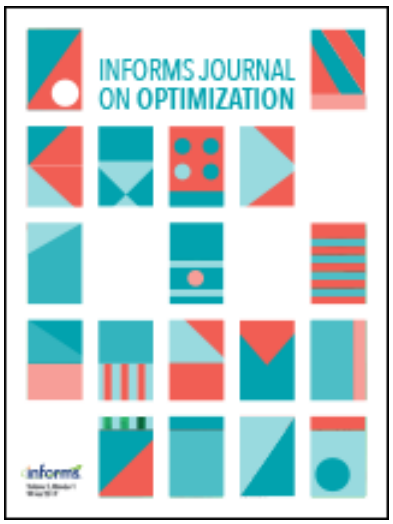


INFORMS Journal on Optimization

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Data-Driven Modeling and Optimization of the Order Consolidation Problem in E-Warehousing

Fatma Gzara, Samir Elhedhli, Ugur Yildiz, Gohram Baloch

To cite this article:
Fatma Gzara, Samir Elhedhli, Ugur Yildiz, Gohram Baloch (2020) Data-Driven Modeling and Optimization of the Order Consolidation Problem in E-Warehousing. INFORMS Journal on Optimization 2(4):273-296. https://doi.org/10.1287/ijoo.2019.0039

http://pubsonline.informs.org/journal/ijoo

INFORMS JOURNAL ON OPTIMIZATION
Vol. 2, No. 4, Fall 2020, pp. 273–296
ISSN 2575-1484 (print), ISSN 2575-1492 (online)

Data-Driven Modeling and Optimization of the Order Consolidation Problem in E-Warehousing

Fatma Gzara,[a] Samir Elhedhli,[a] Ugur Yildiz,[a] Gohram Baloch[a]

[a] Department of Management Sciences, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada

Contact: fgzara@uwaterloo.ca, https://orcid.org/0000-0003-2893-8597 (FG); elhedhli@uwaterloo.ca, https://orcid.org/0000-0003-1922-8286 (SE); uyildiz@uwaterloo.ca, https://orcid.org/0000-0003-0710-5940 (UY); ggohram@uwaterloo.ca, https://orcid.org/0000-0002-8267-8776 (GB)

Received: July 22, 2019
Revised: March 31, 2020
Accepted: May 19, 2020
Published Online in Articles in Advance: October 2, 2020

https://doi.org/10.1287/ijoo.2019.0039

Abstract. We analyze data emanating from a major e-commerce warehouse and provided by a third-party warehouse logistics management company to replicate flow diagrams, assess order fulfillment efficiency, identify bottlenecks, and suggest improvement strategies. Without access to actual layouts and process-flow diagrams and purely based on data, we are able to describe the processes in detail and prescribe changes. By investigating the characteristics of orders, the wave-sorting operation, and the order-preparation process, we find that products from different orders are picked in batches for efficiency. Similar products are picked in small containers called totes. Totes are then stored in a buffer area and routed to be emptied of their contents at induction lines. Orders are then consolidated at the put wall, where each order is accumulated in a cubby. This order consolidation process depends on the sequence in which totes are processed and has a huge impact on order-completion time. We, therefore, present a generalization of the parallel machine–scheduling problem that we call the order consolidation problem to determine the tote-processing sequence that minimizes total order completion time. We provide mathematical formulations and devise heuristic and exact solution methods. We propose a fast simulated annealing metaheuristic and a branch-and-price approach in which the subproblems are variants of the single machine-scheduling problem and are solved using dynamic programming. We also devise a new branching rule, compare it against the literature, and test it on randomly generated and industry data. Applied to the data and the warehouse under study, optimizing the order consolidation is found to decrease the completion time of 75.66% of orders and achieve average improvements of up to 28.77% in order consolidation time and 21.92% in cubby usage.

Funding: This work was supported by Ontario Centres of Excellence [Grant VIPII-23421].

Keywords: e-commerce warehouse • order completion • order consolidation • parallel machine scheduling • branch and price • dynamic programming

1. Introduction

Warehousing operations have a significant impact on logistics efficiency and cost. This is particularly true for e-business, in which customer satisfaction is highly correlated with the order-fulfillment process. Today, intelligent automated systems are the backbone of warehousing. As these systems are taking over manual operations, data are being collected at every step through which an order goes. Data often carries with it valuable information that may be used to improve operations and increase efficiency. It is with this mindset that a data analytics collaboration with a global warehousing and logistics automation company was established, culminating in the current work.

Data analytics may help analyze process flow, predict anomalies, hint at possible courses of action, and validate any proposed changes. But, most importantly, carefully designed data analytics tools may identify weaknesses in large systems in which manual inspection is overshadowed by large production volumes. E-commerce warehouses are an example of such systems. Online orders with varying content and priorities have to be fulfilled in record time to meet customer expectations. Although the demand for individual products may be low, orders contain multiple products, typically of high variety. Not only does an e-warehouse require large physical space, but it also requires an efficient order-fulfillment process in terms of product picking, consolidating, and packaging. Order picking has been identified as the costliest and the most labor-intensive activity in a warehouse. Its cost could reach as much as 55% of the total warehousing cost (De Koster et al. 2007). Several policies, such as batch and wave picking, are commonly used. A batch or a wave is

essentially a group of orders that is fulfilled simultaneously. Items from all orders within a wave or a batch are aggregated into a single pool for simultaneous picking. The pool is then split into multiple pick lists, and each list is handed to an operator. In both policies, products from different orders are grouped in lists for picking, but the policies differ in one important aspect. In wave picking, an order may be split over multiple totes. Warehouse managers decide on the size of the wave; the start and end times of the different operations in the warehouse, including when to pick orders for a wave (wave picking); the number of waves to have; and the assignment of operators to different operations—all as a function of the order arrivals, the buffer space available, and the number of operators. The goal is to keep a constant flow of fulfilled orders and eventually achieve the target output for the day.

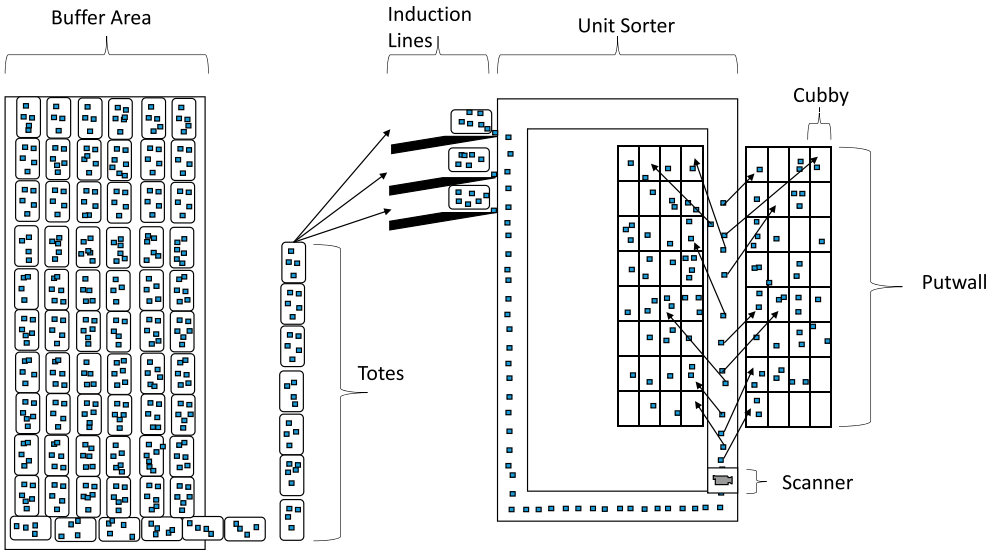
The picking process is done manually as an operator is handed a pick list and a tote and goes through the aisles of the warehouse to pick the corresponding products. After picking, totes are stored temporarily in a buffer area. Totes are then sorted and emptied, and orders are consolidated for subsequent shipping. The current work uses data from an e-commerce warehouse that adopts a wave-picking policy. Totes, with information on the products they carry, are routed to a unit sorter through multiple induction lines. The unit sorter is a conveyor equipped with scanners that scan and route products to their corresponding packaging stations, called cubbies. A cubby is similar to an accumulation lane, in which products from the same order are gathered. This order consolidation process continues until the last product in an order is put. Once an order is complete, it is packaged and moved to the shipping area. Figure 1 shows a high-level process-flow diagram of the warehouse that is entirely deduced from the data as described in Section 3.

The goal of the current work is to analyze raw data to understand the warehousing operation under investigation and the different processes it involves and to identify areas for improvement. We then propose solutions and validate them using the data. We emphasize that no prior information on the warehouse is available, including the nature of the products served, flow diagrams, layout of the warehouse, or the equipment used. The data were provided by a third party that has access through the automated warehousing system maintained at the warehouse.

We start with an in-depth descriptive study to construct a flow diagram. Three main process flows are identified, and the importance of each as well as the relationships among them are examined. We find that consolidation time, which is the completion time of an order at the unit sorter, has a significant impact on the whole operation. In investigating the problem, we identify a general class of parallel machine–scheduling (PMS) problems that we call the order consolidation problem (OCP). We define the OCP, present a nonlinear mixed-integer programming (NMIP) formulation, and develop heuristic and exact solution methodologies based on simulated annealing and branch and price. The contributions of the current work are threefold:

- We carry out a full data-driven study for e-warehousing with descriptive analytics and data-driven process improvement.
- We identify a new research problem, the order consolidation problem, and provide mathematical formulations as well as heuristic and exact solution approaches.

Figure 1. (Color online) Process Flow Diagram Deduced from the Data



• We provide a branch-and-price/column-generation solution methodology for the OCP in which the subproblem is a generalized single machine-scheduling problem. We provide a dynamic programming algorithm to solve it and devise a new branching rule.

The rest of the paper is structured as follows. In Section 2, we review the literature for both warehouse-management systems and parallel machine–scheduling problems. In Section 3, we describe the process used to derive process flows and layout from the data. In Section 4, we derive and analyze process times and identify the order-fulfillment operation as an area of improvement. In Section 5, we introduce the order consolidation problem and provide a formulation. In Section 6, we develop a branch-and-price approach and a simulated annealing algorithm to solve the OCP. In Section 7, we conduct an extensive computational study to test the algorithms and compare them with the real system. Finally, in Section 8, we conclude and suggest future research directions.

2. Literature Review

Warehouse management is a well-studied research area with several research topics, such as picking strategies, routing policies, and accumulation/sorting (A/S) system design. Because the current work focuses on the order consolidation process, we review the literature on A/S systems and order batching and recommend reading the reviews of Gu et al. (2007) and De Koster et al. (2007) for the picking policies.

Order batching aims to increase efficiency by pooling orders together in a batch to be picked by a single picker. The goal is to reduce travel time by picking similarly stored products simultaneously (Henn et al. 2012). The problem was studied by Gademann et al. (2001), Hsieh and Huang (2011), and Grosse et al. (2014), who propose heuristic and exact methods to solve it. Order batching is followed by routing, that is, the sequence in which pickers pick their assigned products. Ratliff and Rosenthal (1983) and Roodbergen and Koster (2001), for example, use a special case of the traveling salesman problem to optimize routing in a rectangular warehouse. Most of the research focuses on developing heuristics for the routing of pickers (Hwang et al. 2004, Petersen and Aase 2004, Theys et al. 2010). Recently, other issues, such as picker blocking and congestion, have been included in routing pickers (Hong et al. 2012, Pan and Wu 2012, Chen et al. 2013).

Order A/S systems are used to consolidate orders in wave picking. An automatic A/S system is usually composed of a closed-loop conveyor with automatic divert mechanisms, a scanner, and accumulation lanes to which the products are diverted (Van den Berg and Zijm 1999). In the warehouse under study, there are cubbies instead of accumulation lanes. A sensor scans products that are inducted to the unit sorter. Products corresponding to the same order are then automatically diverted to the corresponding cubby. Under the assumption of a single order-lane assignment, Bozer and Sharp (1985) examine the advantages of recirculation to avoid lane blocking in an A/S system when an accumulation lane is full. Bozer and Sharp (1985) study a system that processes a relatively small number of large orders and in which each sorting lane is typically dedicated to one order. Simulation is used to analyze the dependence of system throughput on factors such as induction capacity, number of lanes, and length of lanes. Considering A/S systems in which multiple orders may be assigned to one lane, Bozer et al. (1988) and Johnson (1997) suggest that assigning lanes to orders at the induction step is better than any predetermined assignment rule. Meller (1997) considers a two-level order A/S system and develops an algorithm based on decomposing a favorably structured mathematical program to optimally assign orders to lanes based on the arrival sequence of products to the sorting system. Russell and Meller (2003) present a prescriptive model, which examines several system parameter combinations to help decide whether to automate the sorting system.

The problem we identify to improve the order consolidation process is closely related to the PMS literature. Given a set of jobs $J = \{1, 2, \ldots, n\}$ with positive processing times p_j and a set of identical machines $I = \{1, 2, \ldots, m\}$, the PMS problem seeks to find a nonpreemptive schedule such that at most one job is processed on a machine at any given time. The objective is to minimize the total processing time. The first work on PMS problems started at the end of the 1950s and the beginning of the 1960s. For example, McNaughton (1959) and Graham (1966) are the first to provide scheduling rules and analyze their performance for identical parallel machine scheduling with preemption and partial order, respectively. Graham (1966), in particular, provides one of the first worst-case analyses.

