Boysen, Nils; Briskorn, Dirk; Füßler, David; Stephan, Konrad

Put it in the bag: Order fulfillment with a pocket sorter system

Naval Research Logistics (NRL)

Provided in Cooperation with:
John Wiley & Sons

Suggested Citation: Boysen, Nils; Briskorn, Dirk; Füßler, David; Stephan, Konrad (2023) : Put it in the bag: Order fulfillment with a pocket sorter system, Naval Research Logistics (NRL), ISSN 1520-6750, John Wiley & Sons, Inc., Hoboken, USA, Vol. 70, Iss. 8, pp. 858-877, https://doi.org/10.1002/nav.22137

This Version is available at:
https://hdl.handle.net/10419/288193

Received: 6 September 2022 | Revised: 6 June 2023 | Accepted: 12 June 2023

DOI: 10.1002/nav.22137

RESEARCH ARTICLE

WILEY

Put it in the bag: Order fulfillment with a pocket sorter system

Nils Boysen[1] | **Dirk Briskorn**[2] | **David Füßler**[1] | **Konrad Stephan**[1]

[1]Friedrich-Schiller-Universität Jena, Lehrstuhl für Operations Management, Jena, Germany
[2]Bergische Universität Wuppertal, Professur für BWL, insbesondere Produktion und Logistik, Wuppertal, Germany

Correspondence
Nils Boysen, Friedrich-Schiller-Universität Jena, Lehrstuhl für Operations Management, Carl-Zeiß-Straße 3, 07743, Jena, Germany.
Email: nils.boysen@uni-jena.de

Funding information
Deutsche Forschungsgemeinschaft, Grant/Award Number: Grant BO 3148/14-1

Handling Editor: Xin Chen

Abstract

Due to high real estate costs in urban areas, shop floor space is scarce in most brick-and-mortar stores. Maneuvering newly arrived merchandise through narrow aisles during shelf replenishment is time-consuming for the sales staff and impedes customers. Therefore, many retail chains nowadays aim for store-friendly shipments (SFS). By mirroring the layout of a store in the buildup of its dedicated shipments, the need for a zigzag movement through the store when replenishing shelves can be avoided. On the negative side, however, additional effort arises in the distribution centers. A suitable warehousing system to assemble SFS without excessive effort is a pocket (or pouch or bag) sorter, where each item is put into its separate bag. These bags, filled with items, are automatically transported while hanging from an overhead conveyor and can be sorted into any sequence before being delivered to the workstations that build SFS. This article investigates the assembly of SFS with a pocket sorter and presents scheduling procedures to enhance the efficiency of this process for a given set of store orders. We demonstrate that, despite its notorious complexity, the problem can be solved by simple decision rules with good performance. In a case study, we show that this approach can dramatically reduce the completion times of store orders, resulting in savings of more than 60% of the total working hours compared to a simple real-world policy. Another 30% of reduction can be obtained by standardized store layouts.

KEYWORDS

complexity, pocket sorter, retailing, scheduling, store-friendly shipments

1 | INTRODUCTION

In the face of intense e-commerce competition, retail chains worldwide are striving to streamline their order fulfillment processes (Melacini et al., 2018). One crucial factor, as emphasized in a recent survey paper on warehouse operations for retail chains (Boysen et al., 2021), is the implementation of store-friendly shipments (SFS), also known as store-specific shipment buildups. By pre-sorting incoming shipments of newly arrived merchandise, often stacked in roll cages, according to the specific store's layout, sales personnel can avoid zigzagging through the store and instead follow a clear route from shelf to shelf. Due to the high cost and scarcity of shop floor space in urban areas (Hübner et al., 2020), brick-and-mortar stores are typically densely packed with shelves. Consequently, navigating roll cages through the narrow aisles is time-consuming and physically demanding. SFS, on the other hand, reduce unproductive effort during shelf replenishment, resulting in savings on wage costs due to shorter process times and allowing sales personnel to dedicate more time to customer service. Moreover, maneuvering roll cages in a zigzag pattern during store hours can negatively impact customers and their shopping experience. A pocket sorter system is a warehousing solution that facilitates the assembly of SFS without excessive additional effort.

[Correction added on 13 Sep, 2023, after first online publication: Handling Editor information added.]

1.1 | Bulk picking with a pocket sorter

The order fulfillment process described below was observed at a warehouse operated by a logistics service provider that handles full-service order fulfillment for a mid-sized German retailer specializing in hunting equipment and traditional regional costumes. However, the following description is applicable to many retail chains that utilize a pocket sorter to service their stores

Pocket sorter systems, also referred to as bag or pouch sorter systems, are a relatively new technology used for sorting. In this system, individual items are placed in small bags that hang from overhead trolley conveyors. The sorter allows for the resequencing of bags using a complex network of switches and parallel lanes, enabling the retrieval of specific item sequences required for assembling SFS. The pocket sorter is an essential component of a bulk picking system (see Figure 1), which is also known as batch picking in the literature (see Boysen et al., 2021).

Initially, all unit loads containing homogeneous items of a specific stock keeping unit (SKU) are stored in an automated storage and retrieval system (ASRS). This can be a lift-and-shuttle system, as shown in Figure 1A, where the unit loads are bins. Alternatively, crane-operated high-bay racks (Boysen & Stephan, 2016) and carousel systems (Litvak & Vlasiou, 2010) can also be utilized as ASRS options. Orders are processed in waves, which are subsets of the total order set that are jointly sorted. Typically, each wave comprises one order per packing station. For a given wave, the unit loads containing the requested SKUs are retrieved from the ASRS and delivered to one or multiple parallel loading stations, as depicted in Figure 1B. In the loading stations, the current SKU is identified, usually by scanning a bar code, and the system indicates the total number of items of this SKU required for the current wave of orders. A human worker at the loading station retrieves the specified number of items from the unit load and places each item into subsequent bags of the sorter. Once all items of the current SKU have been loaded, the corresponding unit load is returned to the ASRS, and the loading process repeats with the next SKU. The loaded items are transported in bags along a trolley conveyor. Along the way to the packing stations, the bags pass an intermediate buffer, which consists of a complex system of switches and parallel lanes, as shown in Figure 1C. The buffer is used to redirect bags from the main conveyor, resort them, and channel them back onto the main conveyor. This enables sorting the items into the specific sequence required by each individual store. Therefore, the store's layout is translated into a sequence of items, and these items are retrieved from the pocket sorter system in that sequence. After passing the intermediate buffer, the main conveyor carries the bags towards the packing stations, which have a similar setup to the loading station depicted in Figure 1B. At the packing stations, another logistics worker sequentially retrieves the items from their bags in the designated sequence and places them onto a load carrier. Many retail chains utilize roll cages, as shown in Figure 1D. With the items already arriving in the right sequence, the logistics worker only needs to pack them one after another into the roll cages to obtain the SFS. Once an SFS is completed, it is delivered to the shipping area and loaded onto a truck for transportation to the respective store. For an alternative description of pocket sorters and their application in bulk picking, refer to Boysen et al. (2021).

When implementing a pocket sorter in a bulk picking environment, it is essential to determine the dimensions of the main system elements (see Figure 1). Specifically, factors such as the storage and throughput capacity of the ASRS, the number of parallel loading stations, the available number of bags, the capacity of the intermediate buffer, and the number of packing stations need to be decided upon. Moreover, all these decisions must be coordinated to prevent bottleneck stages that could impede the overall system throughput capacity. In this article, we assume a predefined system layout. Our research focuses on addressing the operational decisions involved in processing a given wave of orders for different stores. The key decisions include: (i) determining the loading sequence of SKUs, specifying the order in which they are retrieved from the ASRS and placed into bags at the loading station, (ii) managing the buffer operations that involve exchanging items to and from the intermediate buffer, resulting in a sequence of items approaching the packing stations, and (iii) assigning the items approaching the packing

(A) Storage of unit loads (shuttle warehouse of psb intralogistics)
(B) Loading of bags (Pocket sorter system of Dürkopp)
(C) Sorting of bags (Pouch Pocket of psb intralogistics)
(D) SSSBs in roll cages (roll cage of Huameilong Metal Products)

FIGURE 1 Bulk picking process with pocket sorter system.

stations to specific store orders. In the following section, we will review the relevant literature on this topic.

1.2 | Literature review

Warehousing plays a crucial role in every supply chain, and it comes as no surprise that research in this field has a long-standing tradition. Rather than attempting to summarize the vast body of literature on warehousing, which encompasses various picker-to-parts and parts-to-picker systems, we recommend referring to the most relevant review papers by De Koster et al. (2007), Gu et al. (2007, 2010), and Van Gils et al. (2018). These survey papers provide comprehensive insights into warehousing in general. Other surveys focus on specific aspects within this field, such as robotized warehousing systems (Azadeh et al., 2019), ASRSs (Boysen & Stephan, 2016; Roodbergen & Vis, 2009), automated sorter systems (Boysen, Briskorn, et al., 2019), warehousing systems for e-commerce (Boysen, de Koster, & Weidinger, 2019), and for brick-and-mortar retail chains (Boysen et al., 2021). Only the last two surveys mention pocket sorters and their increasing real-world application in recent years. However, both surveys note the absence of scientific papers specifically focusing on pocket sorter systems. This finding aligns with our own (unsuccessful) literature search. As there is no existing research on this system, we survey related optimization problems with similar structures: (a) resequencing mixed-model assembly lines, (b) shunting freight cars in railway shunting yards, and (c) coordinating picking and order consolidation.

(a) In car manufacturing, production sequences often become disordered, for example, due to paint defects in the paint shop, and consequently need to be reestablished in the original sequence. Resequencing buffers are employed for this purpose. For a comprehensive overview of this domain, refer to Boysen, Scholl, and Wopperer (2012); optimization approaches for this task have been developed by researchers such as Lahmar and Benjaafar (2007), Lim and Xu (2009), and Boysen et al. (2011). The main distinction between these areas lies in the fact that the final item sequence of the pocket sorter is divided into multiple subsequences, whereas a mixed-model assembly line requires a single production sequence.

(b) Similar sorting problems also arise in shunting yards, where inbound freight cars need to be organized into outbound freight trains. In this context, inbound trains, with freight cars arranged in a specific sequence, are maneuvered down a shunting hill and directed into multiple tracks. This process allows the assembly of outbound freight trains while adhering to sequence restrictions. A survey paper on shunting processes is available from Boysen, Fliedner, et al. (2012); algorithms for freight car sorting are presented by researchers such as Daganzo et al. (1983) and Jacob et al. (2011). The primary distinction between this domain and a pocket sorter lies in the absence of a buffer, with multiple successive movements over a shunting hill being performed instead.

(c) The optimization problem discussed in this article, pertaining to the pocket sorter, encompasses multiple hardware components (including loading and unloading stations, as well as an intermediate buffer) and involves multiple decisions (specifically, decisions (i) to (iii) outlined in Section 1.1). Consequently, it can be categorized as a combined warehousing problem, which Van Gils et al. (2018) extensively cover in their survey. The existing literature on combined problems typically addresses the coordination between an initial product retrieval stage and a subsequent order consolidation stage. Examples of such problems include batch picking and sorting (e.g., Gallien & Weber, 2010), order picking and packing (e.g., Zhong et al., 2022), as well as order picking and delivery (e.g., Zhang et al., 2019). Although our optimization problem falls within this category, it focuses on a highly specific setup. Instead of the conventional picker-to-parts order picking process in the first stage, we consider a parts-to-picker bulk picking process. Moreover, the second stage does not involve a traditional conveyor-based sorter system (as discussed in Boysen, Briskorn, et al., 2019), but rather utilizes a pocket sorter to facilitate SFS. This unique combination has not been previously addressed in the literature.

