OR Spectrum (2019) 41:1087–1121
https://doi.org/10.1007/s00291-019-00566-9

REGULAR ARTICLE

Trolley line picking: storage assignment and order sequencing to increase picking performance

David Füßler[1] · Nils Boysen[1] · Konrad Stephan[1]

Received: 15 March 2019 / Accepted: 17 October 2019 / Published online: 2 November 2019

Abstract
Trolley line picking is a special warehousing system particularly suited to fulfill high-volume demands for heavy stock keeping units (SKUs). In such a system, unit loads of SKUs are positioned along a given path passed by automated trolleys, i.e., carriers hanging from a monorail or automated guided vehicles. Once a trolley reaches a requested SKU, it automatically stops and announces the requested items on a display. This is the signal for an accompanying human picker to put the demanded items onto the trolley. In this way, picking continues until, at the end of the path, the current picking order is complete and the trolley moves onward to the shipping area. Meanwhile, the picker rushes back to meet the subsequent trolley associated with the next picking order. The picking performance of the trolley line system is mainly influenced by the picker's unproductive walking from SKU to SKU during order processing and back to the next trolley when switching to the next order. In this paper, we investigate how the storage assignment of SKUs along the path and the order sequence influence picking performance. Specifically, we explore the positive effect of duplicating SKUs and storing them at multiple positions along the path. We formulate the interdependent storage assignment and order sequencing problems and introduce a decomposition heuristic. Our computational study investigates the solution performance of this procedure and shows that SKU duplication can considerably improve picking performance.

Keywords Warehousing · Trolley line picking · Order fulfillment · Scheduling

✉ Nils Boysen
nils.boysen@uni-jena.de
http://www.om.uni-jena.de/

David Füßler
david.fuessler@uni-jena.de

Konrad Stephan
konrad.stephan@uni-jena.de

[1] Lehrstuhl für Operations Management, Friedrich-Schiller-Universität Jena, Carl-Zeiß-Straße 3, 07743 Jena, Germany

1 Introduction

Under the fierce competition of online retailing, traditional retail chains applying brick-and-mortar sales outlets are forced to streamline their supply chains. Many of them are on their way to become omni-channel (also denoted as multi- or cross-channel) retailers and aim to address customers via a mix of online and offline sales channels to allow customers their most convenient access to the offered products (Agatz et al. 2008). Recent concepts even mix online and offline distribution channels (see, e.g., Hübner et al. (2016a, b)). Within the click-and-collect concept, customers order online and visit the stores to pick up their goods, and the ship-from-store concept advocates to assemble demanded goods directly from store inventory and start the last-mile delivery tours from there.

All these retailing trends increase the pressure on the warehouses and distribution centers of retail chains having to fulfill the demands of sales outlets in a timely and efficient manner. Traditional picker-to-parts systems (where a picker, often supported by a picking vehicle to carry the high volume of goods ordered by a typical brick-and-mortar store, moves through the aisles in order to assemble the current pick list) are often no longer efficient enough in this context. Therefore, a lot of alternative warehousing systems, specifically dedicated to the high-volume-low-mix order structures of brick-and-mortar retail chains, have been introduced in the recent decades. A survey on these systems is provided by the latest review paper of Boysen et al. (2019). The paper on hand is dedicated to trolley line picking systems, whose setup and main operational decision problems are elaborated in the following.

1.1 Trolley line picking

In a trolley line picking system, the orders of brick-and-mortar stores are successively assembled each onto dedicated pallets (or roll cages), which are transported along a given picking path by automated trolleys. Such a trolley can be a carrier hanging from a monorail; an example is the CaddyPick™ system of Swisslog (Swisslog Logistics Automation 2011) (see Fig. 1b). Alternatively, ground-based automated guided vehicles (AGVs) can be applied. While moving along the path, a trolley, accompanied by a human order picker, passes SKUs stored to the left and right of an aisle. Once a trolley reaches a SKU demanded by the picking order currently assigned to the trolley, it automatically stops and indicates the requested items on a display mounted on the trolley and/or by a light beacon pointing on the respective SKU. This is the signal for the picker to put the requested number of items on the trolley and, afterward, to acknowledge the pick. Most of these systems are also equipped with a weighing mechanism to further reduce picking errors. Then, the trolley proceeds its way along the path up to the next demanded SKU and the subsequent pick. In this way, trolley and picker move along the path (a schematic layout is depicted in Fig. 1a) until the final SKU of the current order has been processed and the order is complete. Then, the trolley moves onward unattended to the shipping area, and the picker returns to the subsequent trolley already waiting at the first picking position of its assigned picking order.

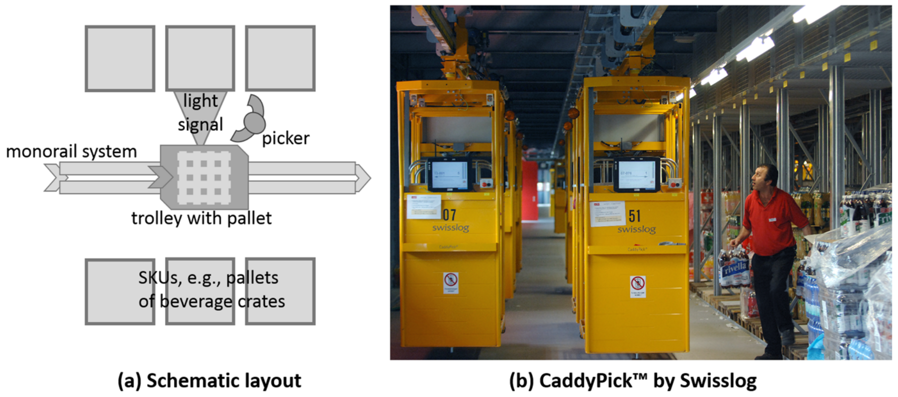

Fig. 1 Trolley line picking

Trolley line picking is especially suited if large orders of many heavy items are to be processed. These prerequisites are fulfilled by many distribution centers supplying brick-and-mortar stores for consumer goods, so that it is not surprising that the reference applications of Swisslog for their CaddyPick system (see Fig. 1b) are a German drugstore chain and a Swiss food retailer (Swisslog 2017). Furthermore, Füßler et al. (2019) report on a trolley line picking system applying ground-based AGVs at a German distribution center that supplies liquor stores, supermarkets, and large restaurants with beverages.

1.2 Levers for improving picking performance and literature review

The key performance indicator for the picking performance in a trolley line system is the walking distance of the picker (see De Koster et al. (2007) for a similar argumentation for traditional picker-to-parts systems). The pure picking time of the picker for retrieving requested items and putting them onto a trolley is fixed once the pick lists are given. Waiting time of the picker for a trolley that has not yet arrived can hardly occur (except for breakdown situations), because the trolleys typically move (slightly) faster than the picker. Their velocity is chosen slow enough to not endanger interfering humans, but fast enough to always be a bit ahead. In this way, a trolley has enough time to position itself at the next picking position and to indicate this location with a beacon, so that the picker can start picking without delay once she has arrived. Finally, since the picker walks unobstructed and does not have to carry SKUs of varying weight, assuming a constant walking speed when moving along the picking path seems a fair approximation of reality. Given these prerequisites, reducing the picker's walking distance directly improves the overall picking performance, which can be influenced by the following levers:

- The *storage assignment* decides on the arrangement of SKUs along the picking path, and these positions determine each order's spread along the path. If the first and last SKU demanded by an order are stored at the first and last storage position, respectively, the picker has to accompany the trolley along the complete path when processing this order. If all SKUs of an order are stored in direct succession instead, the path segment to be accompanied by the picker reduces considerably (as long as the order does not demand all SKUs, which is a rather theoretical case). Thus, the storage assignment determines the distance per order the picker has to accompany the respective trolley. We call this walk of the picker associated with a specific order the order's *escort path*.

Traditionally (i.e., in picker-to-parts warehouses), the storage assignment is a long- to mid-term decision, which remains unaltered for several months (De Koster et al. 2007). Switching the positions of unit loads along the picking path in a trolley line system, however, produces less effort, and there exist automated forklifts that can autonomously relocate the SKUs over night without human assistance, so that the storage assignment can be rearranged according to the transmitted set of picking orders to be processed the next morning. Beyond that, the example system of Swisslog for its CaddyPick system elaborated in Swisslog Logistics Automation (2011) is equipped with automated cranes connecting an automated storage and retrieval system (ASRS), so that replenishment is automated and SKUs can automatically be relocated along the path by a crane. Thus, in trolley line systems the storage assignment is rather an operational problem that can be optimized according to the given order set to be processed during the next work shift. Optimizing storage assignments for a given and deterministic order set rarely occurs. An example case for a traditional picker-to-parts setting and a brief review of the existing literature is provided in Boysen and Stephan (2013). One elementary research question explored in this paper is whether a short-term rearrangement of the storage assignment can improve the picking performance.

- *Order sequence:* Having finished an order at its final picking position along the path, the picker has to return towards the first storage position of the subsequent order. We call this walk, typically executed against the driving direction of the trolley when switching between two successive orders, the *return path*. The return path decreases, the closer the last picking position of the preceding order to the first picking position of its successor. Consequently, optimizing the sequence in which the orders are picked can reduce the return paths and, thus, the total walking distance of the picker.

  In their seminal paper, Gilmore and Gomory (1964) have shown that the resulting order sequencing problem is a special case of the traveling salesman problem on the line (see also Burkard et al. (1998)) and can be solved in polynomial time. In the paper on hand, we apply (a slight modification) of the Gilmore–Gomory procedure in a decomposition approach for order sequencing. Similar adaptions have been made by Bartholdi and Platzman (1986) for order sequencing in carousel racks. We, however, investigate whether storing the same SKU at multiple positions along the path can improve picking performance. With multiple duplicates of SKUs, we also have to choose among multiple alternative escort paths per order. We prove that the resulting optimization problem is strongly NP-hard, but a simple heuristic is shown to deliver excellent results.
- *Picking policy:* Up to now, we have assumed the following simple work protocol, which we call the *order-by-order policy*: Follow the current trolley and execute all indicated picks along the path. Once the current order is complete, return to

the first stop of the subsequent trolley and proceed. This work protocol is very simple and easy to implement in real-life operations. In a previous paper on trolley line picking, however, Füßler et al. (2019) have shown that an alternative protocol called the *order-swapping policy* greatly improves picking performance. Under this work protocol, the picker has additional flexibility and is allowed to process neighboring stops of different trolleys before moving to another section of the line where other orders are waiting for processing. Füßler et al. (2019) introduce a solution procedure, which minimizes the makespan when operating under the order-swapping policy and their computational tests show that performance gains of more than 50% can be obtained compared to the order-by-order policy.

On the negative side, however, order sequencing under the order-swapping policy constitutes a challenging optimization problem due to the blocking of subsequent trolleys, which cannot overtake each other along the path. Furthermore, implementing this work protocol in real-world warehouses also faces great organizational obstacles. The picker always has to consult an information system indicating the next trolley and picking position during order processing to be accessed next. Furthermore, in case of unforeseen events (e.g., some picks are delayed) a re-optimization under real-time conditions is required to adapt the schedule to the altered situation. The order-by-order policy, instead, comes by without organizational overhead. The picker just follows trolley after trolley and picks the indicated SKUs along the path. This is why the aforementioned distribution center for beverages we visited deems the order-swapping policy a rather theoretical solution and prefers the order-by-order policy.

Due to its preferred application in real-life warehouses, we also focus the order-by-order policy in this paper. However, potential performance gains of 50% and more of the order-swapping policy are a challenging benchmark, so that we explore an alternative idea on how to improve picking performance: the duplication of SKUs along the path. By storing (a unit load of) the same SKU at multiple storage positions along the path, the picker receives additional flexibility and can select among multiple alternative escort paths. In this paper, we extend the Gilmore–Gomory procedure (Gilmore and Gomory 1964) and solve the joint order sequencing and escort path selection problem under the order-by-order policy. Our computational study shows that considerable performance gains can be realized by a duplication of SKUs, which are, however, smaller than those promised by the order-swapping policy. On the negative side, duplicate SKUs prolong the picking path within a trolley line system and can—if wrongly placed—lead to even longer escort paths. Therefore, we extend the problem and also optimize the storage assignment of duplicated SKUs along the picking path.