PMS problems have been studied for several objectives, machine types, job types, and constraints. However, with the technological developments in production environments, new job, machine, or performance criteria may be required to tackle recent problems. Table 1 provides a summary in terms of machine type: identical (P), uniform (Q), unrelated (R); job type: single (Sgl), batch (Bch), split (Spl), multiple-order jobs (MOJ); and solution methodology: exact (Exct), heuristic (Heur). According to the table, most of the work focuses on single-order jobs with only two studies on multiple-order jobs. Apart from Boysen et al. (2019), who study a

Table 1. Overview of Literature on Parallel-Machine Scheduling

	Machine type			Job type				Solution		
	P	Q	R	Sgl	Bch	Spt	MOJ	Exct	Heur	Criteria
Belouadah and Potts (1994)	✓			✓				✓		TWCT
De et al. (1994)	✓			✓				✓		Other
Schutten and Leussink (1996)	✓			✓				✓		Lmax
Serafini (1996)	✓					✓		✓		MaxWT
van Den Akker et al. (1999)	✓			✓				✓		TWCT,TWU
Chen and Powell (1999a)	✓			✓				✓		TWE+TWT
Chen and Powell (1999b)	✓			✓				✓		TWCT,TWU
Balakrishnan et al. (1999)		✓		✓				✓		TWT,TWE
Min and Cheng (1999)	✓			✓					✓	Cmax
Xing and Zhang (2000)	✓					✓			✓	Cmax
Weng et al. (2001)			✓	✓					✓	TWCT
Anagnostopoulos and Rabadi (2002)			✓	✓					✓	Cmax
Kim et al. (2002)			✓	✓					✓	TT
Mokotoff and Chrétienne (2002)			✓	✓				✓		Cmax
Chen and Powell (2003)	✓			✓				✓		TWCT,TWU
Lin and Jeng (2004)	✓				✓			✓		Lmax,TU
Logendran and Subur (2004)			✓			✓			✓	TWT
Mokotoff (2004)	✓			✓				✓		Cmax
Mönch et al. (2005)	✓				✓				✓	TWT
Logendran et al. (2007)			✓	✓					✓	TWT
Jampani and Mason (2008)	✓						✓		✓	TWCT
Shim and Kim (2008)	✓					✓		✓		TT
Jia and Mason (2009)	✓						✓		✓	TWCT
Tavakkoli-Moghaddam et al. (2009)			✓	✓					✓	TU,TCT
Fanjul-Peyro and Ruiz (2010)			✓	✓					✓	Cmax
Vallada and Ruiz (2011)			✓	✓					✓	Cmax
Lee et al. (2013)			✓	✓					✓	TT
Kaplan and Rabadi (2013)	✓			✓					✓	TWT
Chen (2015)			✓	✓					✓	TWCT
Liaw (2016)	✓			✓				✓		TT
Bülbül and Şen (2017)			✓					✓		TWCT
Kowalczyk and Leus (2017)	✓							✓		Cmax
Villa et al. (2018)			✓	✓					✓	Cmax
Rauchecker and Schryen (2019)			✓	✓				✓		TWCT

Note. TCT, total completion time; TWCT, total weighted completion time; Cmax, maximum completion time; Lmax, maximum lateness; TWT, total weighted tardiness; TWU, total weighted number of tardy jobs; TWE,, total weighted earliness; TU, number of tardy jobs; MaxWT, maximum weighted tardiness.

single machine-scheduling problem with multiple jobs and multiple orders, there is very limited research considering multiple orders. In that respect, we generalize the problem studied in Boysen et al. (2019) as we introduce the multiple-order job, multiple machine–scheduling problem, which we call the order consolidation problem. None of the existing methods for the multiple machine–scheduling problem apply to this problem. Hence, to the best of our knowledge, we are the first to develop an exact solution approach to the problem.

The OCP minimizes the total completion time of orders, which is known to be an easy criterion for the case of single jobs; however, because of the job definition, the OCP is shown to be NP-hard by reducing it to the parallel machine–scheduling problem with total weighted completion time (TWCT). The latter is studied by Chan et al. (1998), who show that, under some assumptions, the linear relaxation of the set-partitioning formulation is optimal. van Den Akker et al. (1999) solve the same problem using column generation. Chen and Powell (1999b) propose a branch and price for the parallel machine–scheduling problem with TWCT, and Chen and Powell (1999a) study the case with the objective of minimizing total weighted earliness and tardiness. Several variants with the single-job assumption are studied, for example, for the case of unrelated machines (Chen 2015, Lin and Ying 2015, Bülbül and Şen 2017) and setup times (Weng et al. 2001, Chen and Powell 2003, Rauchecker and Schryen 2019). Mason and Chen (2010) study multiple-order jobs with total completion time and TWCT for the single-machine problem, whereas Jampani and Mason (2008) and Jia and Mason (2009) consider the parallel machine version.

Several exact solution approaches are presented for various PMS problems, such as dynamic programming (De et al. 1994), branch and bound (Schutten and Leussink 1996), cutting-plane methods (Mokotoff and Chrétienne 2002, Mokotoff 2004), and column generation and branch and price (Chen and Powell 1999b. van Den Akker et al. 1999. Jampani and Mason 2008). Metaheuristic approaches include genetic algorithms (Min and Cheng 1999, Tavakkoli-Moghaddam et al. 2009, Vallada and Ruiz 2011), simulated annealing (Radhakrishnan and Ventura 2000, Anagnostopoulos and Rabadi 2002, Kim et al. 2002, Kaplan and Rabadi 2013, Lin and Ying 2015), and tabu search (Logendran and Subur 2004, Logendran et al. 2007, Lee et al. 2013)

3. Building Process Flow and Layout from Data

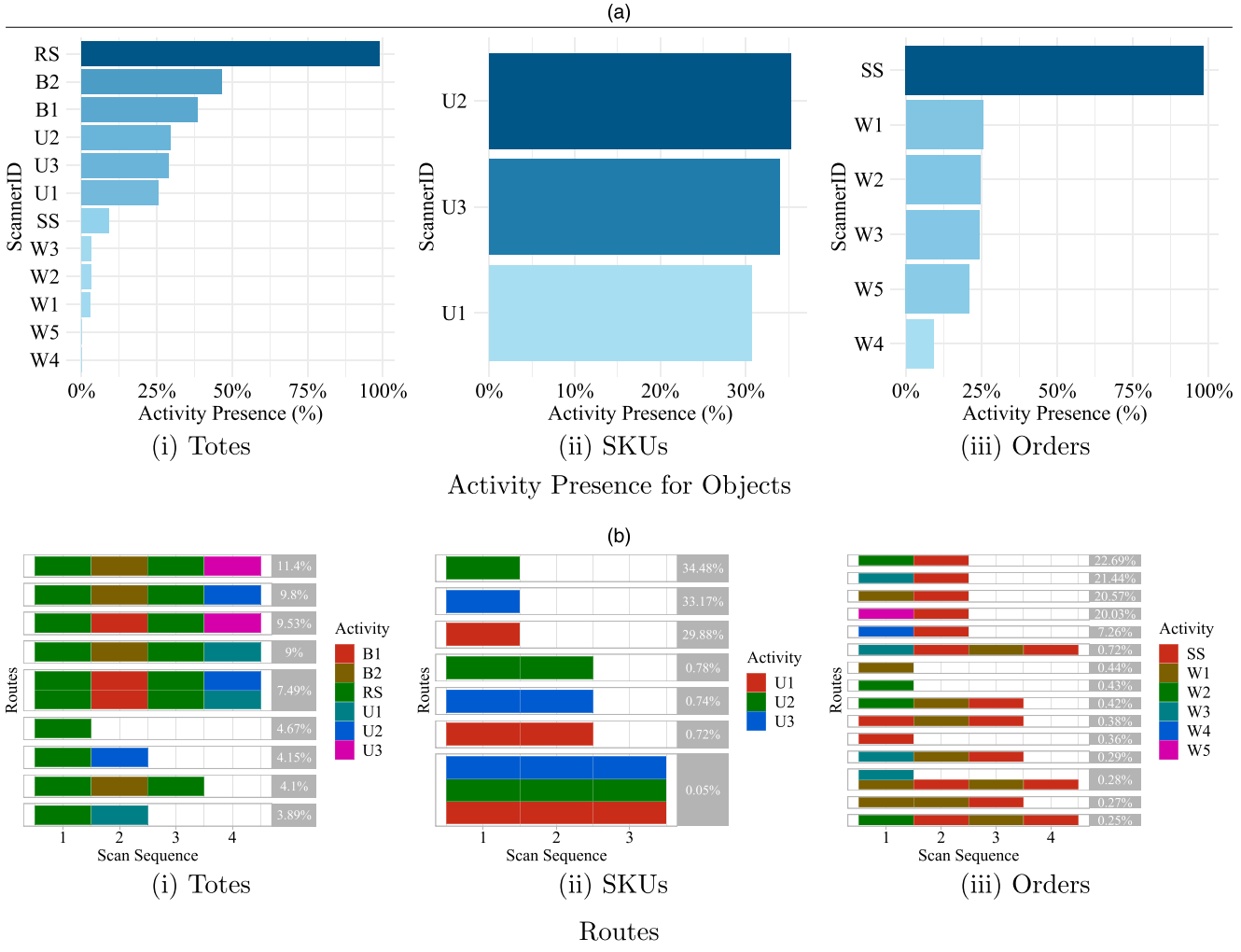
In this section, we detail the procedure followed to derive the warehouse's process flows and layout from data. The data provided contains several data tables that record time stamps of events occurring at the warehouse over a period of 14 days. The first step in understanding the data is to investigate column attributes and the range of values they take. This reveals a total of 12 scanners with unique identifiers that linked each scanner to its function. Based on scanner names, we identify four possible processes/areas: buffer (scanners B1, B2), sorter packing (scanners U1, U2, U3), weighing (scanners W1, W2, W3, W4, W5), and shipping (scanner SS). The 12th scanner is a routing scanner (RS). This first step also reveals three objects being scanned and directed to the various processes/areas. These are totes, stock-keeping units (SKUs), and orders; totes are linked to waves and SKUs, and SKUs are linked to orders. Time stamps or scans provide the date and time of the scan, the object scanned, the scanner, and the destination. Using scanner data, we derive process-flow diagrams for each object type (totes, SKUs, and orders) using the *R* statistical software packages *BupaR* and *ProcessmapR* (Janssenswillen 2019). *BupaR* is used to convert scanner data into an event log object, which serves as input to *ProcessmapR* to visualize process-flow diagrams. We start by removing anomalies (empty scans) as well as rare events to obtain the main process flows to explain warehouse operations. To do so, we first determine the presence of scanners in each object's route and remove infrequent ones. Figure 2(a) plots the percentage of objects scanned at different scanners. Totes scan at all scanners with 99% of the totes being scanned at the routing scanner RS. For each object, we exclude scanners with an activity level less than 10%. For totes, scanners SS, W1, W2, W3, and W4 are removed. Figure 2(b) plots frequent routes for each object type. Out of 1,011 distinct tote routes, 71% of totes follow one of the 10 routes as illustrated in Figure 2(b)(i). In particular, the 35 most frequent routes cover 90% of tote routes. As such, many routes are infrequent, and totes hardly follow them as suggested by the fact that the remaining 976 routes are followed by only 10% of the totes. To plot a process flow, we consider routes that are followed by 90% of the totes.

We carry out similar analyses for SKUs and orders. Figure 2(a)(ii) shows that SKUs are only scanned at scanners U1, U2, and U3 in roughly similar proportions, and Figure 2(b)(ii) illustrates the top 10 routes covering 99.9% of SKU routes. This confirms that SKUs are only scanned at one of the scanners, U1, U2, or U3, and in case of an improper scan, the SKU is rescanned at the same scanner. In a similar fashion, we determine frequent scanners and routes followed by orders as shown in Figure 2, (a)(iii) and (b)(iii), respectively. Orders are scanned at scanners W1, W2, W3, W4, W5, and SS, and scanners W1, W2, W3, W4, and W5 are scale scanners to weigh the order package before the final scan at SS, which is the shipping scanner. This is confirmed from the most frequent routes followed by orders in Figure 2(b)(iii), in which 14 out of 16 routes end at scanner SS. Note also that the 16 most frequent routes cover 96% of order routes. All other routes have a frequency of less than 0.25%. We, therefore, exclude them in deriving process flows for orders.

The resulting process flows for each object type are illustrated in Figure 3, in which the percentage value denotes the percentage of totes flowing through an arc or node, and the number in brackets denotes median time in minutes between two successive scans. For totes, we plot the flow of totes in a time–space network in which the horizontal time axis refers to the scan order, and the vertical axis refers to the scanners as shown in Figure 4. The labels on the arcs refer to the percentage of tote flows and the median time between scans connected by the arc, respectively. We now combine the process-flow diagrams for all objects to draw the layout.

Process flows derived from the scanner data show the flow of objects in the warehouse. For instance, consider tote process flows as illustrated in Figure 4, in which totes are first scanned at scanner RS, from which they are mostly diverted to either scanner B1 (35.2%) or B2 (44.8%). Roughly the same percentage of totes originating from scanners B1 and B2 suggest that totes are rerouted to scanner RS. In contrast to the median time of one minute on arcs (RS, B1) and (RS, B2), the median time for arcs (B1, RS) and (RS, B2) is 128 and 133 minutes, respectively. This suggests that, after scanning at B1/B2, totes are held at a buffer area before being diverted to scanner RS for a second time. Table 2 summarizes the percentage of totes and their destinations after scans at B1 and B2. More than 99% of the totes are diverted to one of two aisles in each scanner.

