research Part E: Logistics and Transportation Review, 115*, 16-34.

8. Azadivar, F., & Wang, J. (2000). Facility layout optimization using simulation and genetic algorithms. *International Journal of Production Research, 38*(17), 4369-4383.

9. Banks, J. (2005). *Discrete event system simulation*: Pearson Education India.

10. Becerril-Arreola, R., Leng, M., & Parlar, M. (2013). Online retailers' promotional pricing, free-shipping threshold, and inventory decisions: A simulation-based analysis. *European Journal of Operational Research, 230*(2), 272-283.

11. Beiranvand, V., Hare, W., & Lucet, Y. (2017). Best practices for comparing optimization algorithms. *Optimization and Engineering, 18*(4), 815-848.

12. Bell, D. R., Gallino, S., & Moreno, A. (2013). Inventory showrooms and customer migration in omni-channel retail: The effect of product information. *Available at SSRN, 2370535*.

13. Bendoly, E. (2004). Integrated inventory pooling for firms servicing both on-line and store demand. *Computers & Operations Research, 31*(9), 1465-1480.

14. Bendoly, E., Blocher, D., Bretthauer, K. M., & Venkataramanan, M. (2007). Service and cost benefits through clicks-and-mortar integration: Implications for the centralization/decentralization debate. *European Journal of Operational Research, 180*(1), 426-442.

15. Bhargava, R., Levalle, R. R., & Nof, S. Y. (2016). A best-matching protocol for order fulfillment in re-configurable supply networks. *Computers in Industry, 82*, 160-169.

16. Boyer, K. K., Hult, G. T., & Frohlich, M. (2003). An exploratory analysis of extended grocery supply chain operations and home delivery. *Integrated Manufacturing Systems*.

17. Boyer, K. K., & Hult, G. T. M. (2005). Extending the supply chain: integrating operations and marketing in the online grocery industry. *Journal of Operations Management, 23*(6), 642-661.

18. Brynjolfsson, E., & Smith, M. D. (2000). Frictionless commerce? A comparison of Internet and conventional retailers. *Management Science, 46*(4), 563-585.

19. Chen, C., & Pan, S. (2016). Using the Crowd of Taxis to Last Mile Delivery in E-Commerce: a methodological research. In *Service Orientation in Holonic and Multi-Agent Manufacturing* (pp. 61-70): Springer.

20. Chen, L., Jin, R., Qin, H., Simchi-Levi, D., & Zhang, Z. (2019). Distributionally Robust Omnichannel Stocking Decisions in Quick Fulfillment Systems. *Available at SSRN 3383881*.

21. Constantinides, E. (2004). Influencing the online consumer's behavior: The Web experience. *Internet Research*.

22. Dabholkar, P. A., & Bagozzi, R. P. (2002). An attitudinal model of technology-based self-service: moderating effects of consumer traits and situational factors. *Journal of the Academy of Marketing Science, 30*(3), 184-201.

23. Daduna, J. R., & Lenz, B. (2005). Online shopping and changes in mobility. In *Distribution Logistics* (pp. 65-84): Springer.

24. De Koster, R. B. (2002). Distribution structures for food home shopping. *International Journal of Physical Distribution & Logistics Management, 32*(5), 362-380.

25. De Koster, R. B. (2003). Distribution strategies for online retailers. *IEEE Transactions on Engineering Management, 50*(4), 448-457.

26. Devari, A., Nikolaev, A. G., & He, Q. (2017). Crowdsourcing the last mile delivery of online orders by exploiting the social networks of retail store customers. *Transportation Research Part E: Logistics and Transportation Review, 105*, 105-122.

27. Eastlick, M. A., & Lotz, S. (1999). Profiling potential adopters and non-adopters of an interactive electronic shopping medium. *International Journal of Retail & Distribution Management*.

28. Elliot, S., & Fowell, S. (2000). Expectations versus reality: a snapshot of consumer experiences with Internet retailing. *International Journal of Information Management, 20*(5), 323-336.

29. Esper, T. L., Jensen, T. D., Turnipseed, F. L., & Burton, S. (2003). The last mile: an examination of effects of online retail delivery strategies on consumers. *Journal of Business Logistics, 24*(2), 177-203.

30. Favier, J., & Bouquet, M. (2006). Europe's e Commerce Forecast: 2006 to 2011

31. Gao, F., & Su, X. (2017a). Omnichannel retail operations with buy-online-and-pick-up-in-store. *Management Science, 63*(8), 2478-2492.

32. Gao, F., & Su, X. (2017b). Online and offline information for omnichannel retailing. *Manufacturing & Service Operations Management, 19*(1), 84-98.