Storing the same SKU at multiple storage position has become an important storage assignment strategy applied in many warehouses of online retailers in the recent years, which is documented by the latest survey paper on warehousing in the e-commerce era by Boysen et al. (2019). E-commerce warehouses, however, face low-volume-high-mix orders of private households, so that these warehouses do not store multiple unit loads of the same SKUs at multiple storage position. Instead, they break down the unit loads into individual items and scatter them all over the warehouse. Optimizing the storage assignment in picker-to-parts warehouses applying the *scattered* *storage* strategy is treated by Weidinger and Boysen (2018). Picker routing for the same warehouse setting is investigated by Weidinger et al. (2019) and Weidinger (2018), respectively. The peculiarities of spreading single items around the warehouse hinder a direct application of these existing approaches to our (completely different) problem setting.

If each new incoming unit load is assigned a new storage position in the warehouse, although another unit load of the same SKU is already stored at another position, we have a storage assignment strategy with long-lasting tradition called *shared storage* (Bartholdi and Hackman 2017). The main intention of this strategy is to save storage space compared to dedicated storage, which has to reserve dedicated shelf space for the maximum inventory level of each SKU in a unique position. Existing research on shared storage in traditional picker-to-parts warehouses is summarized, e.g., by the survey paper of De Koster et al. (2007). In our trolley line system, however, we have a fixed picking path, so that the combined picker routing and storage selection problem (like it is, for instance, solved by Daniels et al. (1998) for the traditional setting) is not directly transferable.

Furthermore, there are some related warehousing systems sharing properties with trolley line picking. One example is AGV-assisted picking, see Boysen et al. (2019). Here, pickers also accompany autonomous picking carts, which relieve them from carrying the items. However, these AGVs are applied in traditional picker-to-parts warehouses, so that instead of a straight picking path we have a more complex aisle structure and the storage assignment is typically not changeably on short notice. More details on these systems are provided by Boysen et al. (2019) and Azadeh et al. (2018). Another related system is the pick-and-pass system (also denoted as pick-to-light or walk-and-pick system). In such a system, a picker moves an order bin, in which a customer order is collected, on a conveyor—typically a non-powered roll conveyor—along a single straight picking aisle. Pick-to-light signals are often applied to indicate the demands for SKUs of the current order, so that the picker can conveniently retrieve the demanded items and put them into the bin while walking along the picking aisle. Solution procedures for storage assignment and order batching have, for instance, been provided by Pan and Wu (2009) and Pan et al. (2015), respectively. Yu and De Koster (2008) and Melacini et al. (2011) present analytical methods to quickly approximate the performance in multi-zone pick-and-pass systems. The main distinction of these systems to trolley line picking is that our trolleys move independently. In pick-and-pass systems, each new order starts and ends at the start of the picking path, whereas in trolley line picking processing an order starts and ends at the first and last picking position of this order, respectively. In addition to the variations of trolley line picking to these related systems elaborated above, SKU duplication has also not been considered here.

To summarize our literature review, we state the gap in the literature for our current paper. There are some related warehousing systems, which, however, slightly vary in their setup, so that we cannot apply existing solution procedures. The only other paper focusing on trolley line picking is the paper of Füßler et al. (2019). They, however, treat trolley lines operated under the order-swapping protocol, which again completely alters the problem setting. Furthermore, we allow SKU duplication which has not been treated before. It can be concluded that this paper treats novel versions of storage assignment and order sequencing for a yet largely unexplored warehousing system.

1.3 Contribution and paper structure

This paper explores how to improve the picking performance of trolley line picking operated under the order-by-order policy. Specifically, we investigate the impact of duplicating SKUs, so that the same SKU can be stored at multiple positions along the picking path. Duplicate SKUs impact both major decision problems to be solved in a warehouse applying trolley line picking:

- *Storage assignment* We optimize the storage locations of SKUs where multiple unit loads of the same SKU can be stored along the picking path. This problem has to ensure that duplicate SKUs do not prolong the selected escort paths of the orders.
- *Order picking* Once the storage assignment of SKUs along the picking path is given, the sequence of order processing is to be determined. Given multiple alternative storage positions for each SKU, the resulting decision problem is a holistic optimization task combining order sequencing, which decides on the return paths of subsequent order pairs, and the selection among each order's alternative escort paths.

We formulate both optimization problems and develop suited optimization procedures. Once these algorithms are available (and proven to deliver an appropriate solution quality in our computational performance tests), we apply them to derive managerial insight. Specifically, we quantify the possible performance gains of both levers (i.e., storage assignment and order picking). Furthermore, we benchmark these improvements with the order-swapping policy, which has previously been investigated for trolley line picking by Füßler et al. (2019).

The remainder of the paper is structured as follows. For didactic reasons, we start with the more restricted optimization problem, which occurs once the storage assignment is already given. Section 2 investigates the joint order sequencing and escort path selection problem. We formulate the resulting optimization problem, prove its computational complexity, and develop a suited (yet very simple) solution procedure. Then in Sect. 3, we extend the problem and also integrate the optimization of the storage assignment. The resulting problem setting is formulated as a mixed-integer programming (MIP) model. For solving this problem, we introduce a straightforward three-stage heuristic procedure. Section 4 evaluates the computational performance of our solution concepts. Then, we skip to managerial aspects in Sect. 4.3 and explore the gains for real-world trolley line systems when applying our two levers for improving picking performance. Finally, Sect. 5 concludes the paper.

2 Order picking for a given storage assignment

This section investigates the joint order sequencing and escort path selection problem, which we dub JOSEPS. This problem presupposes that the storage assignment of

SKUs along the picking path is given. Furthermore, we presuppose that the storage assignment does not contain each SKU only once, but it is possible that (unit loads of) the same SKU are stored at multiple storage locations along the path. Note that the latter is a prerequisite for the picker having the flexibility to choose among multiple escort paths per order. We define the problem (Sect. 2.1), prove computational complexity (Sect. 2.1), provide a MIP model (Sect. 2.2), and introduce a very simple (yet efficient) heuristic solution algorithm (Sect. 2.3).

2.1 Problem definition and assumptions

Consider a trolley line system with a given storage assignment of SKUs along the picking path, where a subset of SKUs may be stored at multiple positions. The storage assignment $S = \langle s_1, \dots, s_m]$ consists of a sequence defining the respective SKU placed at each of the m subsequent and equal sized storage positions along the picking path. Note that we discuss the simplifying assumptions contained in this definition (along with all other prerequisites) below. Within this setup of the trolley line system, a given set of picking orders (i.e., orders placed by brick-and-mortar stores) have to be picked each onto its dedicated trolley. Each order of total order set $O = \{o_1, \dots, o_n\}$ is defined by the subset of SKUs o_j demanded by the respective order. Initially, the picker is assumed to be located at storage position zero at the start of the picking path. Given order set O and storage assignment S, we can preprocess the following two information:

- *Escort paths per order* For each order $o_j \in O$, we can determine the set Γ_j of pairs $(\alpha, \beta) \in \Gamma_j$ referring to the positions of the storage assignment where a potential escort path starts and ends, so that all SKUs demanded by this order can be picked while the picker accompanies the respective trolley along the subpath between these two positions. Thus, we have

$$\Gamma_j = \left\{ \begin{array}{c} (\alpha, \beta) \in \{1, 2, \dots, m\} \times \{1, 2, \dots, m\} : \\ \alpha \leq \beta \wedge \underbrace{\{s_\alpha\} \cup \{s_\beta\} \subseteq}_{\substack{\text{escort path starts} \\ \text{and ends at} \\ \text{demanded SKUs}}} o_j \underbrace{\subseteq \bigcup_{k=\alpha}^{\beta} \{s_k\}}_{\substack{\text{escort path} \\ \text{covers the} \\ \text{whole order}}} \wedge \underbrace{\left| \left\{ \begin{array}{l} k \in \{\alpha, \dots, \beta\} : \\ s_k \in \{s_\alpha\} \cup \{s_\beta\} \end{array} \right\} \right| = \left| \{s_\alpha\} \cup \{s_\beta\} \right|}_{\text{there is no shorter subpath covering the whole order}} \end{array} \right\} \quad (1)$$

  for each order o_j with $j = 1, \dots, n$. The walking distance $d(\alpha, \beta)$ for the picker when moving along an escort path from α to β amounts to $d(\alpha, \beta) = \beta - \alpha$. Note that Γ_j does not contain escort paths (α', β'), where order o_j can also be picked from a shorter escort path $(\alpha, \beta) \in \Gamma_j$ being a subpath of (α', β') (i.e., it is dominated by an escort paths with $(\alpha', \beta') \neq (\alpha, \beta)$ and $\alpha' \leq \alpha$ and $\beta' \geq \beta$).
- *Return paths for all escort path pairs:* If for two subsequent orders o_i and o_j with $i \neq j$ predecessor escort path $(\alpha_i, \beta_i) \in \Gamma_i$ and successor escort path $(\alpha_j, \beta_j) \in \Gamma_j$ are selected, respectively, then the picker has to walk the return path from β_i to α_j of length $d(\beta_i, \alpha_j) = |\beta_i - \alpha_j|$ when switching from the former to the latter.

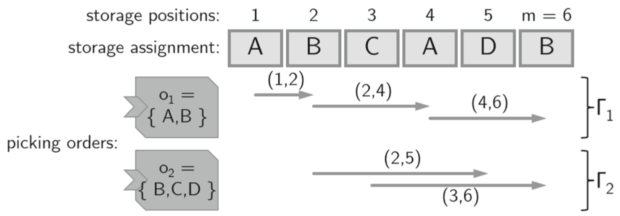

Fig. 2 Example for preprocessing escort and return paths

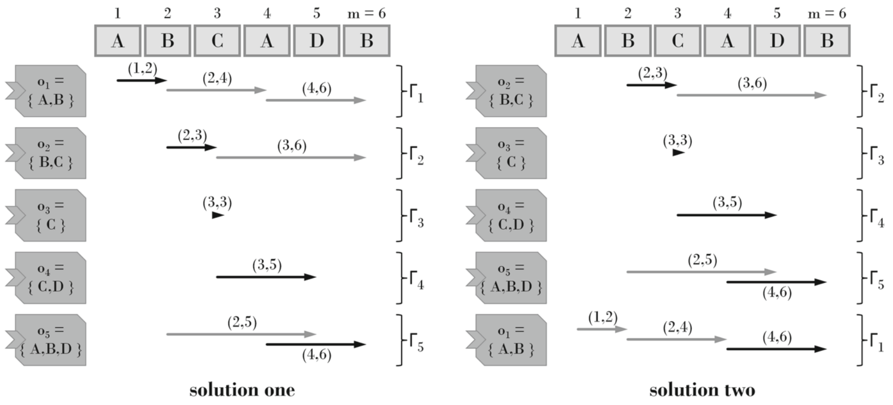

Fig. 3 Example for JOSEPS

Example The calculations of escort and return paths are exemplified within Fig. 2 for storage assignment $S = \langle A, B, C, A, D, B]$. The resulting three escort paths $(1, 2)$, $(2, 4)$, and $(4, 6)$ of order o_1, which demands SKUs A and B, constitute its escort path set $\Gamma_1 = \{(1, 2), (2, 4), (4, 6)\}$. If escort path $(4, 6)$ for order o_1 precedes escort path $(2, 5)$ selected for order o_2 (demanding SKUs B, C, and D), the resulting return path from position 6 to 2 has a length of 4 (unit loads).