Figure 2. (Color online) Activities and Routes Followed by Object Types: Totes, SKUs, and Orders



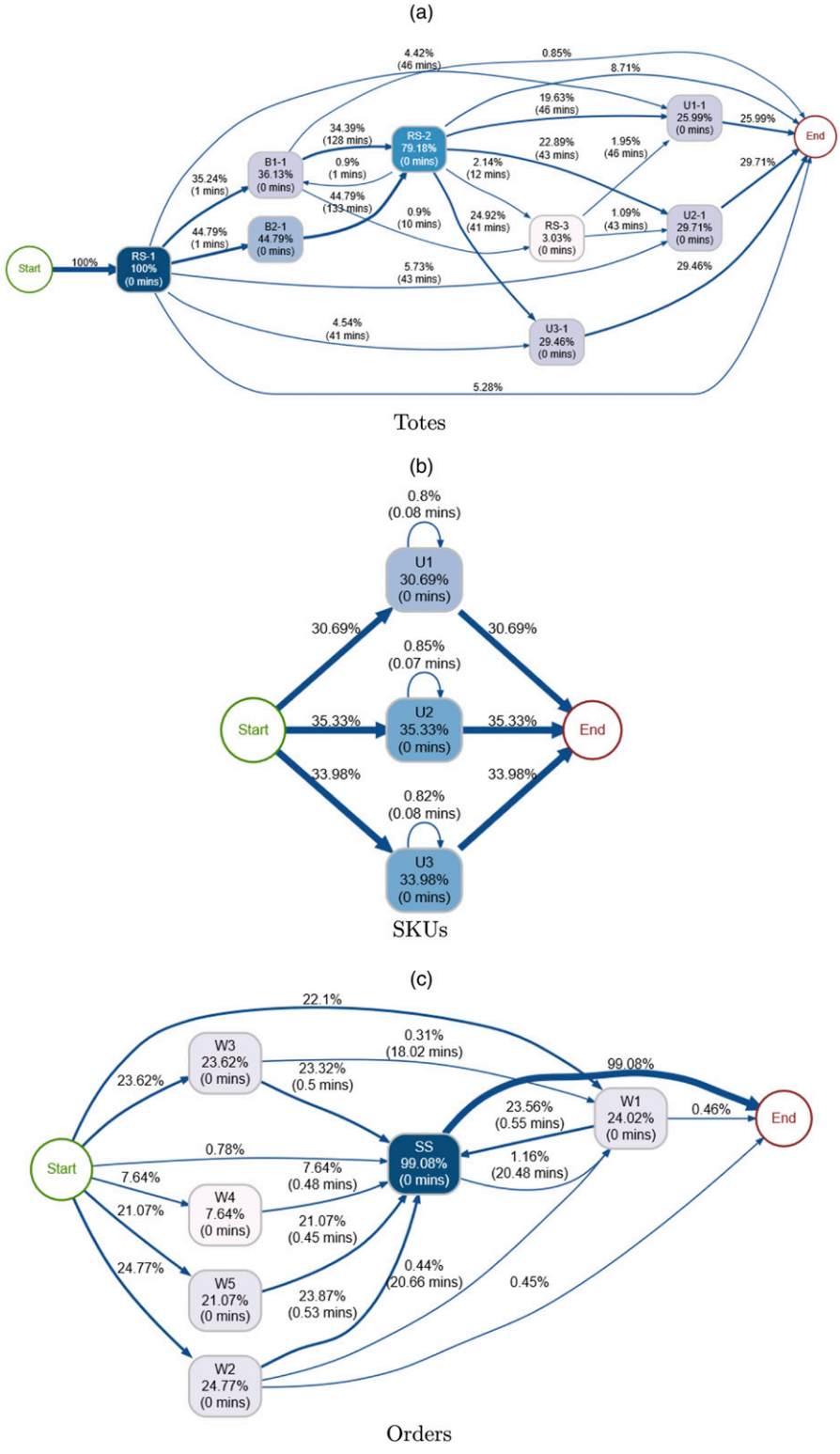
Using this information, we map the process flows in Figure 4 to the warehouse layout region labeled "1" in Figure 5. Process flows show that 85% of the totes have their last scan at scanners U1, U2, or U3. Analyzing destinations for totes at these scanners reveals that totes are diverted to a destination *EmptyTote*. This suggests that there is a dedicated region in the warehouse where empty totes are sent and is labeled as region "2" in the warehouse layout of Figure 5.

The analysis of the process-flow diagram for SKUs shows that they are scanned only at either scanners U1, U2, or U3. This suggests that the content of totes is emptied at scanners U1, U2, and U3. Destinations reveal that each SKU of an order is sent to a cubby assigned to the order, which allows us to deduce region "3" of the warehouse. Process flows for orders show that 99.2% of the orders are first scanned at scale scanners (W1, W2, W3, W4, W5) before being diverted to the shipping scanner SS. Destinations of SS provide information regarding different shipping zones to which an order is diverted. In some cases, orders require reweighting and are sent back to scale scanners. This analysis allows us to deduce the shipping region of the warehouse labeled "4" in Figure 5. After deriving the process flows and layout, we proceed to analyze the process times.

4. Analysis of the Order-Fulfillment Operation

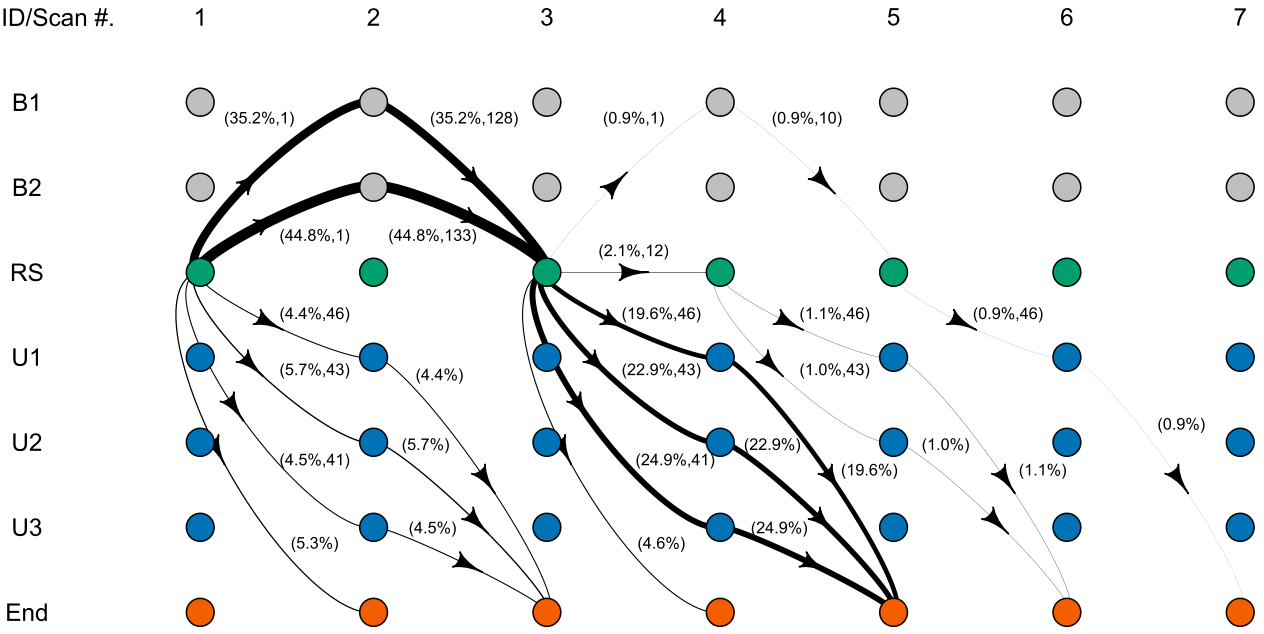
As orders are delivered directly to customers, the daily demand of an e-warehouse is typically higher than that of a regular warehouse. For example, the warehouse under study processes an average of 31,250 orders containing more than 150,000 products and may operate for up to 24 hours in a day. The characteristics of the wave in terms of the number and variety of orders and products is important. For the warehouse under study, statistics on products, orders, and waves for an eight-day period is given in Figure 6.

Figure 3. (Color online) Process Flow Diagrams for Totes, SKUs, and Orders



As displayed in Figure 6(a), the average number of orders and products per wave vary between 790 and 1,405 and 3,955 and 7,100, respectively. Order characteristics impact the pick lists and, consequently, the contents of the totes and, eventually, the consolidation process. Similarly, according to Figure 6(b), 75% of orders contain at most five products, and only 7% contain more than 10. Figure 6(c) displays the number of

Figure 4. (Color online) Time–Space Network for Totes



waves, orders, products, and totes for each day. The number of waves ranges between 6 and 92, the number of orders between 9,835 and 74,680, the number of products between 24,600 and 387,450, and the number of totes between 4,252 and 20,630. This implies that, on average, 32,814 orders containing 165,335 products are processed in 34 waves using 10,742 totes per day.

Order fulfillment comprises of three main processes: picking, sorting, and putting. As illustrated in Figure 7, picking starts as soon as a wave is released. Once products in a tote are picked, the tote is directed to a buffer area. When all totes of the sort wave arrive in the buffer, sortation starts. Totes are diverted from the buffer to the unit sorter. Sortation ends with putting all products in assigned cubbies.

Complete consolidated orders undergo packing before finally being delivered to customers. The putting time of an order is defined as the time between the induction of the first and the last products in the order. The consolidation time for an order is the time between the start of the sorting process and the end of the putting process of the order.

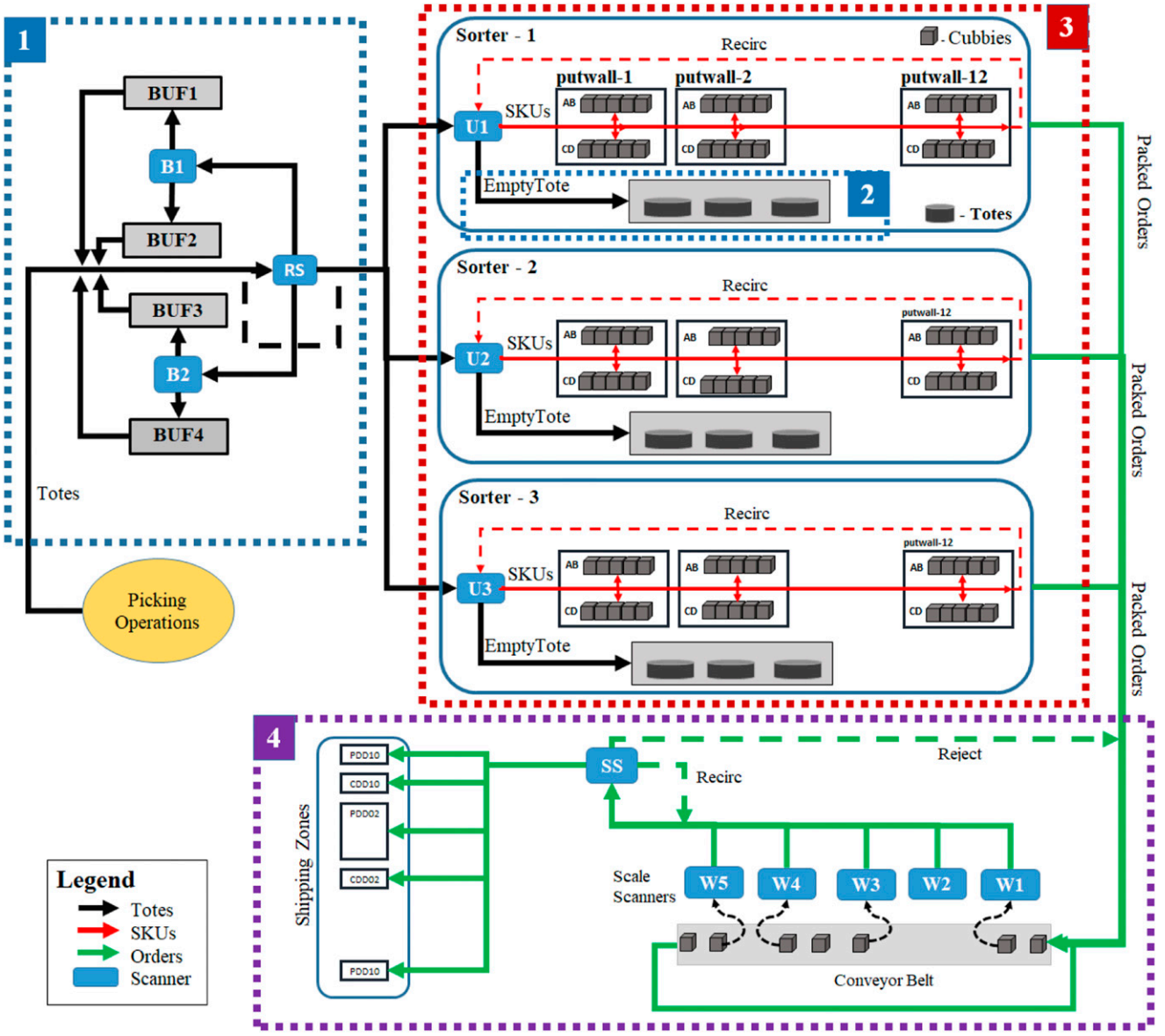
Based on the data, we investigate the average order-completion times to understand the importance of the processes it involves. First, we clean the data by eliminating waves that are processed over multiple days to remove observations with huge break times. Second, we eliminate the waves whose processes are stopped because of lunch or dinner breaks. The reason for eliminating these waves instead of removing the break times is the uncertainty associated with operators and induction lines before and after breaks. We identify 17 waves over eight days. Figure 8 displays picking, putting, consolidation, and waiting times for the 17 waves. On average, picking takes 61 minutes, and consolidation takes 56 minutes. Given the lack of data on the picking process, we focus the analysis on the order consolidation process, which covers waiting in the buffer area and putting.