In conclusion, it can be inferred that the pocket sorter scheduling problem (PSS) presented in this article has not yet been explored in existing literature, and the related optimization problems discussed in previous research exhibit substantially different structures.

1.3 | Contribution and article structure

This article aims to extract the basic decision problem to be solved when operating a pocket sorter in a bulk picking environment. The result is an optimization problem supporting three interdependent decisions: (i) The loading sequence of SKUs at a loading station, (ii) the buffer operations exchanging items to and from the intermediate buffer, and (iii) the assignment of items to store orders. We formulate the resulting combined optimization problem and prove structural properties. Furthermore, we derive exact and heuristic solution methods for solving this optimization problem.

Once these solution methods are available (and proven to have a good performance), we apply them to investigate managerial aspects. First, we answer the question whether

sophisticated optimization of the operational pocket sorter processes enables considerable performance improvements compared to simple scheduling policies like they are, for instance, applied by the warehouse of our case study. Our results indicate that more than 60% of the total working hours in the packing stations can be saved. Furthermore, we evaluate the benefit resulting from retail chains with identical (or at least similar) store layouts on all their brick-and-mortar sales outlets. Our computational study indicates further potential performance gains of up to 30%. Finally, we show that the problem, although involving multiple decisions and being notoriously complex, can be solved with astonishingly good performance by a very simple solution approach based on priority rules.

The remainder of the article is structured as follows. First, Section 2 defines the basic PSS problem and Section 3 provides a mixed integer program (MIP). Then, Section 4 analyzes the computational complexity and proves several structural properties for deriving an efficient heuristic solution procedure. This heuristic is detailed in Section 5 and its computational performance is investigated in Section 6. The latter section also investigates the managerial issues outlined above. Finally, Section 7 concludes the article.

2 | PROBLEM DESCRIPTION

For a first more intuitive understanding, we start with a verbal description of the problem, an example, and our basic assumptions in Section 2.1. A formal problem definition is given in Section 2.2.

2.1 | Problem characterization and assumptions

We have a single loading station, manned with a logistics worker, where one SKU after another arrives from the ASRS in unit loads. We call the sequence in which the SKUs are processed the *loading sequence*. The worker puts the total demand for items of each SKU of the current wave into consecutive bags. We normalize time to (equidistant) time slots each representing the (average) time span it takes the logistics worker to put one item into a bag. Loading an item consists of retrieving the item from the unit load, scanning the item, and putting it into a bag. Depending on the specific setup of the loading station and the characteristics of the SKUs to be handled, the average slot time varies between 10 and 20 s (SSI Schäfer, 2021).

The items loaded into bags are transported along a trolley conveyor that connects the loading station with the packing stations, where store orders are gathered and prepared for packing. Upon leaving the loading station, we encounter what we refer to as the *before-buffer sequence*, where all the requested items of the first SKU are followed by all the requested items of the second SKU, and so on (as determined by the loading sequence). On their way, the bags pass through a switch that connects to an intermediate buffer area. In this buffer, bags can be temporarily stored and later reintroduced into the sequence. When a loaded bag enters the buffer, it frees up a slot that can be occupied by another item reentering the trolley conveyor from the buffer. If no empty slot is available, an item can still be reintroduced, causing a delay of one slot for all subsequent items. The resulting sequence is referred to as the *after-buffer sequence*. This sequence continues forward and eventually reaches a switch that connects to parallel packing stations. Each packing station handles a dedicated store order from the current wave. Each store order specifies the items to be packed in a specific sequence, known as the *packing sequence*. This sequence enables the logistics worker at the respective packing station to assemble the SFS according to the layout of the store by placing one item after another onto the shipping carrier. The process involves three interdependent decisions, which are illustrated in Figure 2.

Specifically, we need to make decisions regarding:

1. The loading sequence, which determines the order in which SKUs are retrieved from the ASRS and incorporated into the before-buffer sequence of items at the loading station.
2. The buffer operations, which involve determining whether an item should be moved into the buffer and, if so, when it should be reintroduced into the after-buffer sequence.
3. The assignment of items to orders.

Our objective is to minimize the total completion time of the orders, which is measured by the slot in which the last item dedicated to each order arrives at the packing station. To gain a better understanding of our PSS problem, we consider the example illustrated in Figure 3.

Example: The current wave consists of three store orders each having a given packing sequence. Store 1, for instance, requires its items being delivered in packing sequence (B, B, C, A). In total, the wave of orders requires two items of SKU A, four items of SKU B, and four items of SKU C. Solution one in Figure 3A has loading sequence (C, B, A), so that the first four slots of the before-buffer sequence contain the four demanded items of SKU C. In the before-buffer sequence, the items approach the switch to and from the buffer. We have the following buffer operations.

- First, we examine the buffer decisions made in the red-marked slots of solution one in Figure 3A: (i) The marked item of SKU C is moved into the buffer, and (ii) since no other SKU is present in the buffer, this slot remains empty in the after-buffer sequence. Later on, (iii) the item of SKU B is moved into the buffer, and (iv) this empty slot is taken by the item of SKU C that was previously moved into the buffer.
- If there is no empty slot available to accommodate a reinserted item, all subsequent items in the following slots are delayed by one slot. This buffer operation is demonstrated by the green-marked slots of solution two depicted

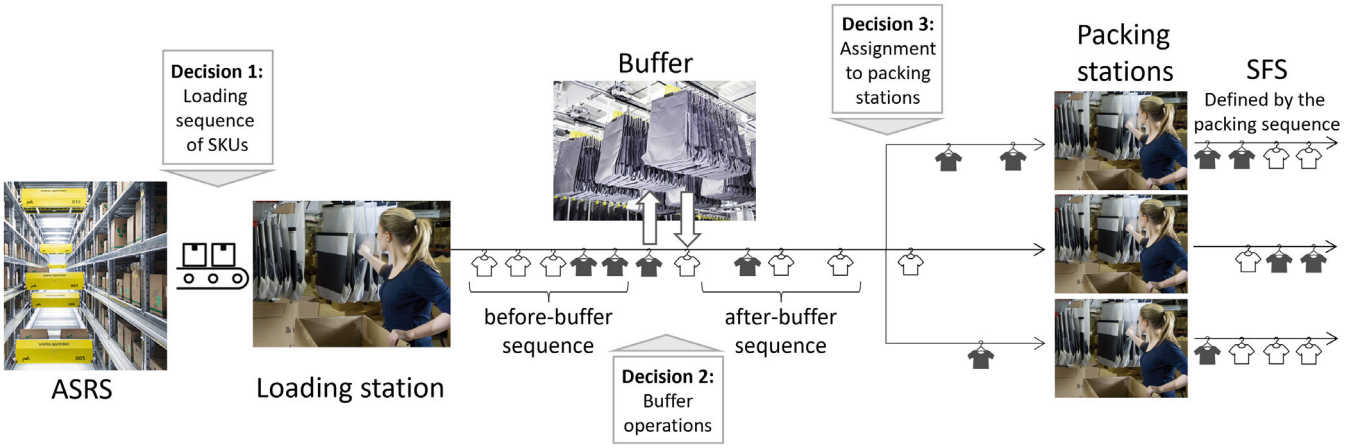

FIGURE 2 Pocket sorter process and associated decisions.

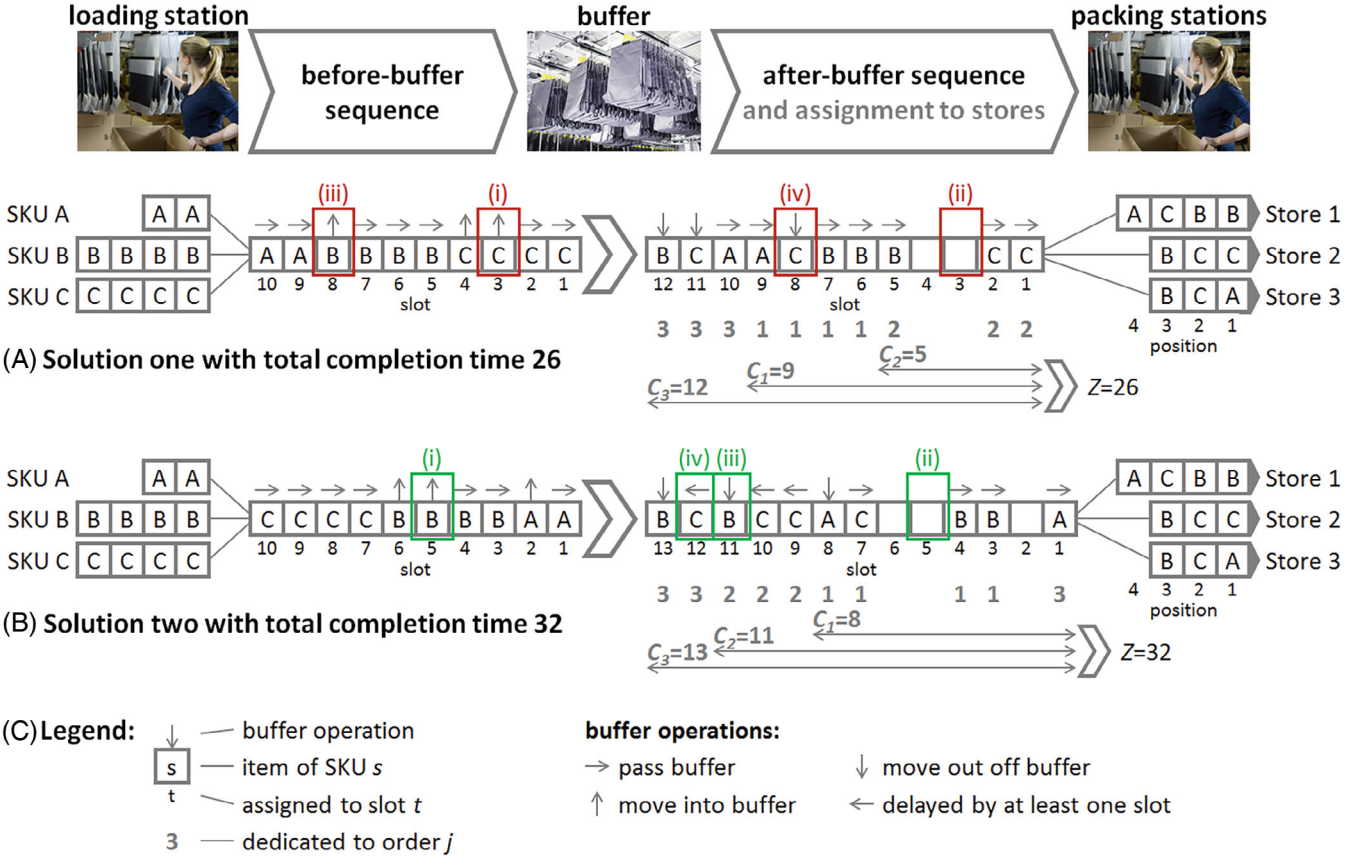

FIGURE 3 Example instance for our pocket sorter scheduling problem with two alternative solutions.

in Figure 3B: (i) First, the marked item of SKU B is moved into the buffer, and (ii) its slot cannot be taken by the item of SKU A in the buffer because this item is required by Store 1 after SKU C (which has not yet passed the buffer); hence, the slot remains empty. Later, (iii) this item is reinserted into the sequence, but there is no available empty slot at its original position, resulting in (iv) a delay of one slot for all subsequent items.