Given these preprocessed paths, a feasible solution to JOSEPS selects exactly one escort path $(\alpha_j, \beta_j) \in \Gamma_j$ for each order o_j and defines an order sequence, i.e., a bijective function $\pi : O \rightarrow \{1, 2, \ldots, n\}$, in which the orders (and the selected escort paths) are executed, such that total walking distance

$$\beta_{\pi_1} + \sum_{i=2}^{n} \left(\beta_{\pi_i} - \alpha_{\pi_i}\right) + \sum_{i=1}^{n-1} |\beta_{\pi_i} - \alpha_{\pi_{i+1}}| \qquad (2)$$

is minimized where π_i returns the order processed at sequence position i of order sequence π.

Example Consider $n = 5$ picking orders demanding the following SKUs: $o_1 = \{A, B\}$, $o_2 = \{B, C\}$, $o_3 = \{C\}$, $o_4 = \{C, D\}$, $o_5 = \{A, B, D\}$ and storage assignment $S = \langle A, B, C, A, D, B]$. The resulting escort paths for each order are $\Gamma_1 = \{(1, 2), (2, 4), (4, 6)\}$, $\Gamma_2 = \{(2, 3), (3, 6)\}$, $\Gamma_3 = \{(3, 3)\}$, $\Gamma_4 = \{(3, 5)\}$,

$\Gamma_5 = \{(2, 5), (4, 6)\}$. Fig. 3 depicts two different solutions based on different escort path selections (indicated by the black arrows) and order sequences. Solution one results in a total walking distance of 8 (i.e., one distance unit from the start of the path towards the first escort path plus the sum of selected escort paths of 6 plus a return path of one distance unit from the end of order o_4 back to the start of o_5). In solution two, the picker has to walk along 12 unit loads.

To extract the elementary problem characteristics, we presuppose a trolley line system in its very basic form. Consequently, JOSEPS is based on the following (simplifying) assumptions and prerequisites:

- Large real-world trolley line systems meander along an S-shaped path through a multi-aisle warehouse. We exclude the possibility of shortcuts from one aisle to the other, so that the picker can only walk back and forth the picking path. Without shortcuts, we can (without loss of generality) assume that the picking path is a straight line. If shortcuts are possible, this additional flexibility could lead to even larger improvement. However, we leave exploring this matter up to future research.
- For a matter of convenience, we assume that storage positions are only arranged on one side of the picking path. Extending JOSEPS and all our solution approaches to a two-sided storage pattern along both sides of the picking path is truly straightforward, but considerably complicates the notation.
- We assume that all SKUs are stored in standardized unit loads, e.g., on EUR-pallets, so that we have equidistant storage positions along the path and can measure distances according to the standardized size of unit loads.
- Each unit load is assumed to contain enough items to satisfy all orders (or, if not, replenishment is fast enough to not impede order picking). Thus, we only have to consider whether an order requires a specific SKU, but not the actual number of demanded items.
- Each order is picked onto a single trolley traveling along the path. The peculiarities if excessively large orders require multiple subsequent trolleys are not considered.
- We only consider a single picker. In large real-world systems, multiple pickers can be employed in multiple subsequent path segments. While the basic problem structure remains basically unaltered, we leave the additional aspects caused by the necessary coordination of pickers up to future research.
- Initially, the picker is assumed to be positioned at storage positions zero (i.e., start of the picking path), and picking is assumed to end once the last picking position of the final order is reached. Note that an alternative is to end order processing once the picker has finally returned to the depot. Appendix C discusses both settings in detail.
- Due to the trolleys' slightly faster speed compared to the picker, waiting time of the picker for a trolley that has not yet arrived at the next picking position is not considered.
- We presuppose a constant walking speed of the picker, so that minimizing the total walking distance is an appropriate proxy for maximizing the picking performance (see Sect. 1.2).

Given this problem setting, solving JOSEPS turns out non-trivial, which follows from Theorem 1.

Table 1 Notation for JOSEPS-MIP

O	Set of picking orders including dummy order 0 (indices i, j)
Γ_j	Set of potential escort paths for order j (indices k, l)
α_k	Start of escort path k
β_k	End of escort path k
y_{ji}	Binary variable: 1, if order i is processed directly after order j, 0 otherwise
x_{jk}	Binary variable: 1, if escort path k of Γ_j is chosen for order j, 0 otherwise
p_j	Continuous variable: sequence position of order j
Δ_i	Continuous variable: distance of return path towards order i

Theorem 1 *JOSEPS is NP-hard in the strong sense, even if*

- *every SKU occurs at most* 3 *times in the storage assignment and*
- *every order contains at most* 2 *SKUs.*

Proof See "Appendix A." □

2.2 A mixed-integer program

Applying the notation summarized in Table 1, JOSEPS can be formulated as a mixed-integer program (dubbed JOSEPS-MIP) consisting of objective function (3) and constraints (4)– (12).

JOSEPS-MIP:

$$\text{Minimize } F = \sum_{i \in O \setminus \{0\}} \Delta_i + \sum_{j \in O} \sum_{k \in \Gamma_j} x_{jk} \cdot (\beta_k - \alpha_k) \tag{3}$$

subject to

$$\sum_{i \in O \setminus \{0\}} y_{ji} = 1 \quad \forall j \in O \tag{4}$$

$$\sum_{j \in O} y_{ji} = 1 \quad \forall i \in O \setminus \{0\} \tag{5}$$

$$p_j + 1 \leq p_i + M \cdot (1 - y_{ji}) \quad \forall j \in O, i \in O \setminus \{0\} \tag{6}$$

$$p_0 = 0 \tag{7}$$

$$\sum_{k \in \Gamma_j} x_{jk} = 1 \quad \forall j \in O \tag{8}$$

$$\Delta_i \geq \sum_{k \in \Gamma_j} \beta_k \cdot x_{jk} - \sum_{l \in \Gamma_i} \alpha_l \cdot x_{il} - M \cdot (1 - y_{ji}) \quad \forall j \in O, i \in O \setminus \{0\} \tag{9}$$

$$\Delta_i \geq \sum_{l \in \Gamma_i} \alpha_l \cdot x_{il} - \sum_{k \in \Gamma_j} \beta_k \cdot x_{jk} - M \cdot (1 - y_{ji}) \quad \forall\, j \in O, i \in O \backslash \{0\} \tag{10}$$

$$y_{ji}, x_{jk} \in \{0; 1\} \quad \forall\, j \in O;\ i \in O \setminus \{0\};\ k \in \Gamma_j \tag{11}$$

$$\Delta_j, p_j \geq 0 \quad \forall\, j \in O \tag{12}$$

Objective function (3) minimizes the sum of distances of all selected escort and return paths. Constraints (4) and (5) ensure that each picking order is processed exactly once, while (6) and (7) eliminate sub-tours. Furthermore, for each order exactly one escort path has to be selected (8). The distance of the return path between two adjacent orders j and i is determined by constraints (9) and (10). Finally, (11) and (12) define the domains of the variables.

2.3 A straightforward heuristic

This section presents a straightforward heuristic solution procedure for JOSEPS. It is a simple priority-rule-based approach (dubbed PRIO), which selects the shortest escort path per order and solves the remaining order sequencing problem with (a slight modification of) the Gilmore–Gomory procedure (Gilmore and Gomory 1964). Our computational tests in Sect. 4.2 will show that among all instances solved to optimality (i.e., by standard solver Gurobi solving JOSEPS-MIP), this simple approach is able to find optimal solutions in 85% of these instances. Over all instances, the gap is well below 1% and the solution time is barely measurable. Further tests with local search procedures and metaheuristics (i.e., swapping escort paths and solving the resulting Gilmore–Gomory instances) have shown that the small improvements over PRIO solutions cannot justify the considerably longer solution times. Thus, we decided to not detail these unsuccessful approaches and only present PRIO as our suggestion for solving large real-world instances of JOSEPS. The two basic stages of PRIO are the following:

In the *first stage* of our rule-based approach, we select the escort path with the smallest distance for each order. That is, escort path $\text{argmin}_{(\alpha,\beta)\in\Gamma_j}\{d(\alpha, \beta)\}$ is chosen for each order o_j with $j = 1, \ldots, n$.

For this selection, we have to solve the remaining order sequencing problem of the selected escort paths in the *second stage*. At first glance, we can directly apply the famous solution procedure of Gilmore and Gomory (1964) for this task, which sequences point-to-point transport requests along a line, such that the total travel distance is minimized in $O(n^2)$. This traveling salesman problem on the line (see Burkard et al. (1998)), however, looks for a round trip, i.e., a tour through all selected escort paths along the line with a final return to the start position. Within JOSEPS, however, the picker is assumed to start at the beginning of the picking path and end at the last storage position of the last order. Thus, we have to solve the path version of the problem, which can be obtained by embedding the approach of Gilmore and

Gomory (1964) in an iterative solution procedure introduced by Füßler et al. (2019). This iterative procedure has a run time of $O(n^3)$ and is detailed by Algorithm 1.

Algorithm 1: Iterative Gilmore–Gomory approach of Füßler et al. (2019)

```
   input: set of n orders with given escort paths
1  add virtual start order $o_0 = (0, 0)$
2  initialize best order sequence $\pi^*$
3  foreach order $o_i$ do
       // construct optimal sequence, which ends at $\beta_i$
4      add virtual order $o^i_{n+1} = (\beta_i, 0)$ from the requested SKU of customer $i$ to the start of the
       picking path
5      apply the approach of Gilmore and Gomory (1964) to determine an optimal tour
6      delete order $o^i_{n+1}$ from the tour
7      divide the tour at $o_0$ and receive sequence $\pi$
       // check for new best solution
8      if $F(\pi) < F(\pi^*)$ then
9          $\pi^* = \pi$
10     end
11 end
   return: best customer sequence $\pi^*$
```

The basic idea is to solve the problem for each possible final escort path separately. If a specific final escort path is evaluated, an extra job is inserted into the job set of the Gilmore–Gomory approach that connects the end position of the currently evaluated escort path with the start of the picking path. In this way, all possible final escort paths (jobs) are evaluated and the best solution among them is returned.

3 Including the optimization of the storage assignment

This section extends JOSEPS and also integrates the storage assignment of SKUs along the picking path as an additional decision variable. If the path has a storage capacity that exceeds the total number of SKUs, so that some SKUs can be duplicated and stored at multiple positions along the path, our extended problem setting also includes the decision which SKUs should be duplicated. We call this problem the joint order sequencing and storage assignment problem and dub it JOSSAP. First, we describe the problem by providing a MIP model (Sect. 3.1) and then we introduce a straightforward three-stage solution procedure (Sect. 3.2).

3.1 Problem definition

The basic extension that converts previous problem setting JOSEPS, where for a given storage assignment of SKUs along the path the processing sequence of orders and the escort path selection is determined, into JOSSAP is that the storage assignment $S = \langle s_1, \ldots, s_m]$ is now a decision variable. The picking path has a given capacity m

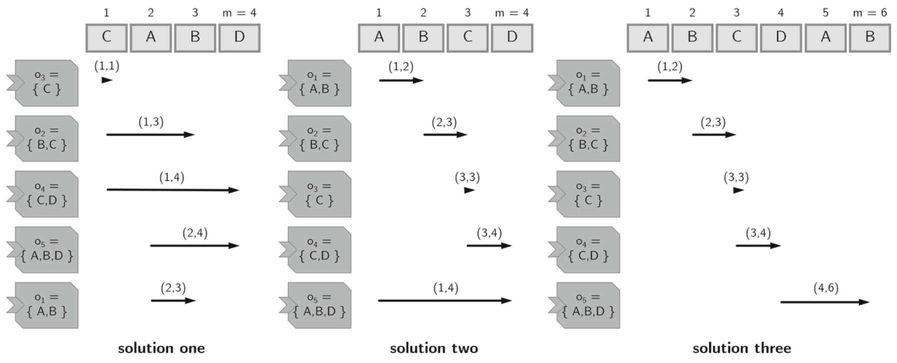

Fig. 4 Example storage assignment and joint order scheduling JOSSAP

of storage positions for unit loads of SKUs. If the total SKU set is denoted by U, with $m \geq |U|$, we have the following two conditions for a feasible storage assignment:

- each storage position receives a SKU, so that we have $s_i \in U$ for all $i = 1, \ldots, m$, and
- each SKU is contained in the picking path at least once, so that we have $U = \bigcup_{i=1}^{m} \{s_i\}$.