Next, we gather statistics on the number of orders, the number of products, the average consolidation time per order, and the average waiting time for packaging per order. Figure 9 plots average order consolidation and waiting time for all waves in increasing order of average time. For this, we include the 273 waves and

Table 2. Destinations After Scans at B1 and B2

Scanner B1		Scanner B2	
Destination	Percentage of totes	Destination	Percentage of totes
BUF1	80.06	BUF3	51.54
BUF2	19.92	BUF4	48.31
Reject	0.02	Reject	0.15

Figure 5. (Color online) Warehouse Layout

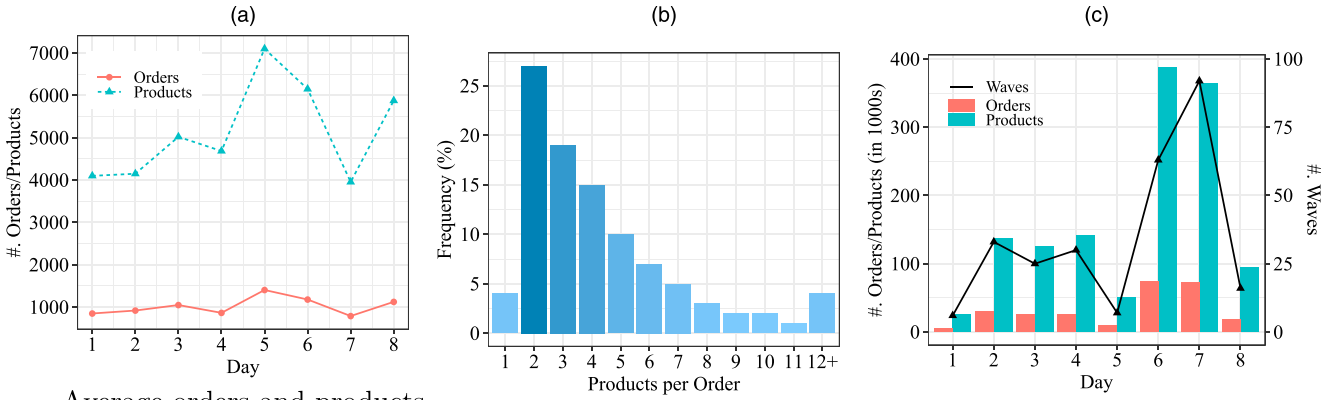


eliminate breaks longer than one minute. Figure 10, (a) and (b), depicts the distribution of orders and products per wave, respectively. More than 50% of waves have between 1,300 and 1,400 products, and 30% have a maximum of 400. Figure 10(b) provides similar features. Around 30% of waves have a maximum of 3,000 products, and around 50% have between 5,000 and 8,000. Both figures suggest that there are two types of waves: small- and large-sized. The distributions of the average consolidation time per order and average waiting time for packaging per order are given in Figure 11, (a) and (b), respectively. In the following section, we investigate the order consolidation process further as it accounts for the majority of processing time after picking.

5. Optimizing the Order Consolidation Process

The consolidation time of an order, that is, the time from the beginning of the wave until the last product in the order is inducted, depends on the number of totes that carry the order's products and their induction sequence. Because all totes of a wave are available in the buffer area at the start of consolidation, totes are emptied at the induction line according to the sequence in which they are released from the buffer area. An order seizes a cubby from the time that the first product from the order is inducted to the time that the last product of the order is inducted. Our calculation of the putting time for an order is tote-based rather

Figure 6. (Color online) Order, Product, and Wave Characteristics



Average orders and products per wave.

Products per order.

Daily operation.

than product-based. A tote-based calculation considers a tote as a whole and equates the induction time of all products from the tote to the induction time of the last product inducted. Product-based calculation would require the actual induction time of products, which is not available as operators pick products arbitrarily.

Reducing the average consolidation time also leads to efficient usage of resources. One of the critical resources is the cubbies. Each wave is assigned to a put wall, which has a determined number of cubbies (1,309 for the warehouse under study). Figure 12 highlights the relation between average cubby usage and average consolidation time. It is clear that there is almost a linear relationship.

To reduce order consolidation times, the OCP is introduced to determine the sequence in which totes are inducted, that is, emptied of their content. The OCP is a generalization of the parallel machine–scheduling problem as it introduces a third element besides jobs and machines. In OCP, there are totes (jobs), induction lines (machines), and orders. An order may be considered a split job, and a tote may be considered as a set of split jobs. To finish processing an order, all of its items from different totes have to be processed. In terms of implementation in the current system, the OCP solution integrates naturally by dictating an optimal sequence in which totes are diverted from the buffer area to the unit sorter and assigned to the induction lanes.

5.1. Mathematical Formulation

We use indices $k \in K$ for orders and $i, j \in J$ for totes. Once in possession of a tote, an operator starts emptying the products one at a time at one of m induction lines. The processing time of tote j, that is, the time it takes to empty it, is denoted by p_j. We define two sets of continuous decision variables for the completion time of orders and totes, denoted by CO_k for order $k \in K$ and C_j for tote $j \in J$, respectively. We also define assignment variable x_{ij}, which takes value one if tote $i \in J$ precedes tote $j \in J$, and introduce additional sets J_k, J_0, and J_{n+1}, where J_k is a subset of J, containing totes with products belonging to order $k \in K$. J_0 and J_{n+1} are sets with

Figure 7. Definitions of the Durations and the Times

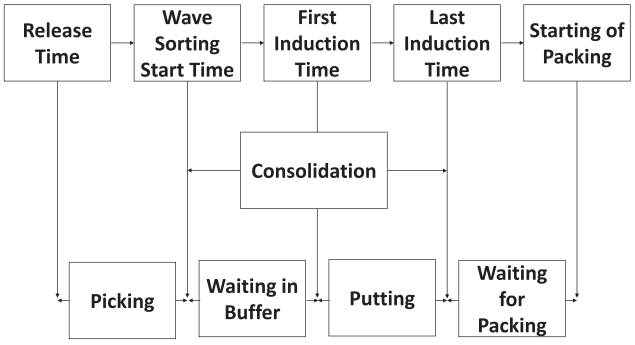
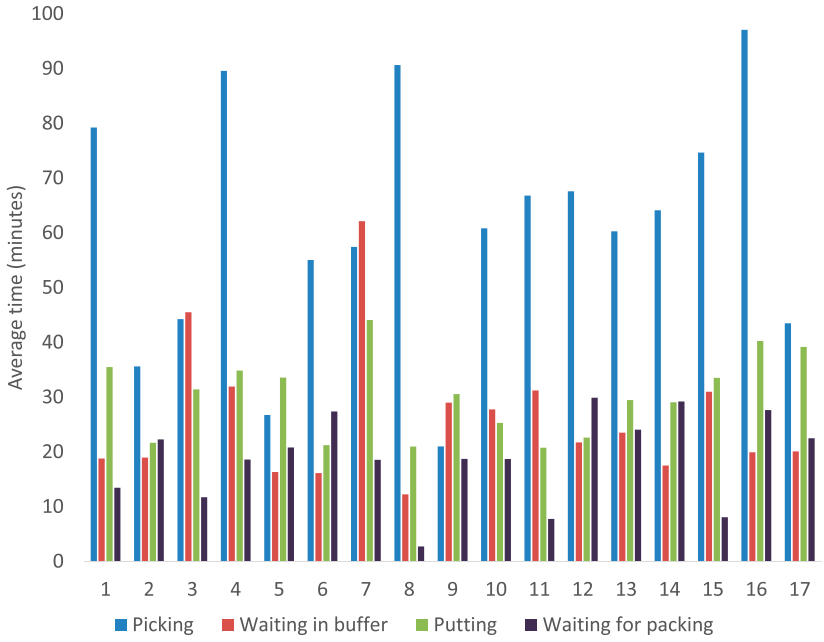


Figure 8. (Color online) Average Picking, Putting, Consolidation, and Waiting Times for 17 Waves



dummy node zero and dummy node $n+1$, respectively. A mixed-integer nonlinear program formulation is given by

$$\text{(NMIP)}: \min \sum_{k\in K} CO_k, \tag{1}$$

$$\text{s.t.} \sum_{i\in J_{n+1}} x_{ji} = 1 \qquad j \in J, \tag{2}$$

$$\sum_{j\in J} x_{0j} = m, \tag{3}$$

$$\sum_{i\in J_0} x_{ij} - \sum_{i\in J_{n+1}} x_{ji} = 0 \qquad j \in J, \tag{4}$$

$$C_j = p_j x_{0j} + \sum_{i\in J} (C_i + p_j) x_{ij} \qquad j \in J, \tag{5}$$

$$CO_k \geq C_j \qquad j \in J_k, k \in K, \tag{6}$$

$$C_j \geq 0 \qquad j \in J, \tag{7}$$

$$CO_k \geq 0 \qquad k \in K, \tag{8}$$

$$x_{ij} \in \{0,1\} \qquad i \in J_0, j \in J_{n+1}. \tag{9}$$

Figure 9. (Color online) Average Consolidation Times for All Waves

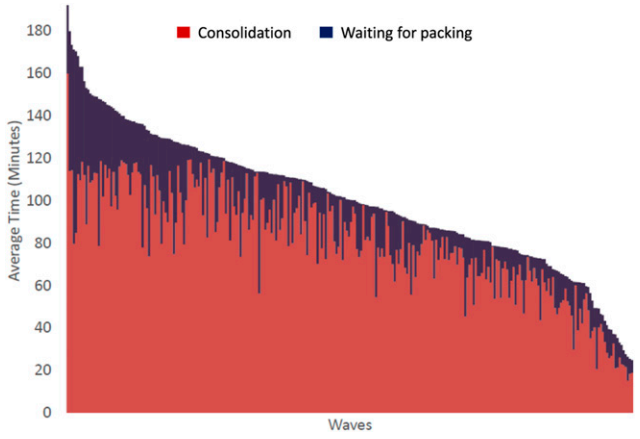
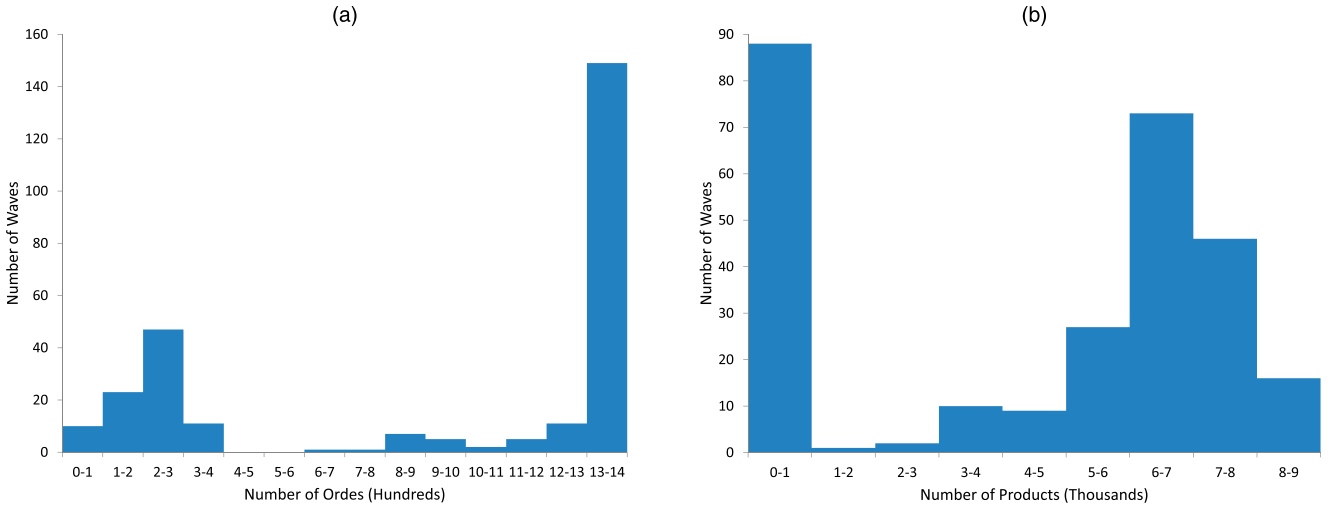


Figure 10. (Color online) Average Wave Sizes



Number of orders per wave.

Number of products per wave.