Finally, we have the assignment of items from the after-buffer sequence to orders, indicated by the gray numbers referring to the respective store. The completion time of an order is reached when the last item of that order arrives. For example, in Figure 3A, the order for Store 2 is completed after slot five, resulting in a sum of completion times of 26 for solution one. On the other hand, solution two in Figure 3B has an objective value of 32.

Our PSS problem is based on several simplifying assumptions and prerequisites, which we discuss below.

- We aim to address an elementary operational decision problem that encompasses all essential problem characteristics. Therefore, we consider the most basic setup, where we have a single loading station supplying multiple packing stations, with each packing station processing exactly one order. In typical scenarios, the loading process is

much faster than the packing process, allowing a single loading station to support multiple packing stations. For larger systems with a greater number of packing stations (e.g., 80 packing stations as reported in Boysen, de Koster, and Weidinger, 2019), multiple parallel loading stations may be required. This modification slightly alters the problem but does not change the fundamental objectives and constraints. However, a detailed analysis of the implications is left for future research.

- We assume that the ASRS supplying the loading station is not a bottleneck resource. More complex situations, where certain loading sequences impose additional retrieval efforts on the ASRS, leading to waiting times for the worker at the loading station, are left for future investigation. Furthermore, we assume that the ASRS can retrieve the SKUs in any sequence and is not restricted by the loading sequence.
- We assume that the buffer area is empty before and after processing each wave. There are large-scale pocket sorter systems with up to 500 000 bags (Boysen, de Koster, & Weidinger, 2019), where items are stored because they are anticipated to be ordered in the near future. These systems are primarily utilized by e-commerce retailers, such as Germany's largest online fashion retailer, Zalando. However, for brick-and-mortar retail chains, which typically have a smaller product assortment compared to e-commerce retailers (Boysen et al., 2021), smaller systems with the bulk picking setup described earlier are more common. Our focus is on these smaller systems.
- We assume that the buffer size is not a limiting factor and that there are always enough bags and buffer positions available. Given the previous assumption, systems with a few hundred bags should be sufficient. Moreover, bags can be stored in a compact manner on overhead conveyors, so the availability of bags and buffer positions is rarely a concern.
- We do not explicitly model the item movement and sorting process within the buffer. We assume that the system can efficiently deliver a specific item for a particular slot whenever needed. Pocket sorters are manufactured by specialized suppliers such as Dürkopp and psb intralogistics (see Figure 1), and they also provide the software to control the movement of bags within the buffer. From the perspective of distribution centers, the buffer is essentially a black box with an interface for retrieving items. Therefore, our problem setting only involves decisions made by the distribution center. Optimizing the flow of bags within the buffer is an interesting decision task in itself but is not addressed in this article.
- We measure time in (equidistant) slots, where each slot represents the average loading time of bags at the loading station. While the actual loading time may slightly vary from slot to slot, we use average times since these variations are difficult to predict when planning the process in advance.
- Our objective is to minimize the sum of order completion times. Completing orders as early as possible offers several advantages. First, the earlier an order is packed and ready, the higher the likelihood that it will reach its store on time. Additionally, logistics workers who have already completed their orders can be assigned to other tasks. In our logistics provider's multi-client warehouse, for example, idle workers are promptly reassigned to other duties via a pager once they finish an order (see Section 6.3).

2.2 | Problem definition

We are given a set $J = \{1, \ldots, |J|\}$ of store orders as input for our decision process. Each order $j \in J$ is defined by a packing sequence σ_j of SKUs and a number $n_{j,s}$ for each SKU s in σ_j. The packing sequence σ_j determines the order in which SKUs are placed on the SFS designated to the store of order j, with $\sigma_j(k)$ being the kth SKU in σ_j. The set S_j comprises the SKUs required by order $j \in J$, and set $S = \bigcup_{j \in J} S_j$ includes all the relevant SKUs for the current wave. As an example, we consider the packing sequence of store 1 shown in Figure 3: $\sigma_1 = (B, C, A)$, $n_{1,B} = 2$, $n_{1,C} = n_{1,A} = 1$. The total number of items of SKU $s \in S$ as n_s, and the overall number of items as n. Corresponding to the three aforementioned components of our decision process, a solution to our problem consists of three parts.

1. The first component is the loading sequence σ^l of SKUs in S, which prescribes the retrieval sequence of unit loads and the loading of the demanded items per SKU into bags at the loading station. Note that loading sequence σ^l differs from packing sequence σ_j, $j \in J$, with the latter being given as an input. Both σ^l and σ_j, $j \in J$, however, reflect sequences of SKUs. We refer to the kth SKU in σ^l by $\sigma^l(k)$. Since for each SKU multiple items may be required, the loading sequence directly translates into the before-buffer sequence π^{in}, where each sequence position refers to a single item and all items of the same SKU occur in direct succession. Note that the latter is not an artificial add-on assumption to ease the optimization process but a characteristic that is inevitable in a bulk picking process (see Section 1.1).
2. The second component is the after-buffer sequence π^{out} of items after passing the buffer, where buffer operations (i.e., movements in and out of the buffer as well as delays of subsequent items) are executed. The buffer operations turn the before-buffer sequence into the after-buffer sequence, where empty slots can only appear in the latter sequence. Note that after-buffer sequence π^{out} differs from before-buffer sequence π^{in}. Both π^{out} and π^{in}, however, reflect sequences of items.

3. The third component is an assignment a of slots of the after-buffer sequence π^{out} to $J \cup \{0\}$ signaling that the slot is empty if it is assigned to 0 or defining the SFS the item in the slot is put on (and, thus, the order in which it ends up) otherwise. Recall that an assignment a directly implies the assignment of items to packing stations, because each order is processed in its dedicated packing station.

While each component of our solution concept is rather straightforward, we should argue that we appropriately represent the buffer operations by having loading sequence σ^l of SKUs and after-buffer sequence π^{out} of items. In particular, we should state necessary and sufficient conditions for π^{out} being obtainable from σ^l by buffer operations.

Lemma 1. *After-buffer sequence π^{out} can be obtained from loading sequence σ^l by resequencing in the intermediate buffer if and only if the first item of each SKU s, where $s = \sigma^l(k)$ for some $k = 2, \dots, |S|$, is in a slot $l \geq f_s = \sum_{k'=1}^{k-1} n_{\sigma^l(k')} + 1$ of π^{out}.*

Proof. First, recall that loading sequence σ^l implies before-buffer sequence π^{in}. Sequence π^{in} contains all items required by orders in J with items of the same SKU being clustered and items of SKU $\sigma^l(k)$ preceding items of $\sigma^l(k+1)$ for each $k = 1, \dots, |S| - 1$.

If the first item of SKU s with $s = \sigma^l(k)$ for some $k = 2, \dots, |S|$, is in a slot $l < f_s$ of π^{out}, it cannot be achieved from σ^l since it would be in an earlier slot in π^{out} than in π^{in}. However, the pocket sorter cannot accelerate items, that is, bring them to an earlier slot, during resequencing.

If the first item of SKU s with $s = \sigma^l(k)$ for some $k = 2, \dots, |S|$, is in a slot $l \geq f_s$, then there are at most $\min\{n_s, l' - l + 1\}$ items of SKU s in slots $l, \dots, l'$, $l' = l, \dots, n$, of π^{out}. In π^{in}, there are $\min\{n_s, l - f_s + (l' - l + 1)\}$ items of SKU s in slots $1, \dots, l'$. Clearly,

$$\min\{n_s, l' - l + 1\} \leq \min\{n_s, l - f_s + (l' - l + 1)\},$$

since $l \geq f_s$. So, for each slot l', $l' = 1, \dots, n$, there are at least as many items of each SKU $s \in S$ in slots $1, \dots, l'$ of π^{in} as in slots $1, \dots, l'$ of π^{out}.

It remains to argue that the resequencing operations can actually arrange π^{out} from σ^l. It is not hard to see that we can simply process π^{in} slot by slot and

- keep the current slot l as is, if an item of the same SKU occupies both, the lth slots of π^{in} and π^{out},
- remove the item from the current slot l of π^{in} and leave the slot empty in π^{out}, if the lth slots of π^{out} is empty, or
- remove the item from the current slot l of π^{in} and fill it with an item of the SKU in the lth slot of π^{out} if both, the lth slots of π^{in} and π^{out} are occupied but differ in the SKU.

While the first two cases can unconditionally be implemented, replacing items in the third case relies on an item of the replacing SKU to be in buffer. However, we can take this for granted due to the above. ∎

We say a solution is *buffer-feasible*, if the condition stated in Lemma 1 is satisfied. That is, the first item of each SKU $s = \sigma^l(k) \in S$, $k = 2, \dots, |S|$, is in a slot $l \geq f_s$ of π^{out}.

Also, after-buffer sequence π^{out} and the assignment a of slots to orders need to be geared to each other. We will say in the following that both, a non-empty slot and the item in the slot, are assigned to an order. First, each order $j \in J$ must get assigned exactly $n_{j,s}$ items of each SKU $s \in S_j$. Furthermore, the sequence of SKUs in σ_j has to be respected. For each $k = 1, \dots, |S_j| - 1$, all items of SKU $\sigma_j(k)$ assigned to j must precede all items of SKU $\sigma_j(k+1)$ assigned to j in π^{out}. If both conditions are satisfied, we call a solution *order-feasible*.

Definition 1. A solution is called feasible if it is buffer-feasible and order-feasible.

For a feasible solution the completion time C_j of order $j \in J$ equals the slot number of the last item in π^{out} assigned to store order j. The sum of order completion times amounts to $\sum_{j \in J} C_j$.

Definition 2. The PSS is to determine, among all feasible solutions, one that minimizes the sum of order completion times.

Finally, we prove an optimality property.

Lemma 2. *In an optimal solution to PSS, every slot of the after-buffer sequence that is larger than $n - \min_{s \in S}\{n_s\}$ contains an item unless all subsequent slots are empty.*

Proof. Consider a solution to PSS where the after-buffer sequence π^{out} contains an empty slot at a position $k \geq n - \min_{s \in S}\{n_s\} + 1$, which is followed by a slot that is occupied by an item. We define an alternative after-buffer sequence $\widetilde{\pi}^{\text{out}}$ by deleting the empty slot k (i.e., by shifting all subsequent items one slot to the front). Obviously, $\widetilde{\pi}^{\text{out}}$ still meets the feasibility criterion of Lemma 1 and results in a smaller objective value than π^{out}. Thus, the initial solution cannot be optimal. ∎

3 | A MIXED INTEGER PROGRAM

This section develops a MIP for PSS. Since PSS is a challenging problem including three interdependent decisions, solving even small-sized instances with a few dozen items turns out as a challenging task for an off-the-shelf solver. Therefore, we present some further ideas on how to streamline the solution process. Specifically, we introduce a preprocessing step (see Appendix A), the MIP model itself, which we dub PSS-MIP (see Section 3.1), valid inequalities to strengthen the MIP (see Section 3.2), and an approach to reduce the time horizon to be considered by the MIP (see Section 3.3). The notation applied throughout this section is summarized in Table 1.