Except for this definition of a feasible storage assignment S that additionally has to be determined, JOSSAP has the same problem structure and objective as JOSEPS.

Example Consider the example introduced in Fig. 3 with $n = 5$ customer orders demanding four SKUs A, B, C, and D as follows: $o_1 = \{A, B\}$, $o_2 = \{B, C\}$, $o_3 = \{C\}$, $o_4 = \{C, D\}$, $o_5 = \{A, B, D\}$. In solutions one and two of Fig. 4, our picking path has a capacity of $m = 4$, so that each SKU receives exactly one storage position. Note that this excludes the possibility of alternative escort paths per order, so that the decisions to be taken by JOSSAP are the storage assignment of SKUs along the path and the processing sequence of orders. In solution one, the storage assignment is $S = \langle C, A, B, D]$ and the corresponding order sequence $\pi = \langle o_3, o_2, o_4, o_5, o_1]$. Both together lead to a total walking distance of 15 unit loads for the picker. Note that this solution consists of one distance unit towards the start of the first escort path plus total escort path lengths of 8 plus total return path lengths of 6. Alternative solution two with storage assignment $S = \langle A, B, C, D]$ and order sequence $\pi = \langle o_1, o_2, o_3, o_4, o_5]$ leads to a total walking distance of 10 (unit loads). Finally, solution three has a larger storage capacity of $m = 6$. The storage assignment is $S = \langle A, B, C, D, A, B]$, whereas the order processing sequence is the same as in solution two. The additional flexibility, however, allows to select a different escort path for the final order and the total walking distance reduces to 6 (unit loads).

Applying the notation summarized in Table 2, JOSSAP can be formulated as a MIP, which we dub JOSSAP-MIP. It consists of objective function (13) and constraints (14)–(29).

Table 2 Notation of JOSSAP-MIP

O	Set of customer orders including dummy order 0 (indices j, i)
U	Set of SKUs (index u)
m	Unit load capacity of the picking path (index t)
o_j	Set of requested SKUs of customer order j
y_{ji}	Binary variable: 1, if order i is processed after order j, 0 otherwise
x_{ut}	Binary variable: 1, if SKU u is stored at position t, 0 otherwise
z_{jut}	Binary variable: 1, if order j receives SKU u from position t, 0 otherwise
α_j	Continuous variable: start position of the escort path selected for order j
β_j	Continuous variable: end position of the escort path selected for order j
p_j	Continuous variable: sequence position of order j
Δ_i	Continuous variable: distance of return path towards order i

JOSSAP-MIP:

$$\text{Minimize } F = \sum_{i \in O \setminus \{0\}} \Delta_i + \sum_{j \in O \setminus \{0\}} (\beta_j - \alpha_j) \tag{13}$$

subject to

$$\sum_{i \in O \setminus \{0\}} y_{ji} = 1 \quad \forall j \in O \tag{14}$$

$$\sum_{j \in O} y_{ji} = 1 \quad \forall i \in O \setminus \{0\} \tag{15}$$

$$p_j + 1 \leq p_i + |O| \cdot (1 - y_{ji}) \quad \forall j \in O, i \in O \setminus \{0\} \tag{16}$$

$$p_0 = 0 \tag{17}$$

$$\sum_{u \in U} x_{ut} = 1 \quad \forall t = 1, \ldots, m \tag{18}$$

$$\sum_{t=1}^{m} x_{ut} \geq 1 \quad \forall u \in U \tag{19}$$

$$\beta_0 = 0 \tag{20}$$

$$\alpha_j \leq t + m \cdot \left(1 - \sum_{u \in o_j} z_{jut}\right) \quad \forall j \in O \setminus \{0\}, t = 1, \ldots, m \tag{21}$$

$$t \cdot \sum_{u \in o_j} z_{jut} \leq \beta_j \quad \forall j \in O \setminus \{0\}, t = 1, \ldots, m \tag{22}$$

$$\beta_j - \alpha_j \geq |o_j| \quad \forall j \in O \backslash \{0\} \tag{23}$$

$$\sum_{j \in O \backslash \{0\}} z_{jut} \leq |O| \cdot x_{ut} \quad \forall u \in U; t = 1, \ldots, m \tag{24}$$

$$\sum_{t=1}^{m} z_{jut} \geq 1 \quad \forall j \in O \backslash \{0\}, u \in o_j \tag{25}$$

$$\Delta_i \geq \beta_j - \alpha_i - m \cdot (1 - y_{ji}) \quad \forall j \in O, i \in O \backslash \{0\} \tag{26}$$

$$\Delta_i \geq \alpha_i - \beta_j - m \cdot (1 - y_{ji}) \quad \forall j \in O, i \in O \backslash \{0\} \tag{27}$$

$$z_{jut}, y_{ji}, x_{ut} \in \{0; 1\} \quad \forall j, i \in O,\ u \in U, t = 1, \ldots, m \tag{28}$$

$$\alpha_j, \beta_j, \Delta_{j,i}, p_j \geq 0 \quad \forall j, i \in O \tag{29}$$

Objective function (13) minimizes the picker's total walking distance for the chosen escort and return paths. Constraints (14) and (15) guarantee that each order is processed exactly once where (16) and (17) eliminate sub-tours. Furthermore, it has to be ensured that each storage position along the picking path receives exactly one SKU (18) and that each SKUs is stored at least once (19). The start α_j and end β_j of each customer order's escort path are determined by constraints (21), (22), and (23). Note that (23) is actually not necessary to formulate the problem, but provides a good lower bound on the length of the escort path. Our computational tests have shown that this increases the performance of a standard solver considerably (see Sect. 4.2). Constraints (25) guarantee that each requested SKU u of an order j has to be picked at the picking path position t if the SKU is stored at that position (24). The distance of the return path between two customer orders is determined by (26) and (27). The domains of binary and continuous variables are set by (28) and (29), respectively.

Finally, we investigate the complexity status of JOSSAP. Note that NP-hardness of JOSSAP does not automatically follow from the previous NP-hardness proof for JOSEPS. At least theoretically, it is possible that JOSEPS turns out being NP-hard only for very specially structured storage assignments that never occur in optimal solutions of JOSSAP. Thus, a formal (yet straightforward) proof is required.

Theorem 2 *JOSSAP is NP-hard in the strong sense.*

Proof See "Appendix B." □

3.2 A three-stage heuristic approach

For solving JOSSAP, this section introduces a straightforward three-stage heuristic procedure. On the first stage, depending on total storage capacity m, we determine how often each SKU should be stored along the path. Then on stage two, we apply a simple simulated annealing (SA) scheme to iterate through different storage assignments applying swap moves. For evaluating each storage assignment within an iteration of the SA, we apply our PRIO heuristic introduced in Sect. 2.3 on stage three to solve the resulting JOSEPS subproblem. In the following, we describe the first two stages of this approach in more detail.

On the *first stage*, we determine how often each SKU $u \in U$ should be stored along the picking path. Given $m \geq |U|$ storage positions, we simply rank all SKUs with a priority rule and duplicate SKUs in the given succession until all storage positions are taken. Each SKU receives $\lfloor \frac{m}{|U|} \rfloor$ storage positions and the first $m - \lfloor \frac{m}{|U|} \rfloor \cdot |U|$ SKUs on the list receive an additional storage position. For ranking the SKUs, we apply the following four priority rules:

- *MostOrders:* An obvious priority rule is to prioritize those SKUs that occur in many orders. Frequently requested SKUs can then be placed at multiple central storage positions, so that they are easily reachable for the picker. Thus, one approach is to order the SKUs in non-increasing order according to priority value $\phi_u^{MO} = |\{o_i \in O | u \in o_i\}|$ for all $u \in U$.
- *MostRelatedSKUs:* A similar idea is to prioritize those SKUs that occur jointly in orders with many different other SKUs, so that we list the SKUs in non-increasing order according to priority value $\phi_u^{MRS} = |\{u' \in U | \exists o_i \in O : \{u, u'\} \subseteq o_i\}|$ for all $u \in U$.
- *LeastOrders:* In contradiction to the previous rules, it may also be advantageous to duplicate SKUs that are rarely demanded. These SKUs can then, for instance, be placed at both ends of the picking path, so that at least some flexibility is available when having to access these SKUs, while frequently requested SKUs can be placed in more central positions where they are easily reachable. Under this priority rule, we rank SKUs in non-decreasing order according to priority value $\phi_u^{LO} = |\{o_i \in O | u \in o_i\}|$ for all $u \in U$.
- *Random:* As an additional benchmark, we also apply the random rule, which assigns a uniformly distributed random number between 0 and 1 to each SKU. In this case, the SKUs receiving additional storage space are determined just by chance.

Note that our selection of priority rules is not exhaustive. However, we leave the development and tests of further rules up to future research.

Example Given the five orders $o_1 = \{A, B\}$, $o_2 = \{B, C\}$, $o_3 = \{C\}$, $o_4 = \{C, D\}$, $o_5 = \{A, B, D\}$ introduced in Fig. 3 and a storage capacity of $m = 6$, priority rule MostOrders selects SKUs B and C to receive two storage positions each, whereas SKUs A and D receive only a single storage position along the picking path. Applying priority rule MostRelatedSKUs (LeastOrders) selects SUKs B and D (A and D) to receive two storage positions.

After having determined the number of occurrences per SKU along the picking path, we apply a straightforward SA procedure on the *second stage*. This procedure starts with a random storage assignment where the occurrences of each SKU determined on stage one are randomly shuffled among the storage positions along the picking path. The resulting storage assignment is then evaluated by applying our PRIO procedure introduced in Sect. 2.3 for solving the resulting JOSEPS subproblem. Note that this evaluation step with PRIO constitutes *stage three* of our three-stage heuristic for JOSSAP. Given this start solution (i.e., storage assignment S) and the corresponding objective value $Z(S)$, we apply a straightforward SA scheme as is defined by Kirkpatrick et al. (1983). To obtain a neighboring solution, i.e., storage assignment S' from current solution S, we simply swap the SKUs stored at two randomly selected storage positions. Yet again new solution S' is evaluated by our PRIO approach determining the associated objective value $Z(S')$. By applying the traditional probability scheme of Aarts et al. (1997)

$$\text{Prob}(S' \text{ is accepted}) = \begin{cases} 1, & \text{if } Z(S') \leq Z(S) \\ \exp(\frac{Z(S)-Z(S')}{T^s}), & \text{otherwise} \end{cases} \tag{30}$$

it is decided if new storage assignment S' is accepted or not. For steering our SA, we apply the straightforward static cooling scheme defined in 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 100, 1, and 0.9995, respectively. Note that preliminary tests not reported in this paper 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: $T^s := T^s \cdot d_r$. In this way, acceptance of worse solutions becomes less likely the more storage assignments have been evaluated. The procedure terminates if either temperature T^s equals stop value T_e or a total computational time of 300 CPU seconds is reached. Furthermore, we implemented our SA in a multi-threaded manner, so that on each available processor core another SA with a random SKU arrangement is initialized and, finally, the best solution found during the search process is returned.

4 Computational study

This section is dedicated to our computational study. We, first, define the generation scheme of our test instances (see Sect. 4.1). Then in Sect. 4.2, we evaluate the computational performance both for JOSEPS and JOSSAP by benchmarking our heuristic procedures with standard solver Gurobi. Once an adequate solution performance of our algorithms has been shown, we apply them to answer our main research questions defined in Sect. 1.3. Specifically, Sect. 4.3 evaluates the improvement of picking performance of our two optimization procedures and benchmarks them with the gains promised by the order-swapping policy investigated by Füßler et al. (2019).