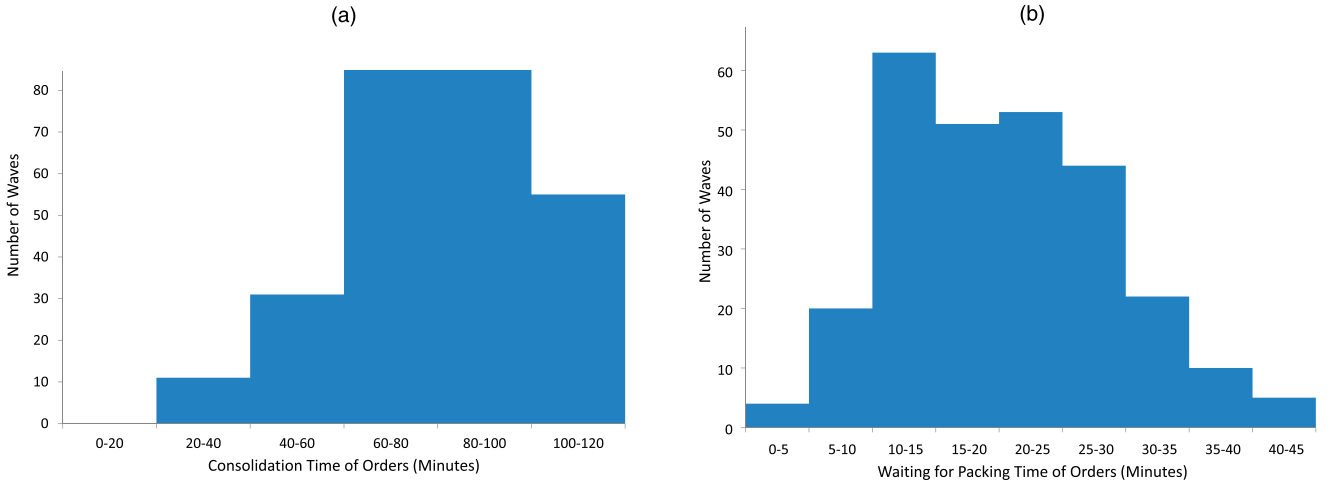
The objective function (1) minimizes the total completion time of orders. Constraints (2) make sure that each tote is followed by exactly one tote unless it is the last tote in the line. Constraint (3) ensures that there are m induction lines, m sequences of totes are formed, and m totes are assigned to the dummy tote. Constraints (4) are the flow balance constraints to ensure that the precedence assignments form sequences. Constraints (5) calculate the completion time of totes. Constraints (6) calculate the order completion times. Although nonlinear, constraints (5) are linearized as

$$C_j \geq p_j x_{0j} \qquad j \in J. \tag{10}$$

$$C_j \geq (C_i + p_j) - M(1 - x_{ij}) \qquad i, j \in J. \tag{11}$$

Let us consider the special case in which orders are not split over multiple totes, that is, the entirety of an order is allotted to one tote. In this case, the OCP reduces to the parallel machine–scheduling problem with weighted

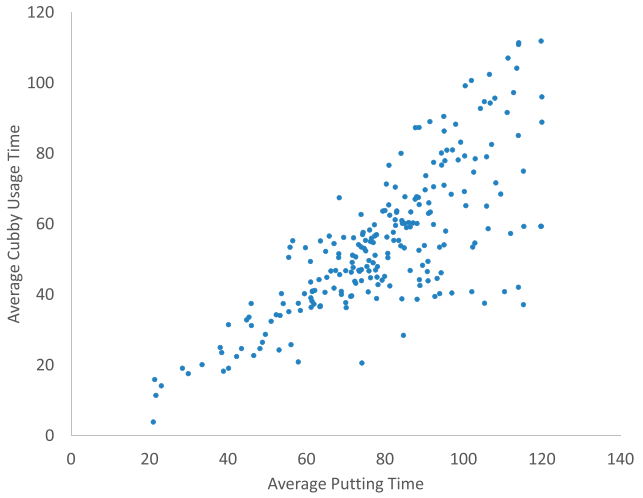
Figure 11. (Color online) Average Consolidation and Waiting Times



Average consolidation time per wave.

Average waiting time for packaging per wave.

Figure 12. (Color online) Relation Between Consolidation Time and Cubby Usage



completion time. Because each order corresponds to a single tote, the variable CO_k is eliminated, and the objective function reduces to

$$\min \sum_{k\in K} \sum_{j\in J_k} C_j. \tag{12}$$

The weighted completion time of a tote is the sum of the completion times of the orders in it. Therefore, the weights correspond to the number of orders in a tote, and the problem reduces to the parallel machine–scheduling problem with total weighted completion time minimization, which is known to be NP-hard. Hence, the OCP is also NP-hard.

6. A Branch-and-Price Approach for the OCP

Let S denote the set of schedules on a single induction line. For each tote $j \in J$, let parameter $b_j^s = 1$ if schedule $s \in S$ includes tote j and zero otherwise. Defining variable y_s to take value one if schedule $s \in S$ is selected and zero otherwise, a set-partitioning formulation, equivalent to (NMIP), is

$$(\text{ISP}): \min \sum_{k\in K} CO_k, \tag{13}$$

$$\text{s.t. } CO_k \geq \sum_{s\in S} C_j^s y_s \qquad j \in J_k, k \in K, \tag{14}$$

$$\sum_{s\in S} b_j^s y_s = 1 \qquad j \in J, \tag{15}$$

$$\sum_{s\in S} y_s = m, \tag{16}$$

$$y_s \in \{0,1\} \qquad s \in S. \tag{17}$$

Constraints (15) and (16) correspond to the original constraints (2) and (3), which are the assignment and induction line capacity constraints. Constraints (15) ensure that each tote is covered by exactly one schedule. Constraint (16) ensures that m schedules are formed. Constraints (14) calculate the completion time of orders.

To solve (ISP), we use a column-generation approach. Let (LSP) denote the linear relaxation of (ISP). Its dual problem is

$$(\text{DSP}): \max m\lambda_0 + \sum_{j\in J} \lambda_j, \tag{18}$$

$$\text{s.t. } \sum_{j\in J_k} \beta_{jk} \leq 1 \qquad k \in K, \tag{19}$$

$$\sum_{k\in K} \sum_{j\in J_k} C_j^s \beta_{jk} - \sum_{j\in J} b_j^s \lambda_j - \lambda_0 \geq 0 \qquad s \in S, \tag{20}$$

$$\beta_{jk} \geq 0 \qquad j \in J_k, k \in K, \tag{21}$$

where β_{jk}, $j \in J_k$, $k \in K$, λ_j, $j \in J$, and λ_0 denote the dual variables corresponding to constraints (14)–(16), respectively. Column generation starts by solving (LSP) on a subset of schedules $\bar{S} \subseteq S$, verifies whether the solution of the restricted problem [LSP($\bar{S}$)] is optimal to the original problem (LSP). If it is not optimal, a new schedule is generated by solving the pricing subproblem:

$$(\text{PP}): \min \sum_{j \in J} \sum_{k: j \in J_k} \bar{\beta}_{jk} C_j - \sum_{i \in J_0} \sum_{j \in J} \bar{\lambda}_j x_{ij}, \tag{22}$$

$$\text{s.t. } (4), (5), (7), (9) \tag{23}$$

for fixed dual variables $\bar{\beta}_{jk}, \bar{\lambda}_j, \bar{\lambda}_0$. If the objective of (PP) is less than $\bar{\lambda}_0$, the corresponding solution defines a new schedule for $\bar{S}$, and the process is restarted. Columns are added to [LSP($\bar{S}$)] until no column with a negative reduced cost is identified. In practice, an alternative stopping criterion is when the relative gap between the lower bound and the upper bound is less than a predetermined value (e.g., 0.001). A lower bound for (LSP) at iteration h is given by $LB_h = \max\{LB_{h-1}, UB + u\}$, where u and UB are the objective values of (PP) and [LSP($\bar{S}$)], respectively.

To solve (ISP) to optimality, the column-generation approach is embedded within branch and bound. For that, a carefully designed branching rule as well as specialized algorithms for the solution of the subproblem before and after branching are needed. At node 0 of the branch-and-bound tree, the pricing subproblem (PP) is a single machine–scheduling problem with minimum weighted completion time. It may be efficiently solved using the dynamic programming algorithm of Chen and Powell (1999b). After branching, however, the aforementioned algorithm fails. To overcome this, we provide a new formulation of the subproblem, and we devise a new branching rule and a new dynamic programming algorithm.

6.1. Solution of the Subproblem

According to Smith's (1956) rule, in any optimal schedule for the parallel machine–scheduling problem with the objective of minimizing weighted completion time, jobs on each machine must follow the shortest weighted processing time (SWPT) rule. Subproblem (PP) determines a subset of jobs that minimizes the total weighted completion time minus the sum of the dual variables. For a given subset of jobs, the problem reduces to a minimum total weighted completion time problem on a single machine, whose optimal solution satisfies the SWPT rule. Based on this important observation, Chen and Powell (1999b) propose a dynamic programming algorithm with $O(nP)$ worst case complexity in which P is the total processing time of jobs and n is the number of jobs. The algorithm uses the recursion function $F(j,t)$ defined as the minimum objective value of a partial schedule that contains a subset of jobs of $\{1,2,\ldots,j\}$ that satisfies the SWPT rule and is completed at time t. It is given by $F(j,t) = \min\{F(j-1, t-p_j) + tw_j - \lambda_j, F(j-1,t)\}$, where w_j and λ_j are the weight and the dual variable value for job j. The solution of the problem is $\min_{0 \le t \le P} F(n,t)$. At node 0 of the branch-and-price algorithm, the pricing subproblem (PP) satisfies the SWPT rule. The weight of a tote (analogous to a job in the machine-scheduling problem) is calculated as $\sum_{k: j \in J_k} \bar{\beta}_{jk}$ for $j \in J$. Hence, (PP) may be solved using the dynamic programming algorithm of Chen and Powell (1999b) given in Algorithm 1.

Algorithm 1 (Dynamic Programming Algorithm to Solve the Subproblem at the Root Node).

For each $j \in N, t \in P$, where t< 0, set $F(j,t) = \infty$
For each t, set $F(0,t) = 0$
For each $j \in N$
  For each $t \in P$
    $F(j,t) = \min\{F(j-1, t-p_j) + tw_j - \lambda_j, F(j-1,t)\}$
  End For
End For
$z^* = \infty$
For each $j \in N, t \in P$
  If $z^* \le F(j,t)$, Then
    $z^* \leftarrow F(j,t)$
  End If
End For

After branching, the subproblem has to be solved efficiently. Unfortunately, branching on variables y_s is difficult to enforce for $y_s = 0$. As the column corresponding to y_s represents a feasible schedule, setting y_s to zero means that the schedule has to be removed, which cannot be enforced in the dynamic programming algorithm.

On the other hand, if branching on x_{ij}, the precedence relationships between totes have to be considered in the subproblem.

Branch and price has a proven track record in solving parallel machine–scheduling problems. The subproblem is often set up as a single-machine subproblem and is solved using dynamic programing. Depending on the structure of the subproblem, a carefully designed branching rule is needed in order to preserve the structure of the subproblem after branching. This often takes the form of a restriction on the sequence of jobs processed at the child nodes.

Generally, one of the important and difficult steps in the design of a branch-and-price algorithm is the branching rule. It is not advisable to append branching requirements as explicit constraints as this may render the subproblems difficult to solve after branching—sometimes as difficult as the original problem. They should, instead, be handled implicitly, either in the solution approach of the subproblem or by altering the subproblem parameters.

After branching, both van Den Akker et al. (1999) and Chen and Powell (1999b) can accommodate the precedence relationship resulting from the TWCT objective function considered. They exploit the fact that there is at least one optimal solution satisfying Smith's rule and only generate columns that satisfy it. When branching, the relationship between jobs is enforced according to Smith's rule, and the subproblem is solved using a revised version of Algorithm 1. Although this works for the parallel machine–scheduling problem, it does not for the order consolidation problem. After branching, the subproblem in OCP is no longer a single machine–scheduling problem. Enforcing the branching constraints modifies the subproblem and makes the dynamic programming algorithm of van Den Akker et al. (1999) and Chen and Powell (1999b) no longer applicable. To deal with this difficulty, we introduce new variables, based on which we provide a new model, a new branching rule, and a new dynamic programming algorithm to solve the resulting subproblem. To the best of our knowledge, none of the previously designed branching rules for parallel-machine scheduling would apply to the current problem.

6.2. A New Formulation of the Subproblem

To efficiently solve the subproblems after branching, we propose an alternative mathematical formulation for the OCP with two sets of binary variables to replace the x variables. Based on the new formulation, we develop a new branching rule and a new dynamic programming algorithm to solve the resulting subproblem after branching.

We use the same indices, parameters, and continuous variables as (NMIP). In addition, we define set L of m machines, denoted by L. We define z_{ij}, which takes value one if totes i and j are processed on the same induction line and tote i precedes tote j. We also define binary variables t_{jl}, which takes value one if tote $j \in J$ is processed in line $l \in L$. The alternative formulation is

$$(\text{NMIP2}): \quad \min \sum_{k\in K} CO_k, \tag{24}$$

$$\text{s.t. } C_i + p_j \le C_j + M(1 - z_{ij}) \qquad i,j \in J, \tag{25}$$

$$z_{ij} + z_{ji} = \sum_{l\in L} t_{il}t_{jl}i, \qquad j \in J, \tag{26}$$

$$\sum_{l\in L} t_{jl} = 1 \qquad j \in J, \tag{27}$$

$$CO_k \ge C_j \qquad j \in J_k, k \in K, \tag{28}$$

$$z_{ij} \in \{0,1\} \qquad i,j \in J, \tag{29}$$

$$t_{jl} \in \{0,1\} \qquad j \in J, l \in L. \tag{30}$$

The objective function (24) minimizes the total completion time, and constraints (25) calculate the completion times of totes. Constraints (26) are nonlinear, and they ensure that $z_{ij} + z_{ji} = 1$ when both totes i and j are processed in the same induction line; otherwise, $z_{ij} + z_{ji} = 0$. Constraints (27) are assignment constraints that enforce each tote to be processed on only one induction line. Constraints (28) calculate the completion times of the orders.