3.1 | Mixed integer model PSS-MIP

Applying the notation summarized in Table 1, PSS-MIP consists of objective function (1) and constraints (2) to (18). We employ three types of binary variables. Binary variable $\sigma^l_{s,k}$, reflects whether SKU s is the kth SKU in the loading sequence ($\sigma^l_{s,k} = 1$) or not ($\sigma^l_{s,k} = 0$). The after-buffer sequence is represented by binary variable $\pi^{\text{out}}_{s,l}$ signaling whether an item of SKU s is in the lth slot of the after-buffer sequence ($\pi^{\text{out}}_{s,l} = 1$) or not ($\pi^{\text{out}}_{s,l} = 0$). Finally, binary variable $a_{j,l}$, represents whether the lth slot of the after-buffer sequence is assigned to order j ($a_{j,l} = 1$) or not ($a_{j,l} = 0$). In total, we have $O(|S|^2 + (|S| + |J|) \cdot L)$ binary variables with L being the maximum length of the after-buffer sequence. Additionally, we employ continuous variable C_j representing the completion time of order j.

$$\text{Minimize } Z(\sigma^l, \pi^{\text{out}}, a, C) = \sum_{j \in J} C_j. \tag{1}$$

Objective function (1) represents the goal to minimize the sum of completion times and is subject to the following constraints, which we introduce in segments.

$$\sum_{s \in S} \sigma^l_{s,k} = 1 \qquad \forall k = 1, \dots, |S|, \tag{2}$$

$$\sum_{k=1}^{|S|} \sigma^l_{s,k} = 1 \qquad \forall s \in S, \tag{3}$$

$$\sum_{s \in S} \pi^{\text{out}}_{s,l} \leq 1 \qquad \forall l = 1, \dots, L, \tag{4}$$

$$\sum_{l=1}^{L} \pi^{\text{out}}_{s,l} = n_s \qquad \forall s \in S, \tag{5}$$

$$\sum_{j \in J} a_{j,l} \leq 1 \qquad \forall l = 1, \dots, L, \tag{6}$$

$$\sum_{l=1}^{L} a_{j,l} = \eta_j \qquad \forall j \in J. \tag{7}$$

Constraints (2) to (7) ensure a well-defined loading sequence σ^l (constraints (2) and (3)), after-buffer sequence π^{out} (constraints (4) and (5)), and assignment a (constraints (6) and (7)).

$$\sum_{k'=1}^{k-1} \sum_{s' \in S} n_{s'} \cdot \sigma^l_{s',k'} + 1 - L \cdot (2 - \sigma^l_{s,k} - \pi^{\text{out}}_{s,l}) \leq l$$
$$\forall\ l = 1, \dots, L; s \in S; k = 1, \dots, |S|. \tag{8}$$

Inequalities (8) ensure that the feasibility criterion according to Lemma 1 is satisfied. Retrieving an SKU s at position k of the loading sequence and simultaneously using it in slot l of the after-buffer sequence (which implies $L \cdot (2 - \sigma^l_{s,k} - \pi^{\text{out}}_{s,l}) = 0$) is only possible, if l is sufficiently large, that is, if l allows

TABLE 1 Notation for PSS.

J	Set of orders (index j)
S	Set of SKUs (index s)
S_j	Set of SKUs required by order j (index s)
L	Maximum length of the after-buffer sequence
n_s	Total number of items of SKU $s \in S$
n	Total number of items over all SKUs: $n = \sum_{s \in S} n_s$
$n_{j,s}$	Number of items of SKU $s \in S_j$ that are required by order j
η_j	Number of items that are required by order j
σ_j	Packing sequence of SKUs required by order j
$\sigma_j(k)$	The kth SKU of order j's packing sequence
$\sigma_j^{-1}(s)$	Position where SKU s occurs in the packing sequence σ_j of order j
$l(\sigma_j)$	Number of different SKUs required by order j
$s_{j,k}$	The SKU of the kth item required by order j
$\sigma^l_{s,k}$	Binary variable: 1, if SKU s is the kth SKU in the loading sequence; 0, otherwise
$\pi^{\text{out}}_{s,l}$	Binary variable: 1, if an item of SKU s is in the lth slot of the after-buffer sequence; 0, otherwise
$a_{j,l}$	Binary variable: 1, if the lth slot of the after-buffer sequence is assigned to order j; 0, otherwise
C_j	Continuous variable: completion time of order j

for feeding all items of the first $k-1$ SKUs of the loading sequence into the sorter before processing the first item of SKU s.

$$a_{j,1} \leq \pi^{\text{out}}_{s_{j,1},1} \quad \forall\, j \in J, \tag{9}$$

$$\underbrace{1-\overbrace{(l-k+1)}^{\text{big M}}\cdot(1-a_{j,l})}_{\text{term (i)}} + \underbrace{\left(\sum_{l'=1}^{l-1} a_{j,l'} - (k-1)\right)}_{\text{term (ii)}} \leq \eta_j \cdot \tag{10}$$

$$\sum_{k'=k}^{\eta_j} \pi^{\text{out}}_{s_{j,k'},l} \quad \forall \begin{array}{c} j \in J; \\ l = 2, \ldots, \left\lfloor \frac{L}{2} \right\rfloor; \\ k = 1, \ldots, \min\{l, \eta_j\}, \end{array}$$

$$\underbrace{1-\overbrace{k}^{\text{big M}}\cdot(1-a_{j,l})}_{\text{term (i)}} - \underbrace{\left(\sum_{l'=1}^{l-1} a_{j,l'} - (k-1)\right)}_{\text{term (ii)}} \leq \eta_j \cdot$$

$$\sum_{k'=1}^{k} \pi^{\text{out}}_{s_{j,k'},l} \quad \forall \begin{array}{c} j \in J; \\ l = 2, \ldots, \left\lfloor \frac{L}{2} \right\rfloor; \\ k = 1, \ldots, \min\{l, \eta_j\}. \end{array} \tag{11}$$

Inequalities (9) to (11) link the entries of assignment vector a to after-buffer sequence π^{out} for the small sequence positions $l = 1, \ldots, \left\lfloor \frac{L}{2} \right\rfloor$. Whenever an order $j \in J$ occurs at position l of assignment vector a for the kth time (for $k = 1, 2, \ldots, \eta_j$), these constraints make sure that after-buffer sequence π^{out} contains an item of SKU $s_{j,k}$ at position l. Constraints (9) force the foremost item $s_{j,1}$ of order j to the first position of π^{out}, whenever $a_{j,1} = 1$ holds. For the following sequence positions $l = 2, \ldots, \left\lfloor \frac{L}{2} \right\rfloor$, constraints (10) and (11) establish the link between a and π^{out} as follows. Due to term (i), they impose a restriction for order $j \in J$ and sequence position $l = 2, \ldots, \left\lfloor \frac{L}{2} \right\rfloor$ only if j is assigned to l. Term (ii) evaluates the number of occurrences of j in a up to position $l-1$ minus $k-1$.

If $a_{j,l} = 1$ and the left hand side of (10) has a value of at least 1, that is if j has at least k occurrences in a up to position l, then constraints (10) enforce the lth entry of π^{out} to be in $\{s_{j,k}, s_{j,k+1}, \ldots, s_{j,\eta_j}\}$. If the left hand side has a value of at most 0, that is if j has at most $k-1$ occurrences in a up to position l (or $a_{j,l} = 0$), then (10) does not restrict π^{out}.

If $a_{j,l} = 1$, the left hand side of (11) evaluates k minus the number of occurrences of j in a up to position $l-1$. If the left hand side has a value of at least 1, that is if j has at most k occurrences in a up to position l, then constraints (11) enforce the lth entry of π^{out} to be in $\{s_{j,1}, \ldots, s_{j,k}\}$. If the left hand side has a value of at most 0, (11) does not restrict π^{out}.

Note that the only case in which both, (10) and (11), restrict π^{out} is the case in which both left hand sides have a value of 1. In this case, constraints (10) and (11) enforce the lth entry of π^{out} to be in $\{s_{j,1}, \ldots, s_{j,k}\} \cap \{s_{j,k}, s_{j,k+1}, \ldots, s_{j,\eta_j}\}$ and, thus, equal to $s_{j,k}$.

$$a_{j,L} \leq \pi^{\text{out}}_{s_{j,\eta_j},L} \quad \forall\, j \in J, \tag{12}$$

$$1-(L-l-k+2)\cdot(1-a_{j,l}) + \left(\sum_{l'=l+1}^{L} a_{j,l'} - (k-1)\right) \leq \eta_j \cdot$$

$$\sum_{k'=1}^{\eta_j-k+1} \pi^{\text{out}}_{s_{j,k'},l} \quad \forall \begin{array}{c} j \in J; \\ l = \left\lfloor \frac{L}{2} \right\rfloor + 1, \ldots, L-1; \\ k = 1, \ldots, \min\{L-l, \eta_j\}, \end{array} \tag{13}$$

$$1-k\cdot(1-a_{j,l}) - \left(\sum_{l'=l+1}^{L} a_{j,l'} - (k-1)\right) \leq \eta_j \cdot$$

$$\sum_{k'=\eta_j-k+1}^{\eta_j} \pi^{\text{out}}_{s_{j,k'},l} \quad \forall \begin{array}{c} j \in J; \\ l = \left\lfloor \frac{L}{2} \right\rfloor + 1, \ldots, L-1; \\ k = 1, \ldots, \min\{L-l, \eta_j\}. \end{array} \tag{14}$$

Inequalities (12) to (14) link the entries of assignment vector a to after-buffer sequence π^{out} for sequence positions $l = \left\lfloor \frac{L}{2} \right\rfloor + 1, \ldots, L$ using the same ideas as in (9) to (11).

$$C_j \geq l \cdot a_{j,l} \qquad \forall\, j \in J; l = 1, \ldots, L, \tag{15}$$

$$a_{j,l} \in \{0,1\} \qquad \forall\, j \in J; l = 1, \ldots, L, \tag{16}$$

$$\sigma^l_{s,k} \in \{0,1\} \qquad \forall\, s \in S; k = 1, \ldots, |S|, \tag{17}$$

$$\pi^{\text{out}}_{s,l} \in \{0,1\} \qquad \forall\, s \in S; l = 1, \ldots, L. \tag{18}$$

Constraints (15) bound the completion times of the orders from below. Finally, constraints (16) to (18) set the domains of the binary variables. In total, we have $O(|S|^2 \cdot L + |J| \cdot L \cdot n)$ constraints.

3.2 | Valid inequalities

To strengthen our MIP, we introduce the following valid inequalities.

(i) Completion time C_j for each order $j \in J$ can be bounded from below by

$$C_j \geq \sum_{k=1}^{l(\sigma_j)} n_{\sigma_j(k)} - \max_{k=1,\ldots,l(\sigma_j)} \left\{ n_{\sigma_j(k)} - n_{j,\sigma_j(k)} \right\}.$$

For each order $j \in J$, all items of all but one required SKU need to be retrieved completely, and from the remaining SKUs at least the items that are required by j itself need to be retrieved before j can be completed. Note that the $l(\sigma_j) - 1$ SKUs that are retrieved completely, are not necessarily equal to the first $l(\sigma_j) - 1$ SKUs of order j.

(ii) The following inequalities enforce a to not assign an empty slot in π^{out} for any order.

$$\sum_{j \in J} a_{j,l} = \sum_{s \in S} \pi^{\text{out}}_{sl} \quad \forall\, l = 1, \ldots, L.$$

Although this constraint does not cut any solutions, it improved the solver's performance nevertheless.

(iii) We can exclude after-buffer sequences where an item of SKU s is scheduled directly after an empty slot (i.e., where $\pi^{\text{out}}_{s,l} = 1$ and $\sum_{s' \in S} \pi^{\text{out}}_{s',l-1} = 0$ hold for some $l \in \{3, \ldots, L\}$), unless SKU s occurs there for the first time (i.e., unless $\sum_{l'=1}^{l-2} \pi^{\text{out}}_{s,l'} = 0$):

$$n_s \cdot \left(1 + \sum_{s' \in S} \pi^{\text{out}}_{s',l-1} - \pi^{\text{out}}_{s,l}\right) \geq \sum_{l'=1}^{l-2} \pi^{\text{out}}_{s,l'}$$
$$\forall\, s \in S; l = 3, \ldots, L.$$

Starting from a solution violating this inequality, we could shift the item in slot l to an earlier slot without increasing the objective value. These constraints reduce symmetry by eliminating feasible solutions, where a slot in the after-buffer sequence is left empty although the next item in the after-buffer sequence is available.