4.1 Instance generation

The optimization problems treated in this paper have not been considered in the scientific literature yet, so that there is no existing testbed. Thus, we have to derive new instances for which we apply the following instance generator. Table 3 lists the different parameters and their respective values handed over to our generator. The values for the number of orders n, the number of SKUs $|U|$, and interval $[\underline{\gamma}; \overline{\gamma}]$, from which

Table 3 Parameter values for instance generation

Parameter	Description	Value(s)
n	Number of orders	10, 30, 50
$\|U\|$	Number of SKUs	10, 30, 50
$[\underline{\gamma}; \overline{\gamma}]$	Interval of SKUs per order	[1; 3], [7; 10], [1; 10]

the number of demanded SKUs per order is randomly drawn, are combined in a full factorial manner which leads to 27 unique parameter settings. For each setting instance generation is repeated 10 times, which leads to a total of 270 test instances.

Each instance is obtained as follows. First, each of the n orders receives a random integer number drawn from interval $[\underline{\gamma}; \overline{\gamma}]$ defining the number of demanded SKUs contained in the respective order. Each of these SKUs is then randomly determined by drawing a random integer between 1 and $|U|$. Furthermore for each JOSEPS instance, we also have to derive an initial storage assignment. This is done by drawing an integer number of unit loads per SKU from interval [1; 3]. All resulting unit loads of SKUs are then randomly arranged along the picking path. When solving JOSSAP (and not explicitly stated otherwise), we assume a storage capacity of $m = |U| \cdot 1.1$, so that 10% of all SKUs can be duplicated. All random numbers follow a uniform distribution.

All procedures have been coded in Visual Basic based on Microsoft's .NET Framework 4.6, and all computations have been performed on a personal computer with Intel Core i7-3770 processor and 4x3.4 GHz clock speed as well as 8 GB DDR-3 RAM. As a standard solver we apply Gurobi Optimizer 8.0.

4.2 Computational performance

In this section, we evaluate the solution performance of our heuristic approaches and benchmark them with standard solver Gurobi solving the MIPs for both problem settings JOSEPS and JOSSAP. We start with the more restricted problem version JOSEPS, where we optimize the order processing for a given storage assignment. The two competitors for this problem are standard solver Gurobi solving JOSEPS-MIP (see Sect. 2.2) and our heuristic PRIO (see Sect. 2.3). The performance criteria we apply to evaluate both solution approaches are summarized in Table 4. Since JOSEPS is an operational decision problem, both solution approaches receive a solution time of 300 seconds. The results of this performance test solving our 270 instances elaborated in Sect. 4.1 are summarized in Table 5. The following findings can be derived from these results:

- Standard solver Gurobi is able to solve all small-sized instances with $n = 10$ orders to optimality and finds feasible solutions for all JOSEPS instances. With larger order numbers $n \geq 30$, however, Gurobi's solution times increase considerably and most instances (i.e., 57.2% of all instances with $n \geq 30$) cannot be solved to proven optimality within the given time frame.
- Our straightforward PRIO approach, which selects the shortest escort path for each order and solves the resulting sequencing problem with the Gilmore–Gomory

Table 4 Performance criteria

Criteria	Description
#best	Number of instances where the respective procedure found the best solution among all competitors
#feas	Number of instances where a feasible solution is found
#opt	Number of instances where the solution method found a proven optimal solution
$\varnothing gap^{best}$	Average gap compared to best solution in %
$\varnothing gap^{LB}$	Average gap to best derived lower bound in %
∅CPUs	Average computational time in CPU seconds

Table 5 Computational performance for JOSEPS

n	$\|U\|$	Gurobi solving JOSEPS-MIP					PRIO			
		#opt	#feas	#best	$\varnothing gap^{best}$	∅CPUs	#opt	#best	$\varnothing gap^{best}$	∅CPUs
10	10	30	30	30	0	0.69	24	24	1.27	0
10	30	30	30	30	0	0.68	27	27	1.11	0
10	50	30	30	30	0	0.62	23	23	1.04	0
Total		90	90	90	0	0.66	74	74	1.14	0
30	10	20	30	30	0	131.61	19	28	0.03	0
30	30	22	30	30	0	121.28	19	27	0.06	0
30	50	20	30	30	0.05	142.22	16	25	0.06	0
Total		62	90	90	0.02	131.7	54	80	0.05	0
50	10	4	30	29	1.36	277.05	4	29	0	0
50	30	6	30	29	0.52	252.46	5	28	0.01	0
50	50	5	30	27	1.02	259.79	5	28	0.01	0
Total		15	90	85	0.97	263.1	14	85	0.01	0

procedure (Gilmore and Gomory 1964), leads to astonishingly good results. For the smallest instances with $n = 10$ orders, the average optimality gap is slightly above 1%, and this gap considerably decreases for larger instances $n \geq 30$. In 85% of all instances where an optimal solution can be obtained, PRIO also finds an optimal solution. In spite of these excellent objective values, the solution time of PRIO is barely measurable. This is good news for practitioners, because a straightforward solution approach is available to solve even instances of real-world size with excellent solution performance (almost) in real time. Note that we also tested first-fit and best-fit local search procedures and a metaheuristic based on varying the selection of escort paths and solving the resulting Gilmore–Gomory instances. The small improvements of these approaches over PRIO's objective values cannot justify the considerably larger computational times, so that we abstain from a detailed description of these unsuccessful attempts.

Next, we switch to JOSSAP and consider the storage assignment as an additional decision variable. First, we compare our four priority rules to decide on the selection of SKUs to be duplicated on stage one of our three-stage heuristic (see Sect. 3.2). Each of these priority rules is applied within our three-stage solution procedure, and Table 6 reports the results of this comparison. Note that since the solution times do not vary just by altering the priority rule on stage one, we do not report the required CPU seconds. Solution time of our three-stage approach is addressed in a later test. Further note that we specifically focus large-sized instances with $n = 50$ orders, $|U| \in \{10, 30\}$ and $[\underline{\gamma}; \overline{\gamma}] \in \{[1; 3], [7; 10]\}$, because here the differences among our priority rules (which for the smaller instances also exist, but are less distinct) become more obvious.

The results summarized in Table 6 suggest the following findings. Priority rule LeastOrders, which prioritizes those SKUs contained in the least number of orders, cannot even outperform a random selection of duplicated SKUs. Obviously, the basic idea of LeastOrders to place frequently requested SKUs in the center and less-frequented SKUs on both ends of the path does not lead to competitive storage assignments. Priority rules MostRelatedSKUs and MostOrders considerably outperform random selections, and on average over all tested parameter settings MostRelatedSKUs leads to the best storage assignments. Thus, for all further tests we only apply this rule, which prioritizes SKUs ordered jointly together with many other SKUs in the order set.

This leads us to our final performance test on JOSSAP, where we compare our three-stage heuristic procedure (see Sect. 3.2) with standard solver Gurobi solving JOSSAP-MIP. This test is summarized in Table 7, and the following conclusions can be drawn from these results:

- Gurobi struggles with solving JOSSAP-MIP. It is only able to prove a single optimal solution among all 330 instances. However, at least the standard solver is able to find feasible solutions for most of the instances, i.e., 315 out of 330 or 95.5%. For the smallest instances with $n = 10$ orders, it is even able to contribute some best solutions. However, the larger the instances, the larger the gaps to the best solutions; for the largest instances with $n = 50$ orders the average gap to the best solutions amounts to a remarkable 129.4% and also the gaps to its own lower bound are large. Unfortunately, these results cannot considerably be improved by spending Gurobi more solution time. Additional tests have shown that even a solution time of one hour per instances leaves a considerable number of the largest instances without an acceptable solution.
- Our three-stage heuristic, instead, delivers much better results. Only in 18 of the smallest instances with $n = 10$ orders Gurobi finds a better solution, but the average gap is only slightly above 3%. In all instances with $n \geq 30$, our three-stage approach outperforms Gurobi and finds a better solution with considerable gaps for Gurobi. Note that we also equipped our three-stage heuristic with more solution time, by reheating the SA multiple times and restarting it with different SKU selections on stage one. This improves the results, but not considerably enough to warrant a detailed description.

It can be concluded that JOSSAP is a challenging optimization task including four interdependent decisions: selection of duplicated SKUs, storage assignment, selection

Table 6 Computational performance for JOSSAP for different priority rules deciding on the duplication of SKUs

n	$\lvert U\rvert$	$[\underline{\gamma};\overline{\gamma}]$	m	MostRelatedSKUs		MostOrders		LeastOrders		Random	
				#best	$\varnothing gap^{best}$	#best	$\varnothing gap^{best}$	#best	$\varnothing gap^{best}$	#best	$\varnothing gap^{best}$
50	10	[7; 10]	$1.1 \cdot \lvert U\rvert$	8	0.21	0	1.86	8	0.21	5	0.66
50	10	[7; 10]	$1.3 \cdot \lvert U\rvert$	2	0.84	1	1.83	5	0.34	4	1.07
50	10	[7; 10]	$1.5 \cdot \lvert U\rvert$	2	1.44	2	1.81	4	1.03	2	1.49
Total				12	0.83	3	1.83	17	0.53	11	1.07
50	10	[1; 3]	$1.1 \cdot \lvert U\rvert$	9	0.92	5	2.86	0	12.13	2	6.74
50	10	[1; 3]	$1.3 \cdot \lvert U\rvert$	9	0.45	7	2.58	0	29.41	1	14.75
50	10	[1; 3]	$1.5 \cdot \lvert U\rvert$	6	0.92	6	2.9	0	47.14	0	24.95
Total				24	0.76	18	2.78	0	29.56	3	15.48
50	50	[7; 10]	$1.1 \cdot \lvert U\rvert$	2	1.26	3	1	2	0.97	4	1.07
50	50	[7; 10]	$1.3 \cdot \lvert U\rvert$	5	0.72	3	0.82	0	3.43	3	1.88
50	50	[7; 10]	$1.5 \cdot \lvert U\rvert$	7	0.47	3	1.61	0	8.27	0	5.52
Total				14	0.82	9	1.14	2	4.22	7	2.82
50	50	[1; 3]	$1.1 \cdot \lvert U\rvert$	7	3.99	3	5.05	0	58.92	0	40.74
50	50	[1; 3]	$1.3 \cdot \lvert U\rvert$	6	4.03	4	5.75	0	128.2	0	58.02
50	50	[1; 3]	$1.5 \cdot \lvert U\rvert$	5	1.69	5	1.34	0	147.88	0	56.92
Total				18	3.24	12	4.05	0	111.67	0	51.89

Table 7 Computational performance for JOSSAP

n	$\|U\|$	Gurobi solving JOSSAP-MIP						Three-stage-heuristic			
		#opt	#feas	#best	$\varnothing gap^{LB}$	$\varnothing gap^{best}$	∅CPUs	#best	$\varnothing gap^{LB}$	$\varnothing gap^{best}$	∅CPUs
10	10	1	30	14	42.48	4.73	296.25	19	41.8	3.0008	19.05
10	30	0	30	8	53.5	17.02	300.09	24	47.59	3.3317	27.19
10	50	0	30	9	48.66	22.18	300.2	23	41.38	3.3654	33.32
Total		1	90	31	48.21	14.64	298.85	66	43.59	3.23	26.52
30	10	0	30	0	64.94	11.11	300.06	30	61.15	0	87.14
30	30	0	30	0	80.74	74.98	300.2	30	64.42	0	128.21
30	50	0	30	0	85.94	131.24	300.47	30	64.84	0	157.66
Total		0	90	0	77.21	72.44	300.24	90	63.47	0	124.34
50	10	0	30	0	69.02	14.58	300.18	30	64.81	0	233.86
50	30	0	26	0	86.94	92.04	300.3	30	75.4	0	270.35
50	50	0	24	0	91.06	241.81	300.72	30	68.56	0	281.49
50	100	0	25	0	93.94	134.33	302.6	30	84.95	0	293.32
50	200	0	30	3	95.38	164.25	310.61	30	87.9	0	301.45
Total		0	135	3	87.27	129.4	300.4	150	76.32	0	276.09

of escort paths, and order sequence. Unfortunately, standard solver Gurobi is not able to solve instances of reasonable size for this holistic problem. For our three-stage approach (applying our PRIO heuristic solving the JOSEPS subproblems containing two of the four decisions with excellent performance), instead, our computational tests indicate a reasonably good performance, so that we apply this approach for investigating some managerial issues on trolley line picking in the following section.