In the subproblem, we solve a single machine–scheduling problem by selecting and scheduling a subset of totes. A formulation of subproblem [SPP] that uses z_{ij} and t_j is

$$(\text{SPP2}): \min \sum_{j\in J}\sum_{k:j\in J_k}\bar{\beta}_{jk}C_j - \sum_{j\in J}\bar{\lambda}_j t_j, \tag{31}$$

$$\text{s.t. } C_i + p_j t_j \le C_j + M(1 - z_{ij}) \qquad i,j \in J, \tag{32}$$

$$z_{ij} + z_{ji} \le 0.5(t_i + t_j) \qquad i,j \in J, \tag{33}$$

$$t_i + t_j - 1 \le z_{ij} + z_{ji} \qquad j \in J, \tag{34}$$

$$z_{ij}, t_j \in \{0,1\} \qquad i,j \in J. \tag{35}$$

Note that $t_j = 1$ implies that tote j is selected. The objective function (31) minimizes the total weighted completion time of the selected totes plus the cost of including a tote in the schedule. Constraints (32) calculate the completion time of tote $j \in J$. Constraints (33) and (34) enforce the relationship between totes if they are both selected.

For any feasible solution $y_s, s \in S$ in (ISP), there is a corresponding solution z_{ij}, t_j, feasible to (SPP2), such that

$$z_{ij} = \sum_{s\in S} e_{ij}^s y_s, \tag{36}$$

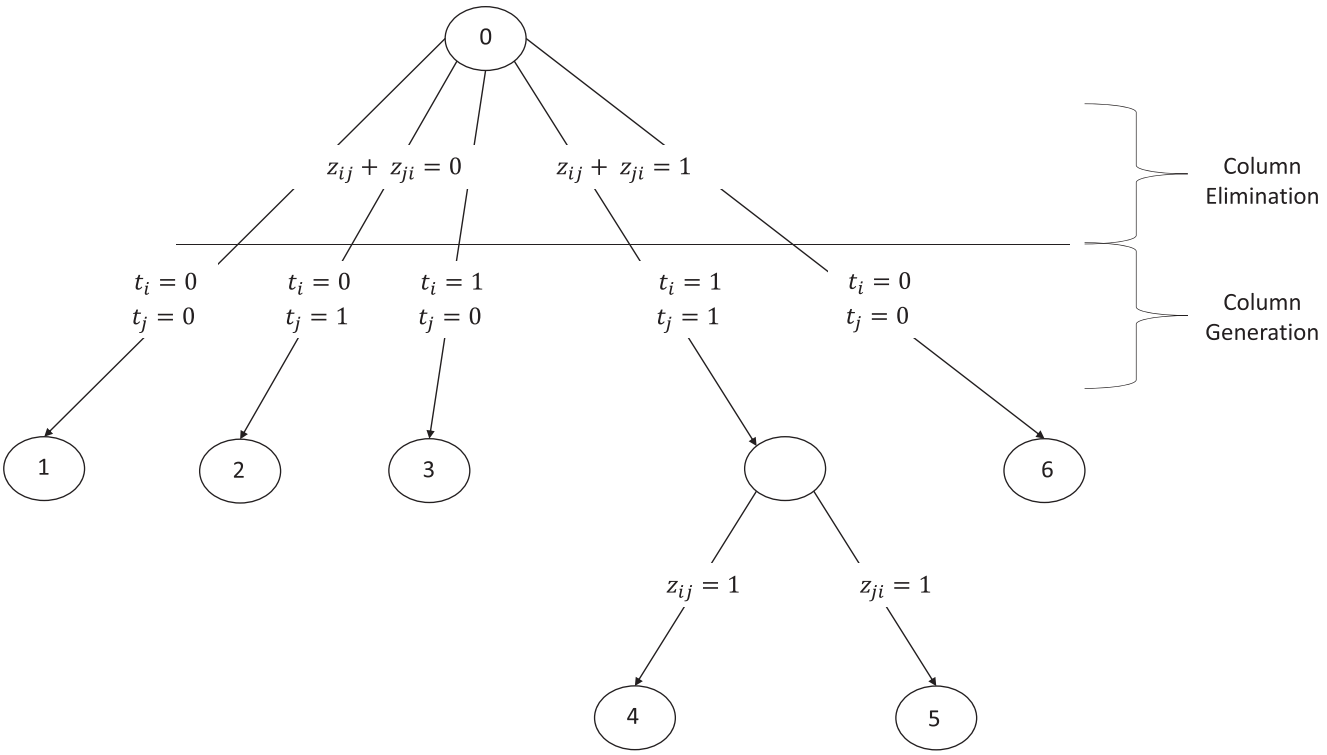
where e_{ij}^s is one if totes i and j are both contained in schedule s and tote i is processed before tote j. Once the root node is solved, if the solution $y_s, s \in S$ is fractional, the corresponding z_{ij} variables are computed using (36).

6.3. A New Branching Rule and Solution of the Subproblem

Instead of branching on fractional z_{ij} variables by creating two branches for $z_{ij} = 0$ and $z_{ij} = 1$, we propose to branch on whether two totes are in the same column. Accordingly, a pair of totes (i,j) is selected such that $z_{ij} + z_{ji}$ is closest to 0.5, that is, with the maximum integer infeasibility. Five branches are then created: two branches with $z_{ij} + z_{ji}$ fixed to one and the other branches with $z_{ij} + z_{ji}$ fixed to zero. If $z_{ij} + z_{ji}$ is fixed to zero, the initial restricted master problem of the corresponding child node consists of all the columns of its parent node except the ones in which totes i and j are scheduled in the same column. If $z_{ij} + z_{ji}$ is fixed to one, the initial restricted master problem of the corresponding child node consists of all the columns of its parent node except the ones in which only one of i or j is scheduled in a column. When $z_{ij} + z_{ji} = 0$, there are three possibilities for the subproblem. We may select only tote i, only tote j, or neither. As a result, the first child node imposes that tote i and j cannot be scheduled at the same column. When generating columns with negative reduced cost, columns must contain only tote i from pair (i,j). For the second, columns that contain only tote j are generated. In the third child node, no column containing either tote from pair (i,j) can be scheduled. For the child nodes in which $z_{ij} + z_{ji} = 1$, there are two cases: both totes of pair (i,j) are scheduled in the columns generated by the subproblem or neither of (i,j) is scheduled in the same column. When $z_{ij} + z_{ji} = 1$ is enforced in the subproblem, i and j are both processed on the same machine, which means that they are not processed on all the other machines. Hence, columns in which both i and j are absent may also be generated. That is why we allow generating both types of columns (schedules): those in which both are present and those in which none are present. As a summary, we use z_{ij} to eliminate columns when branching and t_j to solve the subproblem and generate new columns. Figure 13 illustrates the branching rule. The third level in Figure 13 corresponds to the case in which $z_{ij} + z_{ji} = 1$ but y_s is still fractional. In that case, we further branch on whether i proceeds j or vice versa.

As mentioned earlier, the dynamic programming algorithm of Chen and Powell (1999b) fails to accumulate precedence-based branching constraints. To overcome this, we propose a new dynamic programming algorithm. Let $F(j,t)$ denote the minimum objective value (total weighted completion time minus the sum of the dual variable values minus the value of selecting a tote) in a partial schedule consisting of a subset of totes of $1, 2, \ldots, j$, provided that the partial schedule follows the SWPT rule and that tote j is the last tote that is completed at time t in the partial schedule. To fix or eliminate a tote, because of the branching constraints, we define a cost π_j and set it large enough to eliminate tote j or sufficiently negative to select tote j. The dynamic programming algorithm is provided in Algorithm 2 in which B_j is the set of jobs that can be processed immediately before j. The worst-case complexity of Algorithm 2 is bounded by $O(n^2P)$ as there are nP states, and it takes no more than $O(n)$ to evaluate one state.

Figure 13. An Illustration of the Branching Rule



Algorithm 2 (Dynamic Programming Algorithm to Solve the Subproblem After Branching).

For each $j \in N, t \in P$, where t< 0, set $F(j,t) = \infty$
For each t, set $F(0,t) = 0$
For each $j \in N$
  For each $t \in P$
    $F(j,t) = \min_{i \in B_j \cup \{0\}} \{F(i, t - p_j) + tw_j - \lambda_j - \pi_j\}$
  End For
End For
$z^* = \infty$
For each $j \in N, t \in P$
  If $z^* \leq F(j,t)$, Then
    $z^* \leftarrow F(j,t)$
  End If
End For

6.4. A Simulated Annealing Approach for the OCP

Simulated annealing (SA) is a metaheuristic approach that is inspired from the physical annealing of solids (Metropolis et al. 1953). SA is an iterative algorithm that moves from one solution x^0 with objective z_0 to a neighboring solution x^1 with objective z_1 if $z_1 \leq z_0$. If not, a move to x^1 is performed with acceptance probability $\exp(z_1 - z_0/\tau)$, where τ is a control parameter that corresponds to the temperature in the analogous annealing process. As the search proceeds, the temperature is gradually decreased by a factor of $1 - q$, where q is the temperature cooling parameter. For the OCP, the solutions are represented as a single vector for all totes. Schedules are obtained by assigning totes from a given solution to induction lines one by one according to availability. Compared with representing a solution using m vectors, one for each induction line, a single vector representation allows for a better neighborhood search.

The SA algorithm is initiated using a list-based heuristic, which forms a list of totes and then assigns them sequentially whenever an induction line becomes available. The list is formed in descending order of $\frac{v_j}{p_j}$, where v_j is the number of distinct orders in tote $j \in J$. A neighbor solution is found through either a swap or an insertion. We randomly generate two tote indices and randomly select whether a swap or an insertion is

performed based on a probability γ. Insertion means the second tote is inserted in the position of the first tote. The search continues until τ drops below 0.01.

The parameters of the SA algorithm are determined experimentally. We test three values for the initial temperature parameter $\tau_0 = 5n$, $\tau = 10n$, and $\tau = 15n$; three values for the temperature cooling parameter $q = 0.01$, $q = 0.001$, and $q = 0.0001$; and five values for the swap probability $\omega = 0$, $\omega = 0.25$, $\omega = 0.5$, $\omega = 0.75$, and $\omega = 1$. We use a problem setting with four lines, 50 totes, 100 orders, and select processing times of totes from the interval [10, 40]. We generate 200 random instances and run the SA algorithm for each parameter combination. The best setting found is $\tau_0 = 5n$ and $q = 0.001$. We find that swap probabilities of 0.75 and 1 lead to the best results. As such, we set ω to 0.85.

7. Computational Results

We perform extensive computational testing to assess the performance of the branch and price and the simulated annealing methods and to compare against the actual system. Coding is done in C#, mathematical models are solved using CPLEX 12.6, and testing is performed on a PC with 64-bit Windows 7 operating system with 8 GB RAM and a 3.6 GHz Intel i7-4790 processor. The test instances and algorithm implementation details for each algorithm are given in the corresponding sections.

7.1. The Performance of the Branch-and-Price Approach

To warm start the column-generation procedure, we devise a construction heuristic that determines schedules based on the relationship between totes. We consider that two totes are neighbors if there is at least one order that has products in both. Starting from an arbitrary tote, the heuristic sorts its neighbors in descending order of processing time. The neighbors are assigned one by one to the first available induction line. Once all neighbors are assigned, the last tote is picked as the new tote whose neighbors are identified, and the same assignment procedure is applied. We generate as many random solutions as the number of totes and then extract columns from these solutions to start the column generation. The dynamic programming algorithm used to solve the subproblem allows the generation of multiple columns. Aside from the most negative reduced cost one, which is required for the convergence of the bounds, we add up to 10 additional columns.

We generate instances randomly by varying the number of orders $|K| = 20, 40, 60, 80, 100$; number of totes $|J| = 10, 20, 30, 40, 50$; and number of induction lines $|J| = 2, 3, 4, 5, 6$. The difficulty of the instances depends on how orders are split among totes. When every order is contained in a single tote, the problem reduces to the parallel machine–scheduling problem, which we compare against in Section 6.2. In this section, we generate instances for the more general OCP. The maximum number of totes that an order is split over is denoted by γ. For each order, a random number of totes between one and γ is generated, and the order is split randomly over them. The processing times $p_j, j \in J$ are randomly generated in [20, 80]. Five instances are generated for each problem setting, and a time limit of 3,600 s is imposed for each instance.

Average results are presented in Table 3, in which each row is an average of five instances. The table provides the average number of nodes explored, the number of columns generated, the relative gap, defined as 100×(best found feasible solution − lower bound)/lower bound, and the computational time in seconds under different values of γ. The last three rows provide the minimum, average, and maximum over all rows, respectively. For the average CPU, the <0.001 and the >3,600 entries are counted as 0.001 and 3,600, respectively.

The relative gap is, on average, 1.76% and is at most 3.38%, which shows the effectiveness of the branch-and-price algorithm. The results show that higher values of γ lead to higher gaps and higher computational times. Also, as expected, as the number of nodes explored increases, so does the number of columns generated and the computational time. The branch-and-price algorithm only branches for instances with $\gamma = 2, 3$.

7.2. Comparison with Chen and Powell (1999b)

Being a special case of the order consolidation problem, the proposed branch-and-price methodology is applicable to the parallel machine–scheduling problem. In order to assess it, we compare it with the branch-and-price approach of Chen and Powell (1999b), which we implemented. We use the same stopping criteria of 1% relative gap and time limit of 3,600 s of CPU for both algorithms. We test on 125 instances corresponding to five instances of five problem sizes and five induction line settings of two, three, four, five, and six induction lines. Average results are presented in Table 4. For each, we report the lower bound (LB), upper bound (UB), the relative gap (GAP) calculated as 100×(best found feasible solution − lower bound)/lower bound, and the CPU time in seconds.