(iv) If two SKUs $s, s' \in S$ jointly occur in at least one order and if s' neither occurs without s nor before s in any order, we cannot profit from retrieving SKU s' before SKU s unless s' requires fewer items than s (i.e., unless $n_{s'} < n_s$). Thus, we can add the following constraints, which prohibit retrieving s' before s in such cases:

$$\sum_{k=1}^{|S|} k \cdot (\sigma^{l}_{s',k} - \sigma^{l}_{s,k}) \geq 1$$
$$\forall\, s, s' \in S : n_s \leq n_{s'} \wedge \Big\{ j \in J : \exists k' \in \{1, \ldots, l(\sigma_j)\} : s' \in \bigcup_{k''=1}^{k'} \{\sigma_j(k'')\} \ni /s \Big\} = \varnothing.$$

Note that having s' preceding s in the loading sequence does not necessarily increase the objective value, for example, in instances where neither s nor s' is the last SKU in any order's packing sequence.

(v) Due to Lemma 2, we can add the following constraints, which prohibit empty slots followed by non-empty slots in the rear part of π^{out} (and therefore also in a):

$$(L - l) \cdot \sum_{j \in J} a_{j,l} \geq \sum_{j \in J} \sum_{l'=l+1}^{L} a_{j,l'}$$
$$\forall\, l = n - \min_{s \in S}\{n_s\} + 1, \ldots, L - 1. \tag{19}$$

These constraints cut non-optimum solutions where some orders have unnecessarily large completion times since late slots in the after-buffer sequence are occupied.

Note that we also tested some more valid inequalities, but this is the subset of rules that noticeably improves the performance of the standard solver Gurobi (see Section 6).

3.3 | Reducing the length of the after-buffer sequence

As our computational tests in Section 6 will show, solving our MIP with a default solver profits from a tight approximation of the maximum possible length L of the after-buffer sequence. According to our ideas presented in the preprocessing step (see Appendix A), L is yet slightly smaller than $2 \cdot n$, with n being the overall number of items to be processed. However, with a feasible (and near-optimal) solution to PSS on hand, for example, provided by our heuristic solution procedure described in Section 5, we are (potentially) able to further reduce L. Specifically, we aim to identify slots at the end of the after-buffer sequence that cannot be occupied by items in an optimal solution. Let *sol* be a feasible solution and Z_{sol} the related objective value. In the following, we describe how to employ solution *sol* for bounding the maximum possible length L of the after-buffer sequence in optimum solutions.

According to inequalities (19), the final slot L can only be occupied by an item, if each slot $l \in \{n - \min_{s \in S}\{n_s\} + 1, \ldots, L\}$ is occupied by an item. Thus, if the after-buffer sequence π^{out} of an arbitrary solution contains an item at position L, then π^{out} terminates with $L - n + \min_{s \in S}\{n_s\}$ occupied slots. The after-buffer sequence, then, contains only $2n - L - \min_{s \in S}\{n_s\}$ items in the slots $1, \ldots, n - \min_{s \in S}\{n_s\}$. With this information about the location of items within π^{out} on hand, we are able to derive a lower bound $LB(L)$ for the sum of the order completion times under the condition that $\pi^{\text{out}}(L)$ is not empty. Let $\omega : J \to \{1, \ldots, |J|\}$ be a permutation of order set J, such that for all $j, j' \in J$ we have $\omega(j') < \omega(j)$ whenever $\eta_{j'} < \eta_j$ (i.e., a nondecreasing ordering of order set J according to the number of items required by the orders). In the best case, the orders of J are completed according to the ordering ω (i.e., according to the shortest

processing time (SPT) rule), such that the majority of orders is finished in slots $l \in \{1, \dots, 2n - L - \min_{s\in S}\{n_s\}\}$ and only a few orders (i.e., the largest ones) are finished in slots $l \in \{n - \min_{s\in S}\{n_s\} + 1, \dots, L\}$. $LB(L)$ then amounts to

$$LB(L) = \underbrace{\sum_{j\in J} \sum_{\substack{j'\in J:\\ \omega(j')\le\omega(j)}} \eta_j}_{\substack{\text{lower bound for the sum of completion times}\\ \text{assuming that all orders can be finished with}\\ \text{the first } n \text{ slots of the after-buffer sequence}}} + \underbrace{(L-n)\cdot x_L}_{\substack{\text{correction term:}\\ \text{at least } x_L \text{ orders are}\\ \text{each delayed by } L-n \text{ slots}}} \tag{20}$$

with

$$x_L = |J| - \min\left\{k \in \{1, \dots, |J|\} : \sum_{\substack{j\in J:\\ \omega(j)\le k}} \eta_j > 2n - L - \min_{s\in S}\{n_s\}\right\} + 1, \tag{21}$$

being the number of many-item orders that are completed in slots $n - \min_{s\in S}\{n_s\} + 1, \dots, L$. If $LB(L)$ exceeds Z_{sol}, we can conclude that in an optimal solution to PSS the Lth slot of the after-buffer sequence cannot be occupied by an item, so that we are able to reduce L by one. The computation of $LB(L)$ (with a reduced L) and the comparison with Z_{sol} can then be restarted until L cannot be further reduced. Obviously, reducing L leads to lower numbers of variables and constraints in our MIP formulation.

4 | ANALYSIS OF THE PROBLEM STRUCTURE

In this section, we investigate the problem structure of PSS and provide an in-depth analysis of computational complexity. First, we show that PSS is strongly $\mathcal{NP}$-hard. Then, we investigate what happens with the complexity status of the problem, if we fix parts of the solution. Here, we aim to identify levers for decomposition approaches, for example, based on an efficient neighborhood structure, so that some metaheuristic can evaluate the remaining subproblem for a given partial solution in polynomial time. We first report on some subproblems remaining $\mathcal{NP}$-hard, before we come to promising subproblems solvable in polynomial time.

We start with an expected result: not only PSS itself but also variants where parts of the solution are predetermined are complex optimization problems.

Theorem 1. *The following variants of PSS are strongly $\mathcal{NP}$-hard:*

1. *PSS,*
2. *PSS with given after-buffer sequence π^{out}, and*
3. *PSS with given loading sequence σ^l and after-buffer sequence π^{out}.*

Proof. See Appendix B. ∎

Now, we turn our attention to subproblems of PSS that can be solved in polynomial time. First, however, we present an auxiliary result (Lemma 3) which will help us prove Theorem 2 and will support the development of our metaheuristic approach in Section 5. We refer to the subsequence π^{sig} of after-buffer sequence π^{out} starting and ending with the first and the last item in π^{out} as the signature of π^{out}.

Lemma 3. *For a given signature π^{sig}, we can determine in polynomial time the after-buffer sequence π^{out} of minimum length such that π^{sig} is the signature of π^{out} and a loading sequence σ^l ensuring buffer-feasibility with π^{out} exists.*

Proof. Let l^{sig} be the length of signature π^{sig}. We construct π^{out} and π^{in} (implying σ^l) as follows. We consider an after-buffer sequence π^{out} with n leading empty slots before signature π^{sig}. We construct a before-buffer sequence π^{in} of the same length $n + l^{\text{sig}}$ with a maximum number of leading empty slots such that the first item of each SKU appears in π^{in} not later than in π^{out}. Note that this ensures buffer-feasibility, see Lemma 1. Finally, we drop leading slots that are empty in both, π^{in} and π^{out}, and empty slots after the last item in π^{in}.

It remains to detail how π^{in} is constructed. For each SKU $s \in S$ the rearmost feasible position $\overline{\pi}^{\text{in}}(s)$ in π^{in} for the last item of s is determined as $f_s^{\text{out}} + n_s - 1$ where f_s^{out} is the first slot in π^{out} where an item of SKU s occurs. Now, we start with an empty loading sequence σ^l and repeatedly choose among SKUs not in σ^l yet the SKU s with largest value of $f_s^{\text{out}} + n_s - 1$ (using an arbitrary tie breaker). We assign the items of s to the latest empty slots in π^{in} but not to slots larger $f_s^{\text{out}} + n_s - 1$. After dropping leading slots that are empty in both, π^{in} and π^{out}, but before removing empty slots from the end of π^{in} we can measure the length of π^{out} as the number of empty slots in π^{in} plus n. Note that the number of empty slots is minimum since we minimize the number of empty slots after the items of the kth SKU, $k = 1, \dots, |S|$, in σ^l by choosing in each step the SKU with maximum $f_s^{\text{out}} + n_s - 1$. ∎

Theorem 2. *For a given loading sequence σ^l or a given after-buffer sequence π^{out} PSS can be solved in polynomial time, if the number of orders $|J|$ is bounded by a constant.*

Proof. See Appendix C. ∎

Unfortunately, the result in Theorem 2 relies on a fixed number $|J|$ of orders. In real-life applications, however, this number can be large. The pocket sorter system reported by Boysen, de Koster, and Weidinger (2019), for instance, applies 80 packing stations each processing an order in parallel. Thus, using decomposition approaches utilizing this theorem seems only reasonable for small-sized systems. In the following, however, we present a polynomially solvable subproblem of PSS for arbitrary $|J|$.

Theorem 3. *PSS can be solved in polynomial time, if the assignment a is given.*

Proof. See Appendix D. ∎

The following can be concluded from our analysis of computational complexity: PSS is a very complex optimization. However, a decomposition approach (e.g., a metaheuristic framework) iterating through different assignments a could be a promising solution approach, because once an assignment a is given the remaining subproblem of PSS can efficiently be evaluated in polynomial time. For a proof of concept, we will show that a metaheuristic such as simulated annealing (SA) can successfully apply this approach.

5 | A METAHEURISTIC APPROACH FOR SOLVING PSS

In this section, we utilize the insights gained from our problem analysis to develop a suitable heuristic solution approach. We demonstrate that a straightforward metaheuristic like SA can effectively solve the PSS problem by operating on an appropriate neighborhood structure. SA is a simple stochastic local search metaheuristic that accepts modified neighboring solutions based on a probabilistic scheme inspired by thermal processes used to achieve low-energy states in heat baths (see Kirkpatrick et al., 1983; Van Laarhoven & Aarts, 1987). In the following, we outline the key components of our SA approach, including the solution encoding, the neighborhood structure, the generation of initial solutions, and the general procedure.

Solution encoding: Theorem 3 provides us with crucial information for efficiently deriving a complete solution from a concise (partial) solution encoding. Based on this theorem, we determine that the metaheuristic search process should operate on the assignment of slots in the after-buffer sequence to the orders. We define this assignment by a vector a with

$$a(k) = \begin{cases} j & \text{if the after-buffer sequence} \\ & \text{contains an item for order } j \in J \text{ in slot } k \\ - & \text{if the } k\text{th slot of the after-buffer sequence is empty.} \end{cases}$$

In a feasible assignment vector a, encoding a feasible solution to PSS, $|\{k \in \{1, \dots, L\} : a(k) = j\}| = \eta_j$ necessarily holds for each order $j \in J$. A feasible assignment vector, thus, contains exactly $L - n$ empty slots (i.e., $|\{k \in \{1, \dots, L\} : a(k) = -\}| = L - n$). The sequence π^{out} of items (and empty slots) in the after-buffer sequence then trivially follows from a. With π^{out} on hand, a buffer-feasible loading sequence σ^l and before-buffer sequence π^{in} can be determined in polynomial time (if they exist) by applying the approach used in the proof of Lemma 3. If buffer-feasible loading sequence σ^l and before-buffer sequence π^{in} do not exist for (assignment vector a and) after-buffer sequence π^{out}, then the approach used in the proof of Lemma 3 adds leading empty slots into a and π^{out} in order to ensure buffer-feasibility. Note that σ^l is not unique, if $\overline{\pi}^{\text{in}}(s_1) = \overline{\pi}^{\text{in}}(s_2)$ (as defined in Lemma 3) holds for at least one pair of SKUs s_1, s_2 with $s_1 \neq s_2$. To not systematically exclude parts of the solution space, we thus apply a random SKU (sub)sequence among those SKUs with identical values for $\overline{\pi}^{\text{in}}$. The leading empty slots in a and π^{out} are then postponed as far as possible without violating buffer-feasibility.