4.3 Managerial aspects

In this section, we aim to derive some decision support for warehouse managers having to setup and operate a trolley line picking system. Specifically, we want to answer the following three questions:

(1) How does a duplication of SKUs impact the picking performance?
(2) Can the gains by optimizing a trolley line picking system operated under the order-by-order policy compete with those improvements possible, if the picker is allowed to flexibly switch among queuing trolleys when following the order-swapping policy?
(3) If the storage assignment cannot be adapted on short notice and a rearrangement of SKUs along the path specifically optimized for the order set to be processed the next morning is not possible, can the remaining optimization of the order processing (for a given storage assignment) still improve the picking performance considerably?

First, we address question (1) and explore the impact of duplicating SKUs along the picking path on the picking performance. On the positive side, storing SKUs multiple times along the picking path according to a shared storage policy increases the flexibility when selecting among the orders' escort paths. A higher level of SKU duplication increases the number of alternative escort paths per order and leads to more flexibility. On the negative side, a bad placed (duplicated) SKU may be in the way and prolong escort and return paths if they have to be passed without contributing to order completion. To explore this coherency, we test how an increasing storage capacity m along the picking path and, thus, an increasing level of SKU duplication impacts picking performance. Picking performance is still measured in the total walking distance of the picker when completing the given set of orders (see Sect. 1.2) and we apply our three-stage heuristic to optimize the storage assignment and the order processing process (JOSSAP) for the respective storage capacity m. For the test, we apply our instance generator described in Sect. 4.1 and derive 30 different instances with $n = 50$ orders for each of four scenarios, in which we have a small (large) number of different SKUs with $|U| = 10$ ($|U| = 50$) and small (large) orders with $[\underline{\gamma}; \overline{\gamma}] = [1; 3]$ ($[\underline{\gamma}; \overline{\gamma}] = [7; 10]$). Note that within Fig. 5 each scenario is identified by the shortcut $[\underline{\gamma}; \overline{\gamma}]/|U|$, so that [7;10]/10, for instance, denominates the scenario with large orders (i.e., drawn from $[\underline{\gamma}; \overline{\gamma}] = [7; 10]$) and few SKUs (i.e., $|U| = 10$). For each scenario, each instance is solved with our three-stage heuristic for different storage capacities m, which we vary by multiplying the number of SKUs $|U|$ with 1.0, 1.1, 1.3, 1.5, and 2.0. Thus, we vary the capacity from the most compact picking path where each SKUs

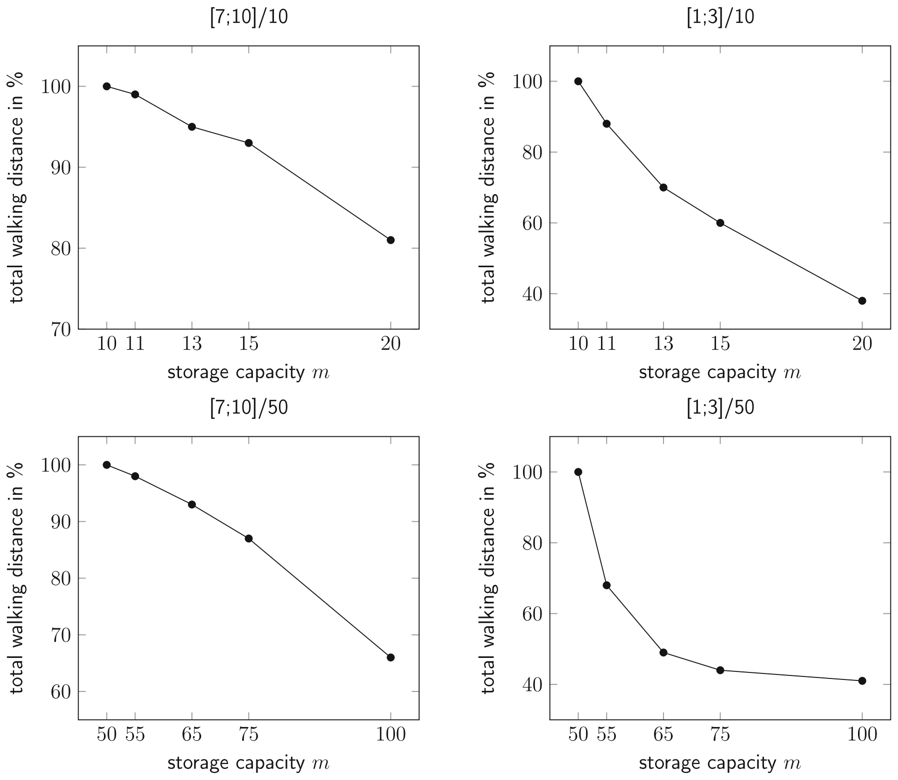

Fig. 5 Impact of storage capacity m on the picking performance

is stored exactly once up to a picking path being twice as long where each SKUs is stored two times. Note that it is also possible to add selected SKUs even more than twice and others only once. We leave exploring this additional flexibility up to future research. The results of this test are depicted in Fig. 5, where we relate the picking performance (i.e., the total walking distance of the picker) measured in % of the performance gained for the most compact picking path (where no SKU duplication is applied) to different storage capacities m. The following conclusions can be drawn from the results of Fig. 5:

- First of all, all four scenarios profit from additional storage capacity. Thus, an increasing level of SKU duplication positively impacts the picking performance, if the storage assignment and the order picking process are properly optimized.
- However, the extents of the performance gains vary among the scenarios. The largest improvement is possible, if the orders are small and demand just a few SKUs. These cases are depicted on the right two graphs of Fig. 5, where the number of SKUs per order is drawn from interval $[\underline{\gamma}; \overline{\gamma}] = [1; 3]$. In both cases for a picking path where every SKU is duplicated, only 40% of the picker distance compared to the default without SKU duplication arises. The goods news for practitioners is that the storage capacity needs not be doubled. Already smaller

capacities where only the most demanded SKUs are duplicated lead to impressive performance gains. For smaller orders, it is less likely that the same SKU appears in plenty other orders, so that it is easier to avoid badly placed SKUs that only prolong escort and return paths without contributing to the majority of orders.

- For larger orders, depicted within the left two graphs of Fig. 5, where the number of SKUs per order is drawn from interval $[\underline{\gamma}; \overline{\gamma}] = [7; 10]$, avoiding badly placed SKUs is much harder, because there is a lot of overlap among the SKU composition of orders. This effect is strengthened if the number of SKUs is small, depicted within the upper two graphs of Fig. 5, where the total number of SKUs is $|U| = 10$. However, even in the worst case (upper left graph of Fig. 5 with few SKUs, i.e., $|U| = 10$, and large orders, i.e., $[\underline{\gamma}; \overline{\gamma}] = [7; 10]$) a considerable improvement of 20% picking performance can still be realized by duplicating all SKUs along the picking path.

Next, we address question (2) and relate the performance gains enabled by our SKU duplication with the gains of the order-swapping work protocol, which has previously been investigated in the paper of Füßler et al. (2019). Recall (from Sect. 1.2) that the order-swapping policy enables a more flexible picking process. The picker is allowed to process neighboring stops of different trolleys before moving to another section of the line where other orders are waiting for processing. The order-by-order policy assumed in this paper, instead, enforces the picker to strictly complete order after order. In the next test, we explore whether a duplication of SKUs along the picking path can compensate for the loss of flexibility compared to the order-swapping work protocol. To do so, we apply the same instances as in our previous test and solve each instance within the four scenarios with our three-stage approach for a storage capacity of $m = |U| \cdot 2$, so that each SKU is duplicated. Furthermore, we compute the picker's total walking distance when allowing the order-swapping policy without SKU duplication for a random storage assignment. This case is solved with the multi-stage heuristic solution approach introduced by Füßler et al. (2019) for deriving the order sequence and scheduling the picker under the order-swapping policy. We relate the reduction of the picker's total walking distance of both competitors to the case where no SKU duplication is allowed solved with our three-stage approach. The latter defines our default (i.e., 100%). The results of our test are summarized in Fig. 6, and the following findings can be drawn from these results:

- The results reveal that in all four scenarios both approaches considerably improve the picking performance compared the default case where the order-by-order policy is applied without SKU duplication. In three out of four scenarios, however, the additional flexibility of the order-swapping policy leads to larger performance gains than our SKU duplication. Especially, if the order sizes are large (i.e., the number of SKUs per order is drawn from $[\underline{\gamma}; \overline{\gamma}] = [7; 10]$), the order-swapping policy turns out especially valuable, because (other than the order-by-order policy) the picker need not follow each order through large parts of the picking path, but can process clusters of queuing trolleys in different sections of the line.
- If the order sizes are small (i.e., the number of SKUs per order is drawn from $[\underline{\gamma}; \overline{\gamma}] = [1; 3]$), however, both competitors lead to similar results (especially in the case with few SKUs, i.e., $|U| = 10$, and small orders, i.e., $[\underline{\gamma}; \overline{\gamma}] = [1; 3]$). Here,

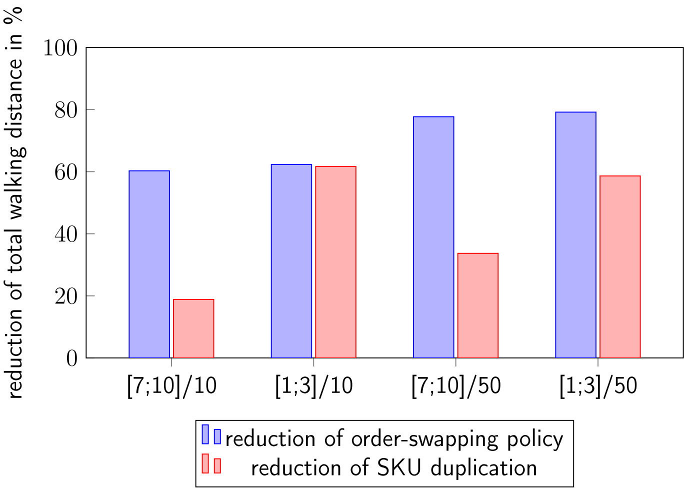

Fig. 6 Performance benchmark of the order-swapping policy versus our SKU duplication

the order-by-order policy does not suffer from badly placed SKUs and the picker's escort paths are short, so that both competitors lead to similar improvements.

It can be concluded that the order-swapping policy promises larger performance gains than our SKU duplication (except if we have few SKUs and small orders). However, picking performance is just one part of the trade-off. The price of the performance gains of our SKU duplication are additional space requirements for the enlarged picking path and a continuous rearrangement of the storage assignment. Realizing the order-swapping policy requires an additional information system that steers the picker on her movement from trolley to trolley and re-optimizes the picker schedule under real-time conditions (see Sect. 1.2). Thus, the negative aspects of both policies and their resulting additional investment and operational costs have to be integrated into the decision for a suited organization of a trolley line picking system.