For all instances, both algorithms find an optimal solution within 1%, and in 35 instances, both find an optimal solution. Out of the 125 instances, both methods find the same solution in 112, our proposed approach

Table 3. Performance of the Branch-and-Price Algorithm

Problem			Nodes explored			Columns generated			Relative gap			CPU, s		
$\|J\|$	$\|K\|$	$\|L\|$	γ=1	$\gamma = 2$	$\gamma = 3$	$\gamma = 1$	$\gamma = 2$	$\gamma = 3$	γ=1	γ=2	γ=3	$\gamma = 1$	$\gamma = 2$	$\gamma = 3$
10	20	2	1	113.2	272.8	211.8	1,142.6	2,130.4	0	0.83	0.95	0.2	6.80	16.8
10	20	3	1	433.6	1,298.6	182.6	2,177.4	6,852	0	0.75	0.99	<0.001	15.60	59.6
10	20	4	1	1,106.2	3,003	143.4	4,511.8	12,812.6	0	0.84	0.99	<0.001	33	111.2
10	20	5	1	1,478.6	2,627	135.2	6,814.4	12,923.4	0	0.88	0.83	<0.001	38	90.2
10	20	6	1	299.8	2,420.4	149.6	1,241	9,284.4	0	0.61	0.88	<0.001	6.8	70.2
20	40	2	1	529.2	4,338.8	808.2	19,685.4	146,448.6	0.04	0.7	1.13	6	347.6	2,873.6
20	40	3	1	2,878	7,100.4	636.8	44,416.2	131,676	0.2	0.72	3.21	2.4	625.4	2,576.4
20	40	4	1	3,610	13,329.2	645.4	28,701.6	101,511.4	0.04	0.81	3.38	1.6	721.2	3,084.2
20	40	5	1	3,497.2	14,268.2	561	17,569.6	73,408.8	0.29	0.64	2.8	0.4	740.4	3,026.6
20	40	6	1	4,800.2	6,931	509.6	22,339.8	27,956.2	0.11	0.82	2.63	<0.001	914.60	2,215.8
30	60	2	1	146.8	472	2,029	14,531.6	65,495.4	0.01	0.79	2.36	45.20	783.4	>3,600
30	60	3	1	522.2	1,724.4	1,372.4	12,244.6	63,421.6	0.05	0.67	2.05	15	735.4	>3,600
30	60	4	1	841.8	2,457.6	1,265.8	9,387.6	41,721.4	0.06	0.71	2	8.6	727.8	>3,600
30	60	5	1	1	922.2	1,295.4	1,337.2	13,040.8	0.11	0.63	1.56	6.2	9.6	2,831.2
30	60	6	1	1.8	245.6	1,173.4	1,160.6	3,890.6	0.21	0.55	1.68	3.8	6.2	2,242.2
40	80	2	1	1	63.4	3,553.8	9,609.2	23,451.8	0.01	0.58	1.73	181	1,166	3,295.40
40	80	3	1	97	218	2,594.2	9,431.2	25,859.6	0.02	0.67	1.9	62.4	730.20	2,920.6
40	80	4	1	1	190.4	1,983.2	2,829.6	11,999.4	0.03	0.61	1.91	29	70.40	3,109.8
40	80	5	1	200.6	47.6	2,015.4	5,668.2	4,421.6	0.05	0.65	1.86	20.20	758.20	2,242.2
40	80	6	1	1.2	41	2,151.6	2,299.2	3,489.4	0.03	0.74	1.62	17.20	43.80	2,986.2
50	100	2	1	1	6	5,513.4	9,197.8	10,081.2	0.02	0.42	1.55	1,056.80	3,341	>3,600
50	100	3	1	1	18.2	3,418.8	4,406.6	6,766.6	0.02	0.47	1.47	315	667.6	2,491.4
50	100	4	1	1	29	3,074.4	4,115.8	6,717.6	0.03	0.44	1.66	175	406.4	3,080.8
50	100	5	1	1	2.6	3,020.6	3,665.2	3,407	0.07	0.42	1.46	122.60	327.4	>3,600
50	100	6	1	1	3.4	2,925.4	3,351	3,516.4	0.1	0.56	1.47	89.60	1,465.6	>3,600
Minimum	1	1	2.6	135.2	1,142.6	2,130.4	0	0.42	0.83	<0.001	6.2	16.8		
Average	1	822.62	2,481.23	1,654.82	9,673.41	32,491.37	0.06	0.66	1.76	86.33	587.54	2,436.98		
Maximum	1	4,800.20	14,268.20	5,513.40	44,416.20	146,448.60	0.29	0.88	3.38	1,056.8	3,341	>3,600		

Table 4. Comparison with Chen and Powell (1999b) on the PMS Problem

		Chen and Powell (1999b)				Proposed B&P			
$\|J\|$	$\|K\|$	LB	UB	GAP	CPU	LB	UB	GAP	CPU
10	20	1,398.28	1,398.48	0.0096	0	1,398.28	1,398.28	0	0.04
20	40	4,873.6	4,879.16	0.1348	1.56	4,873.6	4,879.64	0.136	2.08
30	60	10,393.24	10,400.36	0.0872	11.52	10,393.24	10,400.28	0.0864	15.76
40	80	18,500.84	18,505	0.0272	51.4	18,500.84	18,505	0.0272	61.96
50	100	26,285	26,295	0.0476	140.44	26,285	26,295	0.0476	351.8

finds a better solution in six, and Chen and Powell (1999b) find a better solution in seven. In terms of CPU time, Chen and Powell (1999b) is faster. This is expected because of the differences between the respective subproblems that allow Chen and Powell (1999b) to take advantage of the SWPT rule.

7.3. The Performance of the SA Algorithm

The SA algorithm is expected to be fast and to handle large-scale instances in order for it to be integrated within the systems of the industrial partner. However, we need to confirm that the solutions it provides are of high quality. For that purpose, we assess the quality of the SA solution against the lower bound obtained with the branch-and-price approach. Table 5 displays the average relative gap, calculated as $100^*[(UB - LB)/LB]$ with 100×(best found feasible solution – lower bound)/lower bound, and the CPU time in milliseconds for $\gamma = 1, 2, 3$. It is clear that SA provides very high-quality solutions in less than one second of CPU. In the next section, we solve industry-size instances with SA and compare with the actual system to assess the value of optimizing order consolidation on the system.

Table 5. Performance of the Simulated Annealing Algorithm

Instance			$\gamma = 1$		$\gamma = 2$		$\gamma = 3$	
$\|J\|$	$\|K\|$	$\|L\|$	Gap	CPU, ms	Gap	CPU, ms	Gap	CPU, ms
10	20	2	0	27.4	0.91	25.4	0.78	25.6
10	20	3	0	33.4	0.35	29.2	0.9	28.8
10	20	4	0	34.2	0.79	29	0.95	30.8
10	20	5	0	31.8	0.84	28.2	0.9	34.6
10	20	6	0	26.8	0.99	29	0.82	31.6
20	40	2	0	86	0.61	92.4	1.35	92.6
20	40	3	0	84	0.85	84	0.97	86.6
20	40	4	0	89.6	0.35	95.6	1.68	93.8
20	40	5	0	88	0.34	88.4	5.09	102.6
20	40	6	0.02	109.2	1.08	89	1.44	110.4
30	60	2	0	204.8	0.45	228	0.71	237.4
30	60	3	0	204.2	0.48	231	2.19	216.2
30	60	4	0	222.2	0.73	208.4	0.93	221.6
30	60	5	0	229.8	0.26	230.8	0.89	183.2
30	60	6	0	188.8	0.62	188.8	0.93	184.4
40	80	2	0	350.2	0.35	340	0.63	310.8
40	80	3	0	294.8	0.47	357.6	1	321.8
40	80	4	0	349.4	0.51	314.4	0.86	368.2
40	80	5	0	383.8	0.24	369.4	1.07	460.6
40	80	6	0	383.4	0.58	318.2	0.52	379.2
50	100	2	0.01	580	0.41	494.8	0.64	485.8
50	100	3	0	464.6	0.3	444.6	0.4	449.6
50	100	4	0	439.4	0.23	446.6	0.56	449.6
50	100	5	0	508.8	0.34	614.4	0.91	516
50	100	6	0.01	517.8	0.49	506.2	0.66	508.2
Minimum			0	26.8	0.23	25.4	0.4	25.6
Average			0.0016	237.30	0.54	235.34	1.11	237.2
Maximum			0.02	580	1.08	614.4	5.09	516

Table 6. Characteristics of Industry Instances

	Characteristics			Orders per tote			Totes per order			Items per tote			Processing time/tote		
Instance	Totes	Orders	Products	Max	Mean	Std. dev.	Max	Mean	Std. dev.	Max	Mean	Std. dev.	Max	mean	Std. dev.
W1	353	1,243	4,202	30	11.54	8.39	19	3.27	2.45	30	11.9	8.76	60	24	14
W2	374	1,384	6,702	30	17.27	11.1	27	4.66	3.36	30	17.9	11.76	60	33	19
W3	420	1,374	6,275	30	14.51	9.42	24	4.43	2.98	30	14.94	9.75	60	27	16
W4	368	1,373	5,646	30	14.71	10.47	23	3.94	2.73	30	18.34	10.97	60	28	16
W5	336	1,367	5,601	30	16.07	9.66	29	3.95	2.81	30	16.66	10.1	60	30	16
W6	257	851	2,978	30	11.12	8.76	26	3.36	3.18	30	11.58	9.26	60	25	16
W7	253	932	3,925	30	15.13	10.76	14	4.1	1.7	30	15.51	11.1	60	29	18
W8	377	989	4,667	30	11.81	9.66	25	4.5	3.25	30	12.3	10.2	60	27	18
W9	169	949	3,932	30	22.54	10.09	19	4.01	1.9	30	23.2	10.4	60	38	15
W10	121	267	1,234	30	9.37	7.22	20	4.24	4.21	30	10.2	8.3	60	22	14

Note. Max, maximum; Std. dev., standard deviation.

7.4. Analysis of Impact on System Performance

In this section, we analyze the effects of optimizing order consolidation on system performance. To do so, we extract 10 instances from the data provided by the industry partner. These instances are selected based on the quality of the data and on being representative of a typical system operation. The number of totes, orders, and products for the 10 instances as well as distribution characteristics of the data are given in Table 6. Other statistics on the order, tote, and product content are also provided in the table. In addition to order and tote data, each instance comes with the actual sequence in which the totes were sorted and orders consolidated. We compare this sequence to our optimized solution on order consolidation time and on cubby-usage time. When verifying the schedule operation of the induction lines corresponding to the sequences, we find that operation is sometimes interrupted by idle times, which may, for example, correspond to break times, operator availability, etc. In order to ensure fair comparison, we consider two scenarios when simulating the actual and optimized sequences. In scenario 1, we assume that all lines are available at all times. In this case, each sequence is used to assign totes to the next available line. In scenario 2, we use actual line availability with idle times from the real system, and the same sequences are used to assign totes to lines. The order consolidation time corresponds to the completion time of the last tote with products from the order. Cubby-usage time is the time between the first and last totes of the order. Given that the instances are of large size and that the goal is to analyze the improvement in system performance if the order consolidation operation is optimized, we compare with SA solutions. Any improvement in performance would be a lower estimate on that achieved when a proven optimal solution is used.

We present the average consolidation times (in minutes) per order under both scenarios in Table 7. The improvement is calculated as 100(current system − proposed solution)/current system. The improvement over the current system is indeed significant. For scenario 1, at least 6.74% improvement is achieved with an average of 28.77%. For scenario 2, these values are 5.96% and 19.9%, respectively. We compare the number of orders whose consolidation time improves (decreases), increases, or remains the same. This comparison is

Table 7. Comparison Between the Proposed Solution and Current System.

	Scenario 1			Scenario 2		
Instance	Proposed solution	Current system	Improvement, %	Proposed solution	Current system	Improvement, %
W1	14.66	25.13	41.66	20.91	31.51	33.64
W2	27.71	38.94	28.83	70.17	79.25	11.45
W3	26.01	37.6	30.84	63.61	80.96	21.42
W4	21.44	30.55	29.82	123.23	139.92	11.93
W5	21.56	32.22	33.1	23.81	32.19	26.03
W6	11.46	18.65	38.56	23.16	29.84	22.39
W7	17.77	26.85	33.83	21.26	30.27	29.76
W8	30.75	32.97	6.74	45.97	48.88	5.96
W9	18.93	21.52	12.04	23.12	26.65	13.24
W10	5.3	7.83	32.3	5.86	7.64	23.19
Average	19.56	27.22	28.77	42.11	50.71	19.9

Table 8. Orders with Improved, Increased, and Unchanged Completion Times

	Scenario 1			Scenario 2		
Instance	Improved	Increased	Unchanged	Improved	Increased	Unchanged
W1	1,020	209	14	918	309	16
W2	1,098	261	25	920	423	41
W3	1,099	258	17	959	384	31
W4	1,075	285	13	891	464	18
W5	1,102	248	17	979	373	15
W6	689	157	5	577	261	13
W7	771	151	10	722	195	15
W8	479	457	53	426	499	64
W9	579	332	38	605	315	29
W10	206	59	2	188	73	6
Total	8,118	2,417	194	7,185	3,296	248
Percentage	75.66	22.53	1.81	66.97	30.72	2.31

given in Table 8. About 75.66% and 66.96% of orders have improved completion times under scenarios 1 and 2, respectively.