Note that an assignment vector a, thus, potentially carries a lot of empty slots and still requires a repair scheme for finding a feasible solution. If we allow, however, for a repair scheme, then it seems promising to omit empty slots, because this allows for a much more compact encoding. Thus, we restrict the solution encoding of our SA approach on *no-empty-slots assignment vectors* $\widetilde{a} : \{1, \dots, n\} \to J$, that is, on assignment vectors without any empty slots. Vector $\widetilde{a}$, thus, has length n and describes the succession in which the orders of J are supplied with an item of the after-buffer sequence. To derive after-buffer sequence π^{out}, we feed $\widetilde{a}$ into the repair scheme described above. This rather complex formal description of the solution encoding in our SA can easily be explained with the help of the following example.

Example: We have two orders with given packing sequences $\sigma_1 = (A, B)$ and $\sigma_2 = (A, C)$ with $n_{1,A} = n_{1,B} = n_{2,A} = n_{2,B} = 1$. An iteration of our SA obtains a no-empty-slots assignment vector $\widetilde{a} = (1, 1, 2, 2)$ defining the orders to which items arriving in the after-buffer sequence are dedicated. To ensure buffer-feasibility, the approach of Lemma 3 integrates one leading empty slot, so that we obtain $a = (0, 1, 1, 2, 2)$ and $\pi^{\text{out}} = (0, A, B, A, C)$. Simultaneously, σ^l and π^{in} are set to (A, B, C) and (A, A, B, C), respectively. The empty slot in a and π^{out} is then postponed by one unit which changes a and π^{out} to $a = (1, -, 1, 2, 2)$ and $\pi^{\text{out}} = (A, -, B, A, C)$. This translates into completion times 3 and 5

for orders 1 and 2, respectively, and a sum of order completion times of 8.

Neighborhood structure: To modify a current solution of our SA into a neighboring solution, we apply swap moves. That is, we randomly determine two distinct sequence positions within no-empty-slots assignment vector $\widetilde{a}$ and swap the orders specified at the selected positions. In this way, we obtain neighboring solution $\widetilde{a}'$. Whether we accept $\widetilde{a}'$ and make it to the starting point of our further search is decided by the standard probabilistic acceptance scheme of SA, see Van Laarhoven and Aarts (1987):

$$\text{Prob}\left(\widetilde{a}' \text{ is accepted}\right) = \begin{cases} 1, & \text{if } Z(\widetilde{a}') \leq Z(\widetilde{a}) \\ exp\left(\frac{Z(\widetilde{a})-Z(\widetilde{a}')}{T^s}\right), & \text{otherwise,} \end{cases} \quad (22)$$

where $Z(\widetilde{a})$ and T^s refer to the sum of completion times resulting from a no-empty-slot assignment vector $\widetilde{a}$ and a steering parameter called temperature (see below), respectively.

Initial solutions: To efficiently utilize our computer environment, we implemented a multi-threaded SA that runs in parallel on all available cores. Since the related single machine scheduling problem $[1||\sum C_j]$ (see Graham et al., 1979) is well known to be solvable to optimality in polynomial time with the shortest-processing time rule (e.g., see Smith, 1956), we initialize the SA process on one core with a solution derived from this rule. Specifically, we sequence all orders according to increasing numbers of demanded SKUs. Then, we start with the first SKU within the packing sequence of the first order. We fix the first positions within no-empty-slot assignment vector $\widetilde{a}$ according to this order's item demand for the active SKU. Afterwards, we switch to the next order in our order sequence and fix all following item demands within $\widetilde{a}$ to the current order's demand for the active SKU (if any). In this way, we iterate through the order sequence. Once this is done, we switch to the next SKU of the packing sequence of the first order and repeat the previous procedure. Once all SKUs of the first order are satisfied, we switch to the next order of our order sequence and continue the process with all SKUs not already addressed. The procedure stops once all order demands are fulfilled. All other cores are initialized with no-empty-slot assignment vectors $\widetilde{a}$ gained from random order sequences. In this case, the only adaption of the previous procedure is that we initiate the process with a random order sequence.

General procedure: Starting with an initial solution, we generate neighboring solutions iteratively and determine whether to accept a solution based on the probability defined in (22). To control our SA, we apply the straightforward static cooling scheme proposed by Kirkpatrick et al. (1983). As values for the control parameters initial temperature T^s, stop temperature T_e, and decrease rate d_r, we apply $\min\{0.4 \cdot Z(\widetilde{a}^*); 100\}$, 1, and 0.999, respectively. Here, $Z(\widetilde{a}^*)$ refers to the objective value of initial assignment vector $\widetilde{a}^*$. Note that preliminary tests, not included in this article, have shown that this parameter setting delivers reasonably good results. After each swap move, the temperature is updated by multiplying it with the decrease rate: $T^s := T^s \cdot d_r$. As a result, the acceptance of worse solutions becomes less likely as more storage assignment vectors are evaluated. The procedure terminates when temperature T^s reaches stop value T_e. We implemented our SA in a multi-threaded manner, where each available processor core initializes an independent SA. Finally, the best solution found during the search process on all cores is returned.

6 | COMPUTATIONAL STUDY

This section focuses on our computational study, where all procedures have been implemented in Visual Basic based on Microsoft's .NET Framework 4.7.1. The computations were performed on a personal computer equipped with an Intel Core i7-3770 processor running at a clock speed of 4 × 3.4 GHz, and with 8 GB DDR-3 RAM. We utilized Gurobi Optimizer 9.0 as our standard solver.

Initially, we provide details about our data instances, which are discussed in Section 6.1. Subsequently, we assess the computational performance of our solution approaches. Specifically, we compare the performance of a default solver solving our MIP with the SA heuristic presented in Section 6.2. Lastly, in Section 6.3, we address the managerial issues outlined in Section 1.3.

6.1 | Data instances

For our computational study, we apply both systematically generated random data and a case study obtained from the warehouse that brought the need to schedule a pocket sorting process to our attention. The artificial data set is applied to systematically explore the performance of our solution approaches and the case study to investigate the managerial issues.

We begin with the generation procedure for our artificial data. Our instance generator takes the parameter values listed in Table 2 as its input data. We differentiate between *small* instances that can still be solved by our default solver for the MIP, and *large* instances that can only be solved using our heuristic approaches. The values for the number of orders ($|J|$), the number of SKUs ($|S|$), and the interval ($[\underline{\gamma}; \overline{\gamma}]$) from which the number of demanded SKUs per order are randomly drawn are combined in a full factorial manner for both cases. This results in four unique parameter settings for small instances and 27 unique parameter settings for large instances. Instance generation has been repeated 25 times for each setting, yielding a total of 100 small instances and 675 large instances. Note that the number of orders ($|J|$) specifies the sizes of the wave being processed concurrently and does not indicate the total number of orders processed in the

TABLE 2 Parameter values for instance generation.

Parameter	Description	Values	
		Small	Large
$\|J\|$	Number of orders	4, 5	10, 30, 50
$\|S\|$	Number of SKUs	4, 5	10, 30, 50
$[\underline{\gamma};\overline{\gamma}]$	Interval of SKUs per order	[1; 4]	[1; 3], [7; 10], [1; 10]
$n_{j,s}$	Number of items per SKU	[2; 4]	[2; 4]

warehouse. The wave size is typically determined by the number of parallel packing stations. Therefore, systems with up to $|J| = 50$ orders per wave represent relatively large warehouses with 50 parallel stations. Similarly, the number of SKUs ($|S|$) does not refer to the total number of SKUs stored in a warehouse, but rather the total number of SKUs demanded by the current wave of orders. The total number of SKUs stored in the warehouse can be much larger.

Specific instances are obtained as follows: first, for each order $j \in \{1, 2, \ldots, |J|\}$, a number of demanded SKUs is drawn from interval $[\underline{\gamma};\overline{\gamma}]$. According to this random number, each order obtains its respective randomly chosen (but pairwise different) SKUs from $\{1, 2, \ldots, |S|\}$. For each SKU a random number $n_{j,s}$ is drawn from [2; 4] to determine the number of items per SKU. A random permutation of the SKUs in j determines the packing sequence σ_j.

Furthermore, we apply a second data set obtained from the logistics provider that operates a pocket sorter system for our German retailer specializing in hunting equipment. The retailer operates 25 brick-and-mortar stores located throughout Germany and offers online shopping through internet and print catalogs. The central warehouse stocks approximately 45 000 SKUs, with additional SKUs drop-shipped from suppliers. Most of these SKUs are exclusively available to online customers, while only about 6000 SKUs are offered in the relatively smaller brick-and-mortar stores. Unfortunately, we were not granted permission to conduct our tests using actual order data. Instead, the logistics provider's managers provided us with the necessary aggregate information, and we randomly generated store order data based on actual orders.

The order sizes of the stores roughly follow a truncated normal distribution with a mean (μ) of 150 and a standard deviation (σ) of 100, from which we draw the number of SKUs per order. The SKU activity profile is skewed, with the top 10% of SKUs accounting for 85% of the demand. The average number of items demanded per SKU is 2.2, so we draw the number of items per SKU from a truncated normal distribution with a mean of 2 and a standard deviation of 2, rounding to the nearest integer. Since no information on the store layout was available, we randomly generated the packing sequences unless otherwise specified. The warehouse's pocket sorter system is connected to nine packing stations, so we set the wave size ($|J|$) to 9. The logistics service provider's managers have agreed upon an average loading time of 12 s per item into a bag, in consultation with the local union representatives. The workers at the packing stations are not exclusively assigned to fulfill store orders. When a packing station completes its current store order but other stations are still processing the current wave, idle workers are redirected to various other duties within the multi-client warehouse. Therefore, once a store order is completed, the remaining time is effectively utilized for other tasks, aligning with the objective of minimizing the total completion time. We repeat the instance generation process ten times and report the average solution values when referring to the results of our case study.

6.2 | Computational performance

In an initial test, we assess the computational performance of the default solver Gurobi when solving our MIP formulated in Section 3. Unfortunately, the default solver can only handle the small instances of our artificial dataset, as defined in Section 6.1. For the large instances, Gurobi failed to return a feasible solution within a runtime of one hour. Even when we applied our best model version, PSS-MIP-EXT-WS-RL (described in Section 3 and Appendix E), where we warm-start Gurobi with the feasible SPT solution, no improvement was achieved before reaching the timeout after one hour. Note, however, that our small instances result in a total of up to 45 demanded items, so that these instances are already pretty challenging. We also implemented a straightforward MIP, that does not exploit any structural properties, and in these tests our instances were already beyond reach. Since PSS is an operational problem requiring fast solutions, we limited the runtime of Gurobi to 300 CPU seconds. Note, however, that with a runtime of 1 h all small instances are solved to proven optimality.