Finally, we address question (3) and explore whether performance gains are still achievable by our SKU duplication, even if the storage assignment is not changed on short notice and specifically optimized for the order set to be processed the next day. Recall (from Sect. 1.2) that brick-and-mortar stores typically transmit their orders at the end of their opening hours, so that overnight there is enough time for an adaption of the storage assignment specifically optimized for the orders to be processed the next morning. There exist automated forklifts that can be applied to autonomously relocate the unit loads of SKUs along the picking path according to the optimization result of our JOSSAP problem. Alternatively, in the example systems of Swisslog for its CaddyPick system (Swisslog Logistics Automation 2011) the picking path is directly connected with an ASRS, so that a crane is fixedly installed to relocate the unit loads. Both solutions, however, cause additional investment costs, so that it is interesting to see whether there is still improvement possible if SKUs are duplicated, but the storage assignment is not specifically optimized for each single order set.

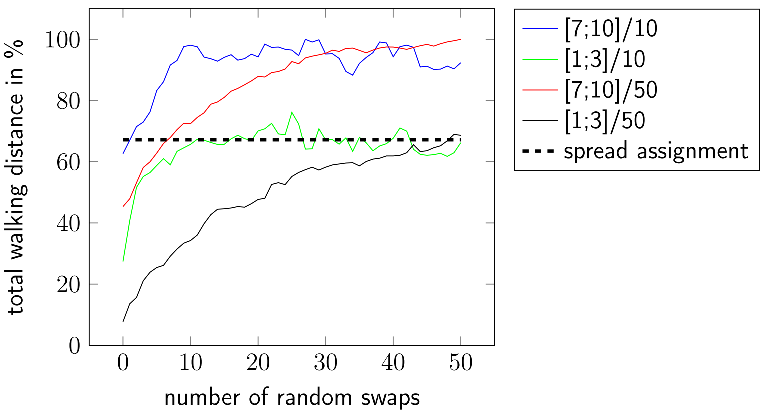

Fig. 7 Performance gains of SKU duplication if the storage assignment deviates from the optimal layout

To explore this matter, we solve the instances of our four scenarios elaborated above with our three-stage heuristic for a storage capacity of $m = |U| \cdot 2$, so that all SKUs are stored twice along the picking path. Solving the instances with our three-stage approach obtains the reference picking effort, if the storage assignment is specifically optimized for the current order set. Then, we apply an increasing number of swap moves altering the storage assignment more and more, so that the deviation to the optimal storage assignment becomes larger and larger. Once a specific number of swaps is executed, we solve the resulting JOSEPS instance for the given (swapped) storage assignment with our PRIO approach (see Sect. 2.3) and record the resulting total walking distance of the picker. In this way, we can record the decreasing performance gains the more the storage assignment deviates from the optimal one.

The more swaps of SKUs are executed, the more the resulting layout equals a random storage assignment. A random storage assignment cannot rule out that the same SKU is stored in neighboring storage positions. Since this is obviously no good idea, we also test a straightforward storage assignment that comes by without any information on the structure of the order set or even the frequencies with which each single SKU is ordered. Instead, we only spread the duplicate unit loads of the same SKU along the path. This is done by going through a random list of the $|U|$ SKUs an assigning the two unit loads of each SKU $i = 1, \ldots, |U|$ to storage positions i and $|U| + i$. We call this storage assignment the *spread assignment* and also obtain the total walking distance of the picker for this layout by our PRIO approach.

We relate the results of the spread assignment and the swapped layouts to the status quo of the distribution center for beverages reported in Sect. 1.1, where the order-by-order policy without SKU duplication is applied to a random order sequence (see also Füßler et al. (2019)). We record the total walking distance in percent derived by PRIO for different numbers of swap moves as well as the spread assignment and compare these results to the non-optimized (status quo) reference value defining 100%. The results of this test in our four scenarios are depicted in Fig. 7. These results suggest the following findings:

- In all four scenarios, our SKU duplication suffers and the performance gains become smaller compared to the status quo of a non-optimized order processing, the more the actual storage assignment deviates from the optimal one. The largest improvement arises for zero swap moves, which means that we have a storage assignment specifically optimized for the current order set. The more swap moves are executed, the larger the deviation from the optimal storage assignment and the smaller the performance gains in all four scenarios.
- On the positive side, however, random storage assignments, where up to 50 swap moves have been executed, still allow for a reduction of the total walking distance of the picker compared to the status quo of a non-optimized order processing. Thus, even if only the order processing is optimized by PRIO for a completely random storage assignment, performance gains can still be obtained.
- Again, we see the result of the previous tests confirmed: especially, if we have many SKUs, i.e., $|U| = 50$, and small orders, i.e., number of SKUs per order are drawn from $[\underline{\gamma}; \overline{\gamma}] = [1; 3]$, our SKU duplication leads to tremendous performance gains, even if the storage assignment is randomly determined. On the other hand, few SKUs, i.e., $|U| = 10$, and large orders, i.e., SKUs per order are drawn from $[\underline{\gamma}; \overline{\gamma}] = [7; 10]$, barely lead to performance gains compared to the non-optimized status quo, if the storage assignment is not part of the optimization problem.
- However, the good news is that even if the investment for relocating the SKUs along the picking path is saved and the storage assignment cannot be adapted on short notice, remarkable performance gains can still be realized just by optimizing the order processing with PRIO, at least if the storage assignments are not randomly determined. On average over all four scenarios an optimized order processing for the spread assignment, which evenly distributes the two unit loads per SKU over the picking path, only produces 67% of the total walking distance produced by the non-optimized status quo solution. Note that the reductions of the spread assignment do not considerably deviate among the four scenarios, so that we do not report each single result (to not overload the diagram). Further note that we do not apply swap moves to the spread assignment, so that the (dashed) graph remains on a constant level (i.e., is not affected by swaps).

It can be concluded that our duplication of SKUs along the picking path of a trolley line picking system can lead to large performance gains compared to non-optimized systems especially if short-term adaptions of the storage assignment are enabled. Without optimized storage assignments the performance gains of an optimized order processing are smaller, but still considerable.

5 Conclusion

This paper is dedicated to the trolley line picking system, where automated trolleys move along a picking path and receive requested SKUs from a human order picker. The picker operates according to the order-by-order policy and accompanies the current trolley until all requested SKUs are placed on it. Only then, she moves backwards against the flow direction of the system towards the successive trolley associated with the next order. We investigate two elementary optimization problems arising, if a duplication of SKUs is applied and unit loads of the same SKU are stored at multiple positions along the path according to a shared storage policy. First, we optimize the order processing if the storage assignment of SKUs along the picking path is given and then we also integrate the storage assignment as an additional decision variable. We investigate the computational complexity of the resulting optimization problems and derive suited heuristic solution algorithms for both settings. In our computational tests, we show that large improvements of the picking performance are possible when optimizing both decision tasks holistically compared to the status quo approach observed in a real-world distribution center for beverages where random storage assignments and order sequences are applied. However, even if the investment for a technical solution to adapt the storage assignment to the order set processed at the next day is saved, only optimizing the order processing for a given storage assignment that spreads the duplicated unit loads of the same SKU along the path still leads to considerable performance gains of more than 30% compared to the non-optimized status quo.

Future research should challenge our solution procedures. Especially, if the storage assignment is part of the decision, we have a very complicated holistic problem setting where four interdependent decisions need to be coordinated: selection of duplicated SKUs, storage assignment, selection of escort paths, and order sequence. Future research should try to improve our straightforward three-stage heuristic approach in order to derive better solutions for even larger problem instances. From the practitioner's point of view, it may be interesting to combine the order-swapping policy investigated by Füßler et al. (2019) with our SKU duplication. If SKUs are duplicated to increase the flexibility of escort path selection and the picker is allowed to flexibly swap among orders, even larger performance gains should be possible by such a combined approach. The resulting optimization problem, however, unites even more decision tasks, so that this problem setting should indeed be a tremendous algorithmic challenge.

Appendix A: Computational complexity of JOSEPS

Solving JOSEPS is a non-trivial task, which we prove in the following. We present a transformation from a strongly NP-complete variant of the satisfiability problem dubbed 1-in-3-Sat* introduced by Stephan and Boysen (2017). 1-in-3-Sat* is defined as follows:

1-in-3-Sat*: Given a set $V = \{v_1, v_2, \ldots, v_{|V|}\}$ of Boolean variables and a collection $C = \{c_1, c_2, \ldots, c_{|C|}\}$ of clauses over V, such that

- no clause contains a negated literal,
- each clause $c_j \in C$ has $|c_j| = 3$, and
- each variable $v_i \in V$ occurs either two or three times in clause set C.

Is there a truth assignment for V, such that each clause $c_j \in C$ has exactly one true literal?

Proof For an arbitrarily given instance of 1-in-3-Sat* the transformation is as follows: For every variable $v_i \in V$ of 1-in-3-Sat* we introduce 3 SKUs V_i, V_i', and V_i'', each

Table 8 Definition of subassignments

	Storage subassignment		
	Name	Length	SKU sequence
Front part	SV_0	1	V_0
	SV_i $(i = 1, 2, \ldots, \lvert V\rvert)$	5 or 6	V_i ; V_i'' ; C_j ; C_k ; C_l ; V_i'
	$SV_{\lvert V\rvert+1}$	1	$V_{\lvert V\rvert+1}$
Rear part	SV_0'	1	V_0'
	SV_i' $(i = 1, 2, \ldots, \lvert V\rvert)$	3	V_i ; V_i'' ; V_i'
	$SV_{\lvert V\rvert+1}'$	1	$V_{\lvert V\rvert+1}'$
	SE	1	E
	$\sum$	$6 \cdot \lvert V\rvert + 3 \cdot \lvert C\rvert + 5$	

Table 9 Order set

	Derived from 1-in-3-Sat* element	Orders in JOSEPS
Variable orders	v_i $(i = 1, 2, \ldots, \lvert V\rvert)$	$\{V_i; V_{i+1}\}$, $\{V_i; V_i'\}$, $\{V_i''; V_{i+1}\}$
Clause orders	c_j $(j = 1, 2, \ldots, \lvert C\rvert)$	$\{C_j\}$
Additional orders		$\{V_0\}$, $\{V_0'\}$, $\{E\}$

occurring exactly twice in the storage assignment of JOSEPS. Furthermore, for every clause $c_j \in C$ we introduce 1 SKU C_j occurring exactly 3 times. Additionally, we introduce 5 SKUs V_0, V_0', $V_{|V|+1}$, $V_{|V|+1}'$, and E, each occurring only once in the storage assignment of JOSEPS. Thus, in total there are $3 \cdot |V| + |C| + 5$ SKUs, each occurring at most three times in the storage assignment. Thus, the entire picking path has a length of $m = 6 \cdot |V| + 3 \cdot |C| + 5$. From left to right (e.g., from storage positions 1 to m), the storage assignment can be divided into $2 \cdot |V| + 5$ subassignments

$$\underbrace{SV_0 \,;\, SV_1 \,;\, SV_2 \,;\, \ldots \,;\, SV_{|V|} \,;\, SV_{|V|+1}}_{\text{front part}} \,;\, \underbrace{SV_0' \,;\, SV_1' \,;\, SV_2' \,;\, \ldots \,;\, SV_{|V|}' \,;\, SV_{|V|+1}}_{\text{rear part}} \,;\, SE,$$

where the subassignments are defined in Table 8. Note that SKU C_j, $j = 1, 2, \ldots, |C|$ occurs in subassignment SV_i, $i = 1, 2, \ldots, |V|$ if and only if variable $v_i \in V$ occurs in the clause $c_j \in C$ of the 1-in-3-Sat* instance. Thus, subassignments SV_i, $i = 1, 2, \ldots, |V|$ consist of either 5 or 6 SKUs if the corresponding variable v_i occurs 2 or 3 times in the clause set C, respectively. (The succession of the 2 or 3 SKUs C_j, C_k, C_l within SV_i is not relevant.)

Moreover, there are $3 \cdot |V| + |C| + 3$ orders according to Table 9 and the order picker starts the picking process at position 0.