33. Geng, J., & Li, C. (2019). Empirical Research on the Spatial Distribution and Determinants of Regional E-Commerce in China: Evidence from Chinese Provinces. *Emerging Markets Finance and Trade*.

34. Gevaers, R., Van de Voorde, E., & Vanelslander, T. (2009). Characteristics of innovations in last-mile logistics-using best practices, case studies and making the link with green and sustainable logistics. *Association for European Transport and Contributors*.

35. Gosavi, A. (2015). *Simulation-Based Optimization*. Berlin: Springer.

36. Govindarajan, A., Sinha, A., & Uichanco, J. (2018). Joint inventory and fulfillment decisions for omnichannel retail networks. *Ross School of Business Paper*(1341).

37. Grewal, D., Iyer, G. R., & Levy, M. (2004). Internet retailing: enablers, limiters and market consequences. *Journal of Business Research, 57*(7), 703-713.

38. Hare, W., Loeppky, J., & Xie, S. (2018). Methods to compare expensive stochastic optimization algorithms with random restarts. *Journal of Global Optimization*, 72(4), 781-801.

39. Hillstrom, K. E. (1977). A simulation test approach to the evaluation of nonlinear optimization algorithms. *ACM Transactions on Mathematical Software (TOMS)*, 3(4), 305-315.

40. Hovelaque, V., Soler, L. G., & Hafsa, S. (2007). Supply chain organization and e-commerce: a model to analyze store-picking, warehouse-picking and drop-shipping. *4OR, 5*(2), 143-155.

41. Huang, D., Zhao, Q. H., & Fan, C. C. (2010). Simulation-based optimization of inventory model with products substitution. *In Innovative Quick Response Programs in Logistics and Supply Chain Management* (pp. 297-312). Springer, Berlin, Heidelberg.

42. Hübner, A., Holzapfel, A., & Kuhn, H. (2015). Operations management in multi-channel retailing: an exploratory study. *Operations Management Research, 8*(3-4), 84-100.

43. Jasin, S., & Sinha, A. (2015). An LP-based correlated rounding scheme for multi-item ecommerce order fulfillment. *Operations Research, 63*(6), 1336-1351.

44. Jung, S., & Kim, H. (2017). Analysis of amazon prime air uav delivery service. *Journal of Knowledge Information Technology and Systems, 12*(2), 253-266.

45. Kacen, J. J., Hess, J. D., & Chiang, W.-y. K. (2013). Bricks or clicks? Consumer attitudes toward traditional stores and online stores. *Global Economics and Management Review, 18*(1), 12-21.

46. Kaggle. (2020). Brazilian E-Commerce Public Dataset by Olist. Retrieved from https://www.kaggle.com/olistbr

47. Keeney, R. L. (1999). The value of Internet commerce to the customer. *Management Science, 45*(4), 533-542.

48. Keramydas, C., Mallidis, I., Dekker, R., Vlachos, D., & Iakovou, E. (2017). Cost and environmental trade-offs in supply chain network design and planning: the merit of a simulation-based approach. *Journal of Simulation, 11*(1), 20-29.

49. Khan, W. A., Yousaf, S., Mian, N. A., & Nawaz, Z. (2013). E-commerce in Pakistan: Growth potentials and e-payment solutions. *Paper Presented at the 2013 11th International Conference on Frontiers of Information Technology*.

50. Kämäräinen, V., & Punakivi, M. (2002). Developing cost-effective operations for the e-grocery supply chain. *International Journal of Logistics, 5*(3), 285-298.

51. Lee, M. K., & Turban, E. (2001). A trust model for consumer internet shopping. *International Journal of Electronic Commerce, 6*(1), 75-91.

52. Lei, Y., Jasin, S., & Sinha, A. (2018). Joint dynamic pricing and order fulfillment for e-commerce retailers. *Manufacturing & Service Operations Management, 20*(2), 269-284.

53. Li, S., & Jia, S. (2019). A Benders decomposition algorithm for the order fulfilment problem of an e-tailer with a self-owned logistics system. *Transportation Research Part E: Logistics and Transportation Review, 122*, 463-480.

54. Ma, S. (2017). Fast or free shipping options in online and Omni-channel retail? The mediating role of uncertainty on satisfaction and purchase intentions. *The International Journal of Logistics Management*.

55. Ma, S., Jemai, Z., Sahin, E., & Dallery, Y. (2017). The news-vendor problem with drop-shipping and resalable returns. *International Journal of Production Research, 55*(22), 6547-6571.

56. Mahar, S., Salzarulo, P. A., & Wright, P. D. (2012). Using online pickup site inclusion policies to manage demand in retail/E-tail organizations. *Computers & Operations Research, 39*(5), 991-999.