To evaluate whether our extensions to the basic MIP, as defined in Section 3.1, contribute to streamlining the solution process, we tested various configurations of our MIP using the off-the-shelf solver Gurobi. These configurations included the preprocessing described in Appendix A, the valid inequalities from Section 3.2, the warm start of Gurobi with the heuristic solution obtained by our SA (described in Section 5), and the reduction of the length L of the after-buffer sequence for the solution obtained by SA (as defined in Section 3.3). The detailed results of this test are reported in Appendix E. It can be concluded from these tests that each of our extensions contributes to improving Gurobi's solution process. While the basic MIP without any extensions could only solve 56% of all small instances to proven optimality before reaching the timeout, the MIP with all extensions improved this fraction to 86% and reduced the solution time by almost half.

Therefore, solving PSS-MIP with all the extensions introduced in Section 3 yields the best performance results. This configuration is also used to benchmark our SA approach from Section 5 on the same instances. The detailed results

of this test are also reported in Appendix E. These results demonstrate that SA finds optimal solutions for 59% of the tested instances and achieves an average gap of just 1.26% compared to the solutions obtained by the best MIP version, with an average solution time of 5.84 seconds. Based on these findings, we conclude that our SA approach produces competitive results for the small instances.

Due to the computational limitations of the standard solver Gurobi, we cannot solve our large test instances (as described in Section 6.1). Therefore, we benchmark our SA approach against the following two simple solution approaches for the large instances:

1 RWP: The real-world policy RWP represents the observed scheduling procedure at the warehouse that brought the PSS problem to our attention. Unaware of the significant influence that different SKU processing sequences and buffer operations can have on throughput performance, they applied the following decision rules at each decision point when considering the subsequent slots of the item sequence:

   (a) At the loading station: If the current slot of the before buffer sequence is empty, load all items of a random SKU that is still demanded by the current wave of store orders. Thus, we have a random loading sequence that implies the before-buffer sequence.
   (b) At the buffer entrance: If the current item is required next by at least one store order, assign it to a random one among them and let it pass the buffer, otherwise move it into the buffer.
   (c) At the buffer exit: If the current slot is empty, remove a random item from the buffer among those required next by at least one store order and assign it randomly to an appropriate order. If no such item is in the buffer, the slot remains empty.

2 SPT: Since the SPT rule is well known to solve related single machine scheduling problem $[1||\sum C_j]$ to optimality in polynomial time (Smith, 1956), we evaluate an adaption of the SPT rule for our PSS. Note that we apply a similar procedure to initialize our SA procedure (see Section 5). However, this version of SPT is more application oriented and comes by with a simple set of decision rules at each decision point:

   (a) At the loading station: If no order is referred to as the *current* order (i.e., at the start of the procedure) or if all SKUs of the current order have already been sent, select the (not already completed) order with the fewest open SKU demands as the current order. Load all demanded items (i.e., of the complete wave) of the SKU at the next open position of the current order's packing sequence.
   (b) At the buffer entrance: If the current item is demanded at the next open position of the packing sequence of one or multiple store orders, assign it to the one among them with the fewest open SKUs and let it pass the buffer, otherwise move the item into buffer.
   (c) At the buffer exit: If the current slot is empty, add an item that is the first open item of an order. If there are multiple orders for which the first open item can be covered by an item in buffer, choose the order with the fewest open SKUs. If no such item is available, the slot remains empty.

Note that ties among orders are broken arbitrarily. The motivation for SPT lies in its optimality for $[1||\sum C_j]$. This machine scheduling problem is to minimize the sum of completion times of a set of jobs that need to be processed consecutively on a single machine. The analogy to PPS can be seen when inspecting how the after-buffer sequence delays the completion of orders. When we imagine an idealized after-buffer sequence with all items of each order assigned to consecutive slots and no empty slots, then the after-buffer sequence in PPS corresponds to the time horizon in $[1||\sum C_j]$ and each order with η_j items in PPS corresponds to a job with processing time η_j in $[1||\sum C_j]$. The sum of completion times of orders, then, reflects the sum of completion times of jobs (plus a constant). Hence, although we will usually not be able to reach such an idealized after-buffer sequence, the intuition is to have orders with small numbers of items packed first, since their items

occupy only few slots in the after-buffer sequence and, therefore, delay completion of other orders only to a small extent.

Applying the performance criteria listed in Table 3, the results of the benchmark test among RWP, SPT, and SA on the large instances of our artificial data set are reported in Table 4. Note that the lower bound to derive performance measure "∅gLB" is derived by Gurobi solving the special lower bound MIP introduced in Appendix F. These results suggest the following findings:

- SA: Our metaheuristic approach achieves the best solutions for each individual instance. For the largest instances with $|J| = 50$ orders, the average runtime of SA increases to 570.2 s. Note, however, that these instances have sequences with up to 2.000 items. Considering a typical average slot duration between 10 and 20 s (as mentioned in Section 2.1), work schedules spanning several hours are thus obtained. Additionally, it is difficult to derive strong lower bounds on optimal PSS objective values. Since the large instances in our artificial dataset are too large to be solved using PSS-MIP-EXT-WS-RL, the performance measure "∅gLB" reports the average gap of the SA solution compared to a theoretical lower bound obtained by solving an auxiliary MIP that relaxes the buffer operations and the assignment of items to orders (see Appendix F for more details). Unfortunately, even the remaining optimization task of SKU sequencing is strongly $\mathcal{NP}$-hard. As a result, Gurobi struggles with solutions for instances with $|J| \geq 30$, and the obtained bounds are not tight enough to make a final judgment on the solution quality of our metaheuristic.
- SPT: The straightforward priority-rule-based approach SPT surprisingly yields excellent results. The solution time never exceeds 0.1 CPU-seconds. The average gap (documented in the column "∅gB") from the best solutions obtained by SA is only 2.18% across all instances. Moreover, in 325 instances (48%), SPT is able to achieve the same objective values as SA.
- RWP: Neglecting the optimization task of PSS, as done in our example warehouse, comes at a significant cost. Random solutions obtained by real-world policy RWP result in gaps well above 50%, compared to both SPT and SA.

These results lead to the following important conclusions of this article: PSS is a complex and challenging optimization

TABLE 3 Performance criteria.

Criteria	Description
#b	Number of instances where the solution method found the best solution among its competitors
∅gLB	Average gap to the best lower bound obtained by Gurobi in % for the MIP of Appendix F
∅gB	Average gap to best solution among all approaches in %
∅impRWP	Improvement over real-world policy RWP in %
∅sec	Average computational time in CPU seconds

TABLE 4 Computational performance of simple priority rule-based approach SPT compared to our SA heuristic and real-world policy RWP.

		SPT			SA		
$\|J\|$	$\|S\|$	∅gB	∅impRWP	#b	∅gLB	∅impRWP	∅sec
10	10	3.3	47.92	32	20.42	52.71	5.84
10	30	1.67	59.77	46	56.12	62.14	25.45
10	50	1.22	62.86	49	59.66	64.62	66.54
Total		2.06	56.85	127	45.4	59.82	32.61
30	10	3.92	48.57	27	28.38	54.45	111.79
30	30	1.49	54.01	42	84.05	56.17	207.57
30	50	1.08	58.51	40	92,79	60.08	344.51
Total		2.16	53.7	109	68,41	56.9	221.29
50	10	4.33	47.62	10	30.27	53.96	444.8
50	30	1.9	50.88	29	89.70	53.63	529.15
50	50	0.73	55.02	50	96.54	56.06	736.65
Total		2.32	51.17	89	72.17	54.55	570.2

problem. However, our computational results demonstrate that a straightforward heuristic based on the SPT rule yields remarkably small gaps. Therefore, for real-world warehouses, applying SPT should be sufficient. However, completely neglecting the optimization task is not advisable. Random solutions, such as those obtained by real-world policy RWP, lead to substantial gaps of 50% or more. Gaps in this range result in a significant loss of performance, which will be further investigated in the following section.

6.3 | Managerial issues

In this section, we examine our case study and the dataset obtained from a third-party warehouse responsible for fulfilling store orders for hunting equipment. By solving this dataset using the real-world policy RWP and the simple rule-based approach SPT, we demonstrate that the significant gaps of 50% or more reported in the previous section result in substantial performance improvements. Workers stationed at packing stations, who have already completed their store orders for the current wave, do not need to wait for the final order's completion. The next wave for additional stores can only commence once the last station has finished its order; the loading station and the main conveyor line, which passes through the buffer, are still occupied processing items for the stations that have not yet finished. However, workers from completed stations can be reassigned to other tasks in the meantime. In our case study's multi-client warehouse, for example, workers are transferred to other areas of the warehouse and receive notifications via pagers when the next wave is about to commence. The benchmark results for RWP and SPT when solving the data instances from our case study are presented in Table 5. To calculate the "average time saving per worker in h," we determine the difference between each worker's completion time and the maximum completion time, and then average these values across all workers. On the other hand, the performance measure "total time saving as a percentage of wave time" sums up the time savings of all workers and divides it by the maximum completion time, which corresponds to the wave time. These results yield the following findings:

We observe that an optimized pocket sorter schedule, even when employing a simple approach like SPT, leads to a significant improvement in workforce capacity utilization.

TABLE 5 Computational performance of priority rule-based approach SPT and real-world policy RWP.

	RWP	SPT
Gap to best solution	65.58%	0%
Average completion time in h	16.03	9.72
Minimum completion time in h	15.89	2.92
Maximum completion time in h	16.11	16.32
Average time saving per worker in h	0.08	6.6
Total time saving in % of wave time	0.51	62.4

Although both policies exhibit similar maximum completion times until the wave is completed after approximately 16 h, there is a substantial variation in average and minimum completion times. Under the RWP policy, workers remain occupied throughout the entire processing time of the wave, while the first worker is released after only 2.92 h under the SPT policy. Additionally, the average completion time is reduced to 9.72 h under SPT, allowing idle workers to be assigned to other tasks. In our case study, a total of 62.4% of the total working hours of the packing workforce for wave processing is saved by applying SPT, compared to only 0.51% when using the real-world policy RWP.

In another experiment, we investigate the impact of store layout standardization on the distribution center's effort in assembling the SFS. It is expected that similar store layouts will reduce the resequencing effort through the buffer, leading to a decrease in the sum of completion times and an improvement in fulfillment performance. In the most extreme scenario, where all stores have identical layouts and demand products in the same packing sequence, no intermediate buffer is required to alter the item sequence. To explore the effect of different levels of store layout standardization and the resulting variation in packing sequences among stores, we set up the following experiment.