The question we ask is whether we can find a solution to the corresponding JOSEPS instance (i.e., a selection of one escort path for every order and an order sequence) resulting in a total walking distance of at most $Z = m + 2 \cdot |V| + 2 \cdot |C|$. Obviously, this

Table 10 Alternative selections of escort paths for a triple $\{V_i; V_{i+1}\}, \{V_i; V_i'\}, \{V_i''; V_{i+1}\}$ of *variable orders* and the associated additional distances (compared to alternative (ii))

Number of alternative	Order(s) to be picked in the Front part	Rear part	Lower bound for further distances (compared to alternative (ii))
(i)	–	$\{V_i; V_{i+1}\}, \{V_i; V_i'\}, \{V_i''; V_{i+1}\}$	$2 \cdot 3$
(ii)	$\{V_i; V_{i+1}\}$	$\{V_i; V_i'\}, \{V_i''; V_{i+1}\}$	0
(iii)	$\{V_i; V_i'\}$	$\{V_i; V_{i+1}\}, \{V_i''; V_{i+1}\}$	$2 \cdot 1$
(iv)	$\{V_i''; V_{i+1}\}$	$\{V_i; V_{i+1}\}, \{V_i; V_i'\}$	$2 \cdot 1$
(v)	$\{V_i; V_i'\}, \{V_i''; V_{i+1}\}$	$\{V_i; V_{i+1}\}$	$2 \cdot 2$ or $2 \cdot 3$ (*)
(vi)	$\{V_i; V_{i+1}\}, \{V_i''; V_{i+1}\}$	$\{V_i; V_i'\}$	$2 \cdot 3$ or $2 \cdot 4$ (*)
(vii)	$\{V_i; V_{i+1}\}, \{V_i; V_i'\}$	$\{V_i''; V_{i+1}\}$	$2 \cdot 3$ or $2 \cdot 4$ (*)
(viii)	$\{V_i; V_{i+1}\}, \{V_i; V_i'\}, \{V_i''; V_{i+1}\}$	–	$2 \cdot 7$ or $2 \cdot 9$ (*)

(*) ... depending on whether variable v_i occurs in 2 or 3 clauses of C, respectively

transformation can be done in pseudo-polynomial time and meets the two conditions listed in Theorem 1.

Next, we show that there is a feasible solution to our JOSEPS instance with an objective value not larger than Z if and only if there is a satisfying truth assignment for the 1-in-3-Sat* instance. Due to the *additional orders* $\{V_0\}$, $\{V_0'\}$ and $\{E\}$ the picker at least has to walk from position 0 (the start position) to m (the rightmost storage position), which prohibits objective values smaller than m. Furthermore, for every *variable order* $\{V_i; V_{i+1}\}$, $\{V_i; V_i'\}$, $\{V_i''; V_{i+1}\}$ ($i = 1, 2, \ldots, |V|$) we each have 3 potential escort paths (i.e., $|\Gamma_{\{V_i;V_{i+1}\}}| = |\Gamma_{\{V_i;V_i'\}}| = |\Gamma_{\{V_i'';V_{i+1}\}}| = 3$). Specifically, for every *variable order* we each have one short escort path in the front part and in the rear part of the storage assignment and a very long escort path from the front to the rear part covering the half of the picking path. Obviously, choosing the latter escort path for one or more *variable orders* leads to an overall walking distance of more than Z, so that in the following we focus the former ones. Thus, the main question is which of the *variable orders* to be picked while passing the front part of the storage assignment and which ones while passing the rear part. Starting with order V_0, picking the orders $\{V_i; V_{i+1}\}$ ($i = 1, 2, \ldots, |V|$) while passing the front part of the storage assignment, proceeding with V_0' and picking orders $\{V_i; V_i'\}$ and $\{V_i''; V_{i+1}\}$ ($i = 1, 2, \ldots, |V|$) in the rear part of the storage assignment before ending with order $\{E\}$ leads us to a lower bound for the walking distance of $m + 2 \cdot |V|$. This way, the order picker has to walk 1 distance unit backwards (and, therefore, 1 additional unit forwards) for $|V|$ times in the rear part, which in total cause an extra distance of $2 \cdot |V|$. As listed in Table 10, any deviation from this strategy causes further distances.

The remaining question is whether we are able to also pick the $|C|$ (pairwise different and single-item) *clause orders* without exceeding the remaining budget for picking distance of $2 \cdot |C|$. Note that only alternatives (i) and (v) allow for picking (2 or 3) *clause orders* when passing the respective subsequence SV_i in the front part without causing more than 2 extra distance units per *clause order* (for alternative (i) this holds only if the corresponding Boolean variable v_i occurs in 3 clauses). Thus, answering the remaining question is equivalent to identifying triples $\{V_i; V_{i+1}\}$, $\{V_i; V_i'\}$ and $\{V_i''; V_{i+1}\}$ $(i = 1, 2, \ldots, |V|)$ of *variable orders* to be picked according to alternative (i) or (v) instead of (ii) in such a manner that all SKUs C_j, $j = 1, 2, \ldots, |C|$ are contained in the respective subassignments SV_i in the front part at least once (i.e., all *clause orders* can be picked), but also not more than once (i.e., without exceeding the remaining budget $2 \cdot |C|$). When we now interpret all triples to be picked according to alternative (i) or (v) (alternative (i) only in the case of 3 occurrences) as TRUE-variables of 1-in-3-Sat* and all triples to be picked according to alternative (ii) as FALSE-variables it becomes obvious that the solutions to both problem instances can directly be transferred to each other and Theorem 1 holds. □

Appendix B: Computational complexity of JOSSAP

Proof To prove Theorem 2, we only have to adapt the ideas from the proof of Theorem 1 by introducing two additional very large order sets making

- the succession of subassignments SV_0 ; SV_1 ; … ; $SV_{|V|+1}$; SV_0' ; SV_1' ; … ; $SV_{|V|+1}$; SE from the proof of Theorem 1, as well as
- the succession of SKUs within these subassignments

obligatory. Furthermore, we have to adapt the bound for the new objective value to the modified order set. However, these adaptions are truly straightforward, so that we abstain from a detailed description. □

Appendix C: On the correlation of JOSSAP with and without return to the depot after the final order

In line with a previous paper on trolley line picking (Füßler et al. 2019), we assume that processing a given order set ends once the picker has reached the final picking position of the last order. In addition to being comparable to the previous paper, the main reason for this assumption is that order picking is an exhausting task, so that (at least in Europe) pickers have the right for a five to ten minutes break per hour. Thus, we assume that such a break starts right after reaching the final pick position of the last order. However, if we assume that the picker directly proceeds with a new order set, then ending a JOSSAP schedule once the picker has completed the last order and has returned to the depot seems the better choice. Then, the picker can instantly start processing the next order set at the depot. Fortunately, the required adaptions to alter our three-stage solution procedure for a final depot return are truly straightforward. The only necessary change is that we have to swap our modified Gilmore–Gomory approach for order sequencing with the original approach elaborated in Gilmore and Gomory (1964). The original considers a return to the start point and is also faster, i.e., the runtime is reduced by factor n.

However, in the following we present the result of a computational test, which shows that the correlation among both objective values is very close to one, i.e., 0.9975 over

Table 11 Pearson correlation coefficient for JOSSAP with and without final return

n	$\|U\|$	cor	n	$\|U\|$	cor	n	$\|U\|$	cor
10	10	0.9970	30	10	0.9988	50	10	0.9990
10	30	0.9871	30	30	0.9986	50	30	0.9995
10	50	0.9734	30	50	0.9972	50	50	0.9984
Total		0.9485	Total		0.9953	Total		0.9975

all instances, so that we conjecture that all our findings for the case without a final depot return also hold if a final depot return is required.

Specifically, we solve all our test instances (see Sect. 4.1) with our three-stage heuristic and obtain the solutions without final depot return. To account for the picker's final return to the depot, we add the number of SKUs to her walking distance, because in the worst case she may be at the very last storage position when ending the final order. Furthermore, we implemented the above adaptions and apply the modified three-stage heuristic for the case with a final depot return. Table 11 reports on Pearson's correlation coefficient (column 'cor') of the objective values for both problem versions.

It can be concluded that the solutions obtained by both approaches are almost identical, which has been confirmed by a visual inspection of the resulting storage assignments. Thus, we are positive that our results also hold for the case with final depot return.

References

Aarts EHL, Korst JHM, van Laarhoven JM (1997) Simulated annealing. In: Aarts EHL, Lenstra JK (eds) Local search in combinatorial optimization, Chichester, pp 91–120

Agatz NA, Fleischmann M, van Nunen JA (2008) E-fulfillment and multi-channel distribution—a review. Eur J Oper Res 187:339–356

Azadeh K, De Koster MBM, Roy D (2018) Robotized warehouse systems: Developments and research opportunities. Transp Sci 53(4):917–945

Bartholdi JJ, Hackman ST (2017) Warehouse and distribution science. Release 0.98. Supply Chain and Logistics Institute

Bartholdi JJ III, Platzman LK (1986) Retrieval strategies for a carousel conveyor. IIE Trans 18:166–173

Boysen N, Stephan K (2013) The deterministic product location problem under a pick-by-order policy. Discrete Appl Math 161:2862–2875

Boysen N, De Koster R, Weidinger F (2019) Warehousing in the e-commerce era: a survey. Eur J Oper Res 277:396–411

Boysen N, De Koster R, Füßler D (2019) The forgotten sons: warehousing systems for brick-and-mortar retail chains. Working Paper Friedrich-Schiller-Universität Jena

Burkard RE, Deineko VG, van Dal R, van der Veen JA, Woeginger GJ (1998) Well-solvable special cases of the traveling salesman problem: a survey. SIAM Rev 40:496–546

Daniels RL, Rummel JL, Schantz R (1998) A model for warehouse order picking. Eur J Oper Res 105:1–17

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

Füßler D, Fedtke S, Boysen N (2019) The cafeteria problem: order sequencing and picker routing in on-the-line picking systems. OR Spectrum 41:727–756

Gilmore PC, Gomory RE (1964) Sequencing a one state-variable machine: a solvable case of the traveling salesman problem. Oper Res 12:655–679

Hübner A, Kuhn H, Wollenburg J (2016) Last mile fulfilment and distribution in omni-channel grocery retailing: a strategic planning framework. Int J Retail Distrib Manag 44:228–247

Hübner A, Holzapfel A, Kuhn H (2016) Distribution systems in omni-channel retailing. Bus Res 9:255–296
Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
Melacini M, Perotti S, Tumino A (2011) Development of a framework for pick-and-pass order picking system design. Int J Adv Manuf Technol 53:841–854
Pan JCH, Wu MH (2009) A study of storage assignment problem for an order picking line in a pick-and-pass warehousing system. Comput Ind Eng 57:261–268
Pan JCH, Shih PH, Wu MH (2015) Order batching in a pick-and-pass warehousing system with group genetic algorithm. Omega 57:238–248
Stephan K, Boysen N (2017) Crane scheduling in railway yards: an analysis of computational complexity. J Sched 20:507–526
Swisslog Logistics Automation (2011) Denner, Schweiz. https://www.youtube.com/watch?v=ltguAbBRld8. Accessed March 2019
Swisslog (2017) CaddyPick: picking made easy. https://www.swisslog.com/de-de/kontakt/downloads?mediaItem=5551B202AE7541FA9662BDC8A95B7041. Accessed March 2019
Weidinger F, Boysen N (2018) Scattered storage: how to distribute stock keeping units all around a mixed-shelves warehouse. Transp Sci 52:1412–1427
Weidinger F (2018) Picker routing in rectangular mixed shelves warehouses. Comput Oper Res 95:139–150
Weidinger F, Boysen N, Schneider M (2019) Picker routing in the mixed-shelves warehouses of e-commerce retailers. Eur J Oper Res 274:501–515
Yu M, De Koster R (2008) Performance approximation and design of pick-and-pass order picking systems. IIE Trans 40:1054–1069