57. Mahar, S., & Wright, P. D. (2009). The value of postponing online fulfillment decisions in multi-channel retail/e-tail organizations. *Computers & Operations Research, 36*(11), 3061-3072.

58. Malykhina, E. (2005). Retailers take stock. *Information Week* (1025), 20-22.

59. Marbach, P., & Tsitsiklis, J. N. (2001). Simulation-based optimization of Markov reward processes. *IEEE Transactions on Automatic Control,* 46(2), 191-209.

60. Marques, A. F., de Sousa, J. P., & Rönnqvist, M. (2014). Combining optimization and simulation tools for short-term planning of forest operations. *Scandinavian Journal of Forest Research, 29*(sup1), 166-177.

61. Mehmann, J., Frehe, V., & Teuteberg, F. (2015). Crowd logistics− a literature review and maturity model. Paper presented at the Innovations and Strategies for Logistics and Supply Chains: Technologies, Business Models and Risk Management. *Proceedings of the Hamburg International Conference of Logistics (HICL)*, Vol. 20.

62. Mele, F. D., Guillen, G., Espuna, A., & Puigjaner, L. (2006). A simulation-based optimization framework for parameter optimization of supply-chain networks. *Industrial & Engineering Chemistry Research, 45*(9), 3133-3148.

63. Mena, C., Bourlakis, M., Hübner, A., Wollenburg, J., & Holzapfel, A. (2016). Retail logistics in the transition from multi-channel to omni-channel. *International Journal of Physical Distribution & Logistics Management*.

64. Michelis, D. (2010). *Social-Media-Handbuch: Theorien, Methoden, Modelle*: Nomos-Verlag-Ges., Ed. Fischer.

65. Nguyen, A. T., Reiter, S., & Rigo, P. (2014). A review on simulation-based optimization methods applied to building performance analysis. *Applied Energy*, 113, 1043-1058.

66. O'Grady, M., & D'Costa, V. (2019). *Forrester Analytics: Online Retail Forecast, 2019 To 2024 (US)*. Retrieved from https://www.forrester.com/report

67. Ortis, I., & Casoli, A. (2009). Technology selection: IDC retail insights guide to enabling immersive shopping experiences. *IDC Retail Insights Report*.

68. Parker, R., & Hand, L. (2009). Satisfying the omnichannel consumers whenever and wherever they shop. *IDC Retail Insights Report*.

69. Petering, MEH. (2015). Comparison of algorithms for the unending real-time traveling repairperson problem by fully embedding them in a simulation model. *Odysseus 2015 International Workshop on Freight and Transportation Systems*

70. Petering, MEH. (2018). Evaluating exact vs. rule-based algorithms for the unending real-time traveling repairperson problem under true simulated operating conditions. *Odysseus 2018 International Workshop on Freight and Transportation Systems*

71. Pflug, G. C. (2012). *Optimization of Stochastic Models: the Interface Between Simulation and Optimization (Vol. 373)*. Springer Science & Business Media.

72. Piotrowicz, W., & Cuthbertson, R. (2014). Introduction to the special issue information technology in retail: Toward omnichannel retailing. *International Journal of Electronic Commerce, 18*(4), 5-16.

73. Rajendran, S., Ansaripour, A., Kris Srinivasan, M., & Chandra, M. J. (2019). Stochastic goal programming approach to determine the side effects to be labeled on pharmaceutical drugs. *IISE Transactions on Healthcare Systems Engineering, 9*(1), 83-94.

74. Rambaran, S. (2016). *The Effect of Omni-Distribution Systems in Managing Demand Order Fulfilment Frequencies: an Apparel Retailer.*

75. Sahney, S. (2008). Critical success factors in online retail–an application of quality function deployment and interpretive structural modeling. *International Journal of Business and Information, 3*(1).

76. Shim, S., Eastlick, M. A., Lotz, S. L., & Warrington, P. (2001). An online pre-purchase intentions model: the role of intention to search: best overall paper award—The Sixth Triennial AMS/ACRA Retailing Conference, 2000☆. *Journal of Retailing, 77*(3), 397-416.

77. Sivakumar, A. I. (1999) Optimization of a cycle time and utilization in semiconductor test manufacturing using simulation based, on-line, near-real-time scheduling system. *In Proceedings of the 31st Conference on Winter Simulation: Simulation---a Bridge to the Future-Volume 1* (pp. 727-735).

78. Slabinac, M. (2015). Innovative solutions for a "Last-Mile" delivery–a European experience. *Business Logistics in Modern Management*.

79. Soars, B. (2003). Showing how it's done. *Grocer, 226*(7603), 30.

80. Statista. (2019). *Retail E-commerce Sales Worldwide from 2014 to 2023*. Retrieved from https://www.statista.com/statistics

81. Tetteh, A., & Xu, Q. (2014). Supply chain distribution networks: Single-, dual-, & omni-channel. *Interdisciplinary Journal of Research in Business, 3*(9), 63-73.