We utilize the data instances from our case study and initially assume that all stores within a wave have identical layouts. In terms of packing sequences, this implies that if product A precedes product B in one store's packing sequence, it will also precede it in all other packing sequences. After solving the resulting data instance using our SPT approach, we introduce increasing levels of divergence among store layouts by randomly swapping pairs of items in the packing sequences of randomly selected stores. As more swaps are performed, the variation among store layouts and their packing sequences increases. By solving the case study instances using SPT for different numbers of swaps, we can evaluate the performance at different levels of store layout standardization when using a pocket sorter to process the same set of orders. Figure 4 (left) illustrates the percentage increase in the maximum completion time compared to the default case where all store layouts are identical. The maximum completion time is reached when the last station is released, indicating the completion of the current wave of store orders and the readiness of the pocket sorter system for the next wave. Figure 4 (right) displays the sum of completion times across all packing stations, representing the total working hours the packing workforce is engaged when processing the current wave. These results lead to the following findings:

- *Identical versus completely different layouts*: Standardizing store layouts offers significant potential for improving both the maximum and sum of completion times. The former, illustrated on the left of Figure 4, allows for faster completion of waves, enabling quicker

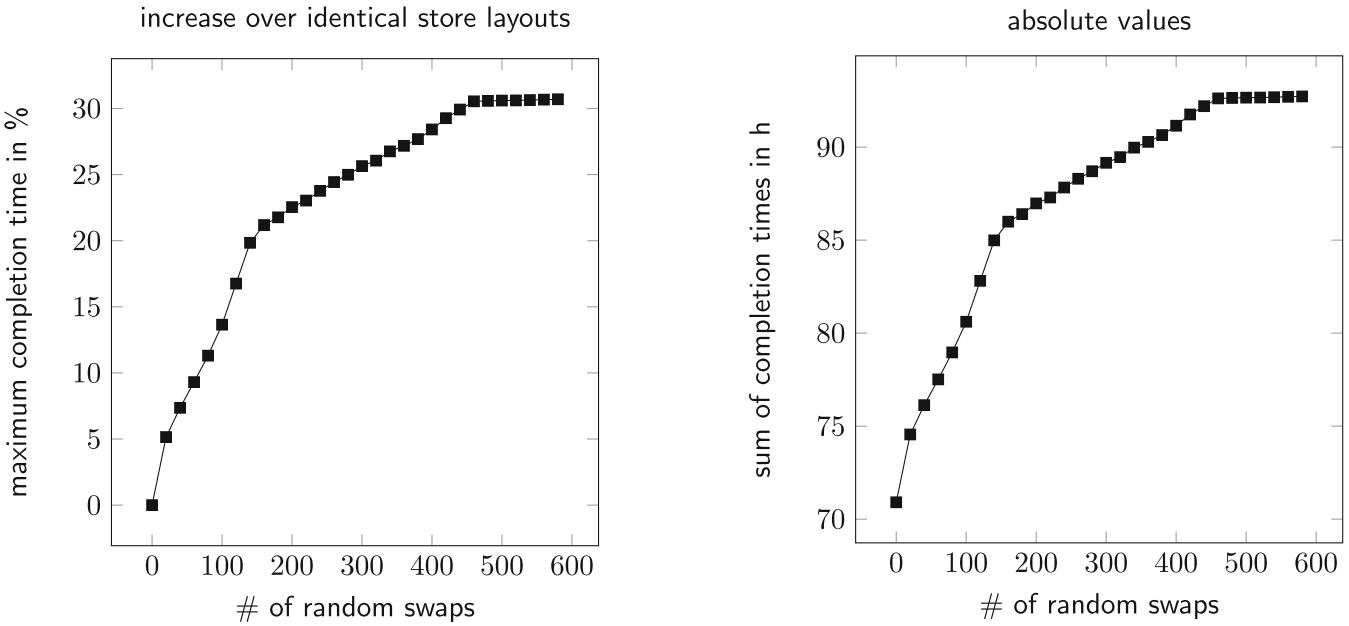

FIGURE 4 Consolidation performance for real-world data set depending on a higher (i.e., after fewer swaps) or lower (i.e., after more swaps) level of store layout standardization.

release of subsequent waves. This improves throughput and optimizes the capacity utilization of the pocket sorter system. When workers at packing stations finish their current orders but still have to wait for other stations, they can be assigned to other tasks, leading to better utilization of the workforce. The reductions in the sum of completion times, depicted on the right of Figure 4, indicate substantial potential for improving workforce utilization. The potential gains from completely identical store layouts (i.e., with 0 swaps) compared to completely different layouts (i.e., 400 swaps or more) exceed 30%.

- *Impact of some standardization*: Unfortunately, our results also indicate that these gains diminish as differences in store layouts increase, resulting in greater variation among the packing sequences of stores. In most retail chains, different store sizes, constructional and organizational aspects, and the evolution of store layouts over time make some degree of variation unavoidable. Our results indicate that, given some level of inevitable difference to begin with, serious effort is required to achieve positive effects through standardizing store layouts.

In conclusion, our managerial findings can be summarized as follows: both optimizing pocket sorter processes, even with simple decision rules, and standardizing store layouts show significant potential for streamlining the assembly of SFS with a pocket sorter.

7 | CONCLUSION

This article is the first to address the operational decision problems that arise when utilizing a pocket sorter for assembling store orders in a retail chain. With the implementation of a pocket sorter, individual products requested by store orders are loaded into bags, transported on a trolley conveyor, resequenced by exchanging bags using an intermediate buffer, and delivered to packing stations. At these stations, the products are retrieved from the bags and packed onto load carriers, such as roll cages, following the specific packing sequence that corresponds to the store layout. This enables more efficient shelf replenishment processes, eliminating the need for zigzagging through the stores. We formulate the operational decision problem to optimize this process, focusing on determining the loading sequence of SKUs into bags, the buffer operations, and the assignment of items to store orders at each packing station. We define the resulting optimization problem, analyze its computational complexity, and propose appropriate exact and heuristic solution approaches. In our computational study, we utilize these algorithms to obtain the main findings of our article, which are as follows:

- First, we demonstrate that optimization can significantly enhance the efficiency of the order fulfillment process with a pocket sorter. By solving our dataset inspired by a real-world warehouse, we find that even a simple heuristic based on the SPT rule can reduce workforce utilization by over 60% compared to the solutions applied in our case study.
- Although our operational optimization problem involves multiple interdependent

decisions and is known to be challenging to solve (especially in terms of computational complexity), a decision rule-based heuristic using the SPT rule produces remarkably good results that are not far behind those achieved by sophisticated optimization techniques.
- Lastly, standardizing store layouts to ensure consistent packing sequences across all sales outlets of a retail chain has the potential to streamline order fulfillment. However, our results indicate that achieving these benefits requires more than just partial standardization. Only when all store layouts are highly similar can we achieve gains of up to 30%.

Pocket sorters offer more than just a promising warehousing technology for SFS. They also enable a flexible and fully automated retrieval process once the products are loaded into bags. When combined with automated packing devices, pocket sorters can become a crucial component of future fully-automated fulfillment factories. Currently, pocket sorters are used not only for fulfilling store orders in a bulk picking environment but also for handling small-sized e-commerce orders with tight time constraints (Boysen, de Koster, & Weidinger, 2019). Therefore, pocket sorters and their impact on order fulfillment deserve further scientific attention in the future. There are several fruitful research tasks that should be addressed, such as exploring alternative objectives (e.g., minimizing the maximum completion time to accelerate wave succession) and investigating more complex setups involving multiple parallel loading stations. Additionally, analyzing the inner-buffer sortation processes would be valuable.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ACKNOWLEDGMENT

Open Access funding enabled and organized by Projekt DEAL.

ORCID

Nils Boysen https://orcid.org/0000-0002-1681-4856

REFERENCES

Azadeh, K., De Koster, R., & Roy, D. (2019). Robotized and automated warehouse systems: Review and recent developments. *Transportation Science*, *53*(4), 917–945.

Boysen, N., Briskorn, D., Fedtke, S., & Schmickerath, M. (2019). Automated sortation conveyors: A survey from an operational research perspective. *European Journal of Operational Research*, *276*(3), 796–815.

Boysen, N., de Koster, R., & Füßler, D. (2021). The forgotten sons: Warehousing systems for brick-and-mortar retail chains. *European Journal of Operational Research*, *288*(2), 361–381.

Boysen, N., de Koster, R., & Weidinger, F. (2019). Warehousing in the e-commerce era: A survey. *European Journal of Operational Research*, *277*(2), 396–411.

Boysen, N., Fliedner, M., Jaehn, F., & Pesch, E. (2012). Shunting yard operations: Theoretical aspects and applications. *European Journal of Operational Research*, *220*(1), 1–14.

Boysen, N., Golle, U., & Rothlauf, F. (2011). The car resequencing problem with pull-off tables. *Business Research*, *4*(2), 276–292.

Boysen, N., Scholl, A., & Wopperer, N. (2012). Resequencing of mixed-model assembly lines: Survey and research agenda. *European Journal of Operational Research*, *216*(3), 594–604.

Boysen, N., & Stephan, K. (2016). A survey on single crane scheduling in automated storage/retrieval systems. *European Journal of Operational Research*, *254*(3), 691–704.

Daganzo, C. F., Dowling, R. G., & Hall, R. W. (1983). Railroad classification yard throughput: The case of multistage triangular sorting. *Transportation Research Part A: General*, *17*(2), 95–106.

De Koster, R., Le-Duc, T., & Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, *182*(2), 481–501.

Gallien, J., & Weber, T. (2010). To wave or not to wave? Order release policies for warehouses with an automated sorter. *Manufacturing & Service Operations Management*, *12*(4), 642–662.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, *5*, 287–326.

Gu, J., Goetschalckx, M., & McGinnis, L. F. (2007). Research on warehouse operation: A comprehensive review. *European Journal of Operational Research*, *177*(1), 1–21.

Gu, J., Goetschalckx, M., & McGinnis, L. F. (2010). Research on warehouse design and performance evaluation: A comprehensive review. *European Journal of Operational Research*, *203*(3), 539–549.

Hübner, A., Schäfer, F., & Schaal, K. N. (2020). Maximizing profit via assortment and shelf-space optimization for two-dimensional shelves. *Production and Operations Management*, *29*(3), 547–570.

Jacob, R., Márton, P., Maue, J., & Nunkesser, M. (2011). Multistage methods for freight train classification. *Networks*, *57*(1), 87–105.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Lahmar, M., & Benjaafar, S. (2007). Sequencing with limited flexibility. *IIE Transactions*, *39*(10), 937–955.

Lim, A., & Xu, Z. (2009). Searching optimal resequencing and feature assignment on an automated assembly line. *Journal of the Operational Research Society*, *60*(3), 361–371.

Litvak, N., & Vlasiou, M. (2010). A survey on performance analysis of warehouse carousel systems. *Statistica Neerlandica*, *64*(4), 401–447.

Melacini, M., Perotti, S., Rasini, M., & Tappia, E. (2018). E-fulfilment and distribution in omni-channel retailing: A systematic literature review. *International Journal of Physical Distribution & Logistics Management*, *48*(4), 391–414.

Roodbergen, K. J., & Vis, I. F. (2009). A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, *194*(2), 343–362.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, *3*(1-2), 59–66.

SSI Schäfer. (2021). SSI Carrier - Flexibles Taschensorter-System für E-Commerce und Omnichannel. https://www.youtube.com/watch?v=JXy0Z4XuElU

Van Gils, T., Ramaekers, K., Caris, A., & de Koster, R. B. (2018). Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *European Journal of Operational Research*, *267*(1), 1–15.

Van Laarhoven, P. J., & Aarts, E. H. (1987). *Simulated annealing*. In Simulated annealing: Theory and applications (pp. 7–15). Springer.

Zhang, J., Liu, F., Tang, J., & Li, Y. (2019). The online integrated order picking and delivery considering pickers' learning effects for an O2O community supermarket. *Transportation Research Part E: Logistics and Transportation Review*, *123*, 180–199.

Zhong, S., Giannikas, V., Merino, J., McFarlane, D., Cheng, J., & Shao, W. (2022). Evaluating the benefits of picking and packing planning integration in e-commerce warehouses. *European Journal of Operational Research*, *301*(1), 67–81.

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article: Boysen, N., Briskorn, D., Füßler, D., & Stephan, K. (2023). Put it in the bag: Order fulfillment with a pocket sorter system. *Naval Research Logistics (NRL)*, *70*(8), 858–877. https://doi.org/10.1002/nav.22137