Finally, we investigate the effect of optimizing the consolidation sequence on cubby usage. Once an order's first product is inducted, a cubby is assigned to the order, and that order seizes the cubby until all products are inducted. Table 9 compares the cubby-usage times in minutes between the proposed solution and the current system under scenarios 1 and 2. On average, 21.92% and 20.71% improvements are achieved in scenarios 1 and 2, respectively. The number of available cubbies is a significant limitation in the system. In order to ensure that there is always an empty cubby for an order, the current practice is to limit the number of orders in a wave by the number of cubbies. Hence, about 20% improvement in cubby usage may lead to change in this practice and a much more significant improvement in overall order completion times.

8. Conclusion

In this work, we use data from an e-warehouse to understand the system in detail and identify areas for improvement. We find that order completion is done through a fragmentation-consolidation process in which orders are fragmented based on product content to form pick lists, and then products are consolidated to constitute original orders. Being picked manually in small batches in containers called totes, the sequence in which batches are processed has a huge impact on the order-completion process. We focus on this problem, which we call the order consolidation problem, provide formulations, and propose several solution methodologies to obtain good quality solutions. Specifically, we define the OCP as a generalization of the parallel machine–scheduling problem, present mixed-integer programming formulations, and develop heuristic and exact solution methodologies based on simulated annealing and branch and price in which the subproblem is a modified single machine–scheduling subproblem. We provide a dynamic programming algorithm to solve it and devise a new branching rule. After comparing and validating the quality of the solutions obtained, we

Table 9. Comparison on Cubby Usage Time

	Scenario 1			Scenario 2		
Instance	Proposed solution	Current system	Improvement, %	Proposed solution	Current system	Improvement, %
W1	10.22	15.04	32.06	14.26	21.56	33.86
W2	22.22	28.28	21.44	34.56	40.35	14.36
W3	20.53	26.64	22.92	43.42	57.13	24.00
W4	16.65	20.96	20.57	77.33	88.05	12.07
W5	16.96	22.47	24.51	18.49	24.99	26.01
W6	7.26	10.48	30.70	12.97	16.74	22.54
W7	14.60	19.71	25.93	17.44	24.10	27.62
W8	22.05	23.57	6.46	32.35	35.83	9.73
W9	13.34	15.52	14.07	16.66	19.91	16.32
W10	3.82	4.80	20.49	4.15	5.22	20.47
Average	14.76	18.75	21.92	27.16	33.39	20.71

optimize the order consolidation process for the warehouse under study. Compared with the system in use, substantial improvements were achieved in order completion time (19.9%–28.77%), in the number of orders with improved fulfillment time (66.97%–75.66%), and in resource utilization at the order consolidation step (20.71%–21.92%). The findings are important for warehouse management and directly impact daily planning. As resources are allocated based on availability, about 20% improvement in put wall usage has implications on the number of operators assigned to it and the size and duration of the picking wave. In addition, a 70% improvement in order completion will surely have direct benefits on order delivery to customers.

The current study is entirely driven by data. At the start, there was no specific research problem in mind except for what the data would reveal. We ended up with an extension of the parallel machine–scheduling problem, for which we employed state-of-the-art techniques to solve it. One of the great advantages of data is their use in validating findings. In this specific case, the results are substantial in improving the internal warehouse planning processes in terms of better use of resources and in satisfying customer expectations in terms of reduced order-completion times. Future research could focus on data from a different operation in the warehouse, for example, picking, and on alternative solution approaches for the OCP.

References

Anagnostopoulos GC, Rabadi G (2002) A simulated annealing algorithm for the unrelated parallel machine scheduling problem. *Proc. 5th Biannual World Automation Congress* (IEEE, Piscataway, NJ), 115–120.

Balakrishnan N, Kanet JJ, Sridharan V (1999) Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Comput. Oper. Res.* 26(2):127–141.

Belouadah H, Potts CN (1994) Scheduling identical parallel machines to minimize total weighted completion time. *Discrete Appl. Math.* 48(3):201–218.

Boysen N, Stephan K, Weidinger F (2019) Manual order consolidation with put walls: The batched order bin sequencing problem. *EURO J. Transportation Logist.* 8(2):169–193.

Bozer YA, Sharp GP (1985) An empirical evaluation of a general purpose automated order accumulation and sortation system used in batch picking. *Material Flow* 2:111–131.

Bozer YA, Quiroz MA, Sharp GP (1988) An evaluation of alternative control strategies and design issues for automated order accumulation and sortation systems. *Material Flow* 4(4):265–282.

Bülbül K, Şen H (2017) An exact extended formulation for the unrelated parallel machine total weighted completion time problem. *J. Scheduling* 20(4):373–389.

Chan LMA, Muriel A, Simchi-Levi D (1998) Parallel machine scheduling, linear programming, and parameter list scheduling heuristics. *Oper. Res.* 46(5):729–741.

Chen F, Wang H, Qi C, Xie Y (2013) An ant colony optimization routing algorithm for two order pickers with congestion consideration. *Comput. Indust. Engrg.* 66(1):77–85.

Chen JF (2015) Unrelated parallel-machine scheduling to minimize total weighted completion time. *J. Intelligent Manufacturing* 26(6):1099–1112.

Chen ZL, Powell WB (1999a) A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. *Eur. J. Oper. Res.* 116(1):220–232.

Chen ZL, Powell WB (1999b) Solving parallel machine scheduling problems by column generation. *INFORMS J. Comput.* 11(1):78–94.

Chen ZL, Powell WB (2003) Exact algorithms for scheduling multiple families of jobs on parallel machines. *Naval Res. Logist.* 50(7):823–840.

De P, Ghosh JB, Wells CE (1994) Due-date assignment and early/tardy scheduling on identical parallel machines. *Naval Res. Logist.* 41(1):17–32.

De Koster R, Le-Duc T, Roodbergen KJ (2007) Design and control of warehouse order picking: A literature review. *Eur. J. Oper. Res.* 182(2):481–501.

Fanjul-Peyro L, Ruiz R (2010) Iterated greedy local search methods for unrelated parallel machine scheduling. *Eur. J. Oper. Res.* 207(1):55–69.

Gademann A, Van Den Berg JP, Van Der Hoff HH (2001) An order batching algorithm for wave picking in a parallel-aisle warehouse. *IIE Trans.* 33(5):385–398.

Graham RL (1966) Bounds for certain multiprocessing anomalies. *Bell System Tech. J.* 45(9):1563–1581.

Grosse E, Glock C, Ballester-Ripoll R (2014) A simulated annealing approach for the joint order batching and order picker routing problem with weight restrictions. Technical Report, Darmstadt Technical University, Darmstadt, Germany.

Gu J, Goetschalckx M, McGinnis LF (2007) Research on warehouse operation: A comprehensive review. *Eur. J. Oper. Res.* 177(1):1–21.

Henn S, Koch S, Wäscher G (2012) Order batching in order picking warehouses: A survey of solution approaches. Manzini R, ed. *Warehousing in the Global Supply Chain* (Springer, London), 105–137.

Hong S, Johnson AL, Peters BA (2012) Batch picking in narrow-aisle order picking systems with consideration for picker blocking. *Eur. J. Oper. Res.* 221(3):557–570.

Hsieh LF, Huang YC (2011) New batch construction heuristics to optimise the performance of order picking systems. *Internat. J. Production Econom.* 131(2):618–630.

Hwang H, Oh Y, Lee Y (2004) An evaluation of routing policies for order-picking operations in low-level picker-to-part system. *Internat. J. Production Res.* 42(18):3873–3889.

Jampani J, Mason SJ (2008) Column generation heuristics for multiple machine, multiple orders per job scheduling problems. *Ann. Oper. Res.* 159(1):261–273.

Janssenswillen G (2019) processmapR: Construct process maps using event data. Accessed January 14, 2020, https://www.bupar.net.

Jia J, Mason SJ (2009) Semiconductor manufacturing scheduling of jobs containing multiple orders on identical parallel machines. *Internat. J. Production Res.* 47(10):2565–2585.

Johnson ME (1997) The impact of sorting strategies on automated sortation system performance. *IIE Trans.* 30(1):67–77.

Kaplan S, Rabadi G (2013) Simulated annealing and metaheuristic for randomized priority search algorithms for the aerial refuelling parallel machine scheduling problem with due date-to-deadline windows and release times. *Engrg. Optim.* 45(1):67–87.
Kim DW, Kim KH, Jang W, Chen FF (2002) Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics Comput.-Integrated Manufacturing* 18(3–4):223–231.
Kowalczyk D, Leus R (2017) An exact algorithm for parallel machine scheduling with conflicts. *J. Scheduling* 20(4):355–372.
Lee JH, Yu JM, Lee DH (2013) A tabu search algorithm for unrelated parallel machine scheduling with sequence-and machine-dependent setups: Minimizing total tardiness. *Internat. J. Adv. Manufacturing Tech.* 69(9–12):2081–2089.
Liaw CF (2016) A branch-and-bound algorithm for identical parallel machine total tardiness scheduling problem with preemption. *J. Indust. Production Engrg.* 33(6):426–434.
Lin B, Jeng A (2004) Parallel-machine batch scheduling to minimize the maximum lateness and the number of tardy jobs. *Internat. J. Production Econom.* 91(2):121–134.
Lin SW, Ying KC (2015) A multi-point simulated annealing heuristic for solving multiple objective unrelated parallel machine scheduling problems. *Internat. J. Production Res.* 53(4):1065–1076.
Logendran R, Subur F (2004) Unrelated parallel machine scheduling with job splitting. *IIE Trans.* 36(4):359–372.
Logendran R, McDonell B, Smucker B (2007) Scheduling unrelated parallel machines with sequence-dependent setups. *Comput. Oper. Res.* 34(11):3420–3438.
Mason SJ, Chen JS (2010) Scheduling multiple orders per job in a single machine to minimize total completion time. *Eur. J. Oper. Res.* 207(1):70–77.
McNaughton R (1959) Scheduling with deadlines and loss functions. *Management Sci.* 6(1):1–12.
Meller RD (1997) Optimal order-to-lane assignments in an order accumulation/sortation system. *IIE Trans.* 29(4):293–301.
Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953) Equation of state calculations by fast computing machines. *J. Chemical Physics* 21(6):1087–1092.
Min L, Cheng W (1999) A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence Engrg.* 13(4):399–403.
Mokotoff E (2004) An exact algorithm for the identical parallel machine scheduling problem. *Eur. J. Oper. Res.* 152(3):758–769.
Mokotoff E, Chrétienne P (2002) A cutting plane algorithm for the unrelated parallel machine scheduling problem. *Eur. J. Oper. Res.* 141(3):515–525.
Mönch L, Balasubramanian H, Fowler JW, Pfund ME (2005) Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Comput. Oper. Res.* 32(11):2731–2750.
Pan JCH, Wu MH (2012) Throughput analysis for order picking system with multiple pickers and aisle congestion considerations. *Comput. Oper. Res.* 39(7):1661–1672.
Petersen CG, Aase G (2004) A comparison of picking, storage, and routing policies in manual order picking. *Internat. J. Production Econom.* 92(1):11–19.
Radhakrishnan S, Ventura JA (2000) Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. *Internat. J. Production Res.* 38(10):2233–2252.
Ratliff HD, Rosenthal AS (1983) Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Oper. Res.* 31(3):507–521.
Rauchecker G, Schryen G (2019) Using high performance computing for unrelated parallel machine scheduling with sequence-dependent setup times: Development and computational evaluation of a parallel branch-and-price algorithm. *Comput. Oper. Res.* 104:338–357.
Roodbergen KJ, Koster R (2001) Routing methods for warehouses with multiple cross aisles. *Internat. J. Production Res.* 39(9):1865–1883.
Russell ML, Meller RD (2003) Cost and throughput modeling of manual and automated order fulfillment systems. *IIE Trans.* 35(7):589–603.
Schutten J, Leussink R (1996) Parallel machine scheduling with release dates, due dates and family setup times. *Internat. J. Production Econom.* 46–47:119–125.
Serafini P (1996) Scheduling jobs on several machines with the job splitting property. *Oper. Res.* 44(4):617–628.
Shim SO, Kim YD (2008) A branch and bound algorithm for an identical parallel machine scheduling problem with a job splitting property. *Comput. Oper. Res.* 35(3):863–875.
Smith WE (1956) Various optimizers for single-stage production. *Naval Res. Logist. Quart.* 3(1–2):59–66.
Tavakkoli-Moghaddam R, Taheri F, Bazzazi M, Izadi M, Sassani F (2009) Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. *Comput. Oper. Res.* 36(12):3224–3230.
Theys C, Bräysy O, Dullaert W, Raa B (2010) Using a TSP heuristic for routing order pickers in warehouses. *Eur. J. Oper. Res.* 200(3):755–763.
Vallada E, Ruiz R (2011) A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *Eur. J. Oper. Res.* 211(3):612–622.
van Den Akker JM, Hoogeveen JA, van de Velde SL (1999) Parallel machine scheduling by column generation. *Oper. Res.* 47(6):862–872.
Van den Berg JP, Zijm W (1999) Models for warehouse management: Classification and examples. *Internat. J. Production Econom.* 59(1–3):519–528.
Villa F, Vallada E, Fanjul-Peyro L (2018) Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource. *Expert Systems Appl.* 93:28–38.
Weng MX, Lu J, Ren H (2001) Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *Internat. J. Production Econom.* 70(3):215–226.
Xing W, Zhang J (2000) Parallel machine scheduling with splitting jobs. *Discrete Appl. Math.* 103(1–3):259–269.