82. Torkzadeh, G., & Dhillon, G. (2002). Measuring factors that influence the success of Internet commerce. *Information Systems Research, 13*(2), 187-204.

83. UPS Shipping Rates (2019), Retrieved from https://wwwapps.ups.com/ctc

84. U.S. Department of Commerce, (2019). *Quarterly Retail E-commerce Sales, 3rd quarter 2019*. Retrieved from https://www.census.gov/retail

85. Verhoef, P. C., Kannan, P. K., & Inman, J. J. (2015). From multi-channel retailing to omni-channel retailing: introduction to the special issue on multi-channel retailing. *Journal of retailing, 91*(2), 174-181.

86. Wolfinbarger, M., & Gilly, M. C. (2001). Shopping online for freedom, control, and fun. *California Management Review, 43*(2), 34-55.

87. Xiao, Y., Chen, F. Y., & Chen, J. (2009). Optimal inventory and dynamic admission policies for a retailer of seasonal products with affiliate programs and drop-shipping. *Naval Research Logistics (NRL), 56*(4), 300-317.

88. Xu, P. J., Allgor, R., & Graves, S. C. (2009). Benefits of reevaluating real-time order fulfillment decisions. *Manufacturing & Service Operations Management, 11*(2), 340-355.

89. Yen, B., Hu, P. J.-H., & Wang, M. (2007). Toward an analytical approach for effective Web site design: A framework for modeling, evaluation and enhancement. *Electronic Commerce Research and Applications, 6*(2), 159-170.

90. Yoon, S.-J. (2002). The antecedents and consequences of trust in online-purchase decisions. *Journal of Interactive Marketing, 16*(2), 47-63.

91. Zeng, Q., & Yang, Z. (2009). Integrating simulation and optimization to schedule loading operations in container terminals. *Computers & Operations Research, 36*(6), 1935-1944.

CURRICULUM VITAE

Amir Kalantari

Place of birth: Tehran, Iran

EDUCATION

Ph.D. University of Wisconsin, Milwaukee August 2020

Major: Industrial Engineering – Operations Research

Minor: Computer Science

Dissertation title: Reevaluating Order Fulfillment Decisions for E-tailers Under True Simulated Operating Conditions

Advisor: Dr. Matthew E. H. Petering

MSc. University of Wisconsin, Milwaukee May 2013

Major: Industrial Engineering – Operations Research

Thesis title: Facility Location Selection for Global Manufacturing

Advisor: Dr. Hamid Seifoddini

BSc. Sharif University of Technology, Tehran, Iran May 2011

Major: Industrial Engineering – Industrial Engineering

TEACHING EXPERIENCE

Primary Instructor, University of Wisconsin – Milwaukee Jan 2013 – May 2017

Course: Engineering Drawing & Computer Aided Design (70 freshman and sophomore students)

Teaching Assistant, University of Wisconsin – Milwaukee Sept 2011 – Dec 2012

Course 1: Introduction to AutoCAD (30 freshman and sophomore students)

Course 2: Simulation Methodology (30 junior and senior students)

RESEARCH EXPERIENCE

Analysis of UWM Off-Campus Transit Service — Jan 2014
University of Wisconsin – Milwaukee

PRESENTATIONS

A Kalantari, M.E.H. Petering, Reevaluating Order Fulfillment Decisions for E-tailers Under True Simulated Operating Conditions, INFORMS Annual Meeting, Nashville, TN, USA, October 2017

M.E.H Petering, **A Kalantari,** A Ross, H Seifoddini, A Mixed Integer Program for Solving the Art Gallery Problem, INFORMS Annual Meeting, Minneapolis, MN, USA, October 2013

GRADUATE INTERNSHIPS

Engineering Intern, Information Technology and BI — Jan 2015 – May 2015
Custom Service Plastics, Lake Geneva – Wisconsin

Engineering Intern, Enterprise Resource Planning — May 2014 – Aug 2014
Custom Service Plastics, Lake Geneva – Wisconsin

Engineering Intern, Lean Manufacturing and Six Sigma — May 2012 – Aug 2012
Custom Service Plastics, Lake Geneva – Wisconsin

HONORS AND AWARDS

- Chancellor's Award (2011-2017)
  Department of Industrial & Manufacturing Engineering, University of Wisconsin – Milwaukee

- Graduate School Travel Award (2012)
  University of Wisconsin – Milwaukee

- Ranked among the top 1% in Iran's university entrance exam (2006)

- Received direct admission to the highly competitive graduate school at Sharif University of Technology because of exceptional performance in the BSc. program (